J Syst Sci Complex (2024) XX: 1–23

# Enhancing Multivariate Time Series Forecasting: A Novel Approach with Mallows Model Averaging and Graph Neural Networks*

**ZHANG Haili· WANG Jiawei · LIU Zhaobo · DONG Hailing**

DOI:
Received: 30 January 2024 / Revised: x x 2024


**Abstract** Multivariate time series forecasting holds substantial practical significance, facilitates precise predictions, and informs decision-making. The complexity of nonlinear relationships and the presence of higher-order features in multivariate time series data have sparked a burgeoning interest in leveraging deep learning approaches for such forecasting tasks. Existing methods often use pre-scaled neural networks, whose reliability and generalization can pose a challenge. In this study, we present an instance-wise graph-based Mallows model averaging (IGMMA) framework for the prediction of multivariate time series. The framework incorporates a model averaging module into the network. It assigns weights to multiple neural networks to improve the robustness of the prediction. In addition, the network loss function is modified based on the Mallows criterion, where penalties are imposed on the parameters and the weights separately. We use our proposed method to predict multicommodity futures prices, and our empirical results show that IGMMA has superior predictive accuracy even when small neural networks are used. This indicates that the model averaging module significantly reduces the parameters required for deep learning training, which enables the training of multiple small models as an alternative to training a large model.

**Keywords** Graph neural networks, model averaging, multivariate time series.

ZHANG Haili
Institute of Applied Mathematics, Shenzhen Polytechnic University, Shenzhen 518055, China.
Email: zhanghl@szpu.edu.cn
WANG Jiawei · LIU Zhaobo (Corresponding author)
Institute for Advanced Study, Shenzhen University, Shenzhen 518060, China. Email: liuzhaobo@szu.edu.cn
DONG Hailing (Corresponding author)
Science of Mathematics College, Shenzhen University, Shenzhen 518060, China. Email: hailing@szu.edu.cn
*This research was supported by the NNSF of China under Grants 12371448, the National Key R & D Program of China under Grants 2023YFE0126800, the Guangdong Provincial Ordinary Universities Youth Innovative Talent Project (Natural Science) under 2023KQNCX064, the Research Foundation of Shenzhen Polytechnic University under Grant 6023312034K, and the Post-doctoral Later-stage Foundation Project of Shenzhen Polytechnic University under Grant 6023271021K.
◇ *This paper was recommended for publication by Editor XX.*

## 1 Introduction

Multivariate time series prediction has a wide range of applications in various domains, including traffic road load [1], power consumption [2], and financial asset price prediction [3]. The dynamic nature of multivariate time series data, with its complex dependencies and changing patterns, presents a significant challenge for traditional forecasting models. Deep learning has become a solution to this challenge because it can capture intricate and complicated patterns and relationships in the data [4].

In recent years, deep learning techniques have been used extensively for multivariate time series forecasting because of advances in computing power and data processing technology [5–7]. These approaches involve the construction of neural network models that automatically learn data patterns and make predictions about multivariate time series. For example, in the case of futures prices, Wang and Gao [5] used the Dalian Commodity Exchange dataset to build a long short-term memory (LSTM) neural network model to predict high and low soybean futures prices. Similarly, Sun et al. [6] employed deep LSTM as their primary architecture and expanded market trading data to predict trends in the futures market. Ji et al. [7] used the hybrid ARIMA-CNN-LSTM model to predict futures prices based on carbon trading.

However, the neural network models mentioned above often require a preset model scale, which is essentially equivalent to model selection. Model selection often leads to unstable estimates, where even slight perturbations in the data can result in the selection of a completely different model [8, 9]. Furthermore, model selection may lose valuable information [10]. Some model selection methods are intended to select the model that best reflects the true data generation process but may not necessarily provide reliable estimation or predictive performance [11]. Therefore, selecting models based on training set data makes it difficult to guarantee that the model has good generalization ability on unknown data, thereby limiting the models' performance in practical applications. To address this issue, this study proposes a deep learning framework based on model averaging.

Model averaging, which is an alternative to model selection, involves the weighted averaging of estimates [12] or predictions from different models using specific weights. This approach prevents the selection of poor models for prediction, thereby improving prediction reliability [13]. By weighting multiple neural network models of different scales during the training process, this study aims to retain the advantages of each model, reduce overfitting risks, and improve the robustness of predictions and the generalization performance of models on different datasets. Consequently, this framework achieves superior performance in a variety of scenarios.

One important aspect of model averaging methods pertains to the selection criterion for assigning weights. Various approaches have been proposed, including information criteria, Mallows model averaging, focused information criterion, optimal weight choice (OPT), Jackknife model averaging methods, and cross-validation methods. In this paper, we focus on a comprehensive review of model averaging techniques based on the Mallows criterion.

The Mallows model averaging method, which was proposed by Hansen [14], is a computationally efficient method for selecting the weights of a combined nested linear model by minimizing

the Mallows criterion. This method was found to have asymptotic optimality for model averaging estimation under the assumption of a discrete weight space. Later, Wan, Zhang, and Zou [15] generalized the conclusion in [14] and proved that model averaging estimation based on the Mallows criterion still has asymptotic optimality under continuous weight space and for general combined models. For heteroscedasticity, Liu and Okui [16] proposed a modified Mallows model averaging method based on [14]. For more complex statistical models, such as threshold autoregressive models, varying coefficient partial linear models, and orthogonal kriging models, please refer to the references [17], [18] and [19]. With the continuous development and application of machine learning techniques in economics and finance [20, 21], many studies have begun using machine learning techniques for financial forecasting [22, 23]. Qiu et al. [24] extended the Mallows model averaging theory to combine it with traditional machine learning methods and proved that model averaging based on this framework could produce more accurate prediction results than could traditional machine learning techniques alone.

Overall, the current research on model averaging methods is mainly focused on statistical models or traditional machine learning models; there is a notable lack of research investigating the integration of model averaging methods into the deep learning framework. It is worth exploring whether model averaging methods can improve the predictive ability of deep neural networks, which have a large parameter space and high nonlinearity. Therefore, this study innovatively proposes relevant algorithms and conducts empirical analysis to study the feasibility of model averaging methods in multicommodity futures price sequence prediction problems. Because of the specific characteristics of the futures market, such as high price volatility, trading frequency, and high leverage [6], prediction models need to have high accuracy and real-time capability to effectively predict price changes in futures assets.

In this study, we propose an instance-wise graph-based Mallows model averaging (IGMMA) framework that incorporates model averaging methods into a deep learning framework for the prediction of multivariate time series. This framework uses deep graph neural networks for feature extraction and applies Mallow's criteria to weight models that correspond to different feature numbers. We conducted ablation experiments using the model averaging module and empirical studies using the modified IGMMA framework with the optimized objective function. The main contributions of this study are as follows:

1) Unlike the existing literature on model averaging that considers statistical or machine learning models, this study innovatively combines model averaging methods with the deep learning framework. This method may provide a solution to address the overfitting problem in deep learning and improve the generalization of graph neural networks.

2) Through empirical analysis, we find that after incorporating the model averaging module, the IGMMA framework can achieve higher prediction accuracy with fewer parameters than several typical deep learning prediction methods. This means that model averaging methods can fundamentally reduce the difficulty of training deep neural networks and have broad application prospects in other problems that use deep learning.

This paper is structured as follows. In Section 2, we present the IGMMA framework and

algorithm. We then conduct empirical analysis to evaluate the algorithm's prediction performance and optimize hyperparameters in Section 3. Finally, Section 4 provides a summary of the work and outlines the prospects of this study.

## 2 IGMMA framework

In this section, we introduce our proposed method for multivariate time series prediction. A multivariate time series prediction considers the interrelationships and influences between multiple variables in analyzing time series data. Each variable value at time $t$ depends not only on its historical values but also on the historical values of other variables. Due to the similarity of external environments or the periodicities in time series, one variable may follow a sequence that is similar to the historical sequences of other variables. This means that there may be mutual dependencies between different variables at different time stamps, and that predicting the interdependent relationships between the historical and current time series is valuable.

Let $\{x^t\}, t \in \{1, 2, \cdots, T\}$ denote an observed multivariate time series of dimension $n > 1$ and length $T \geq 1$. Assume that $x^t$ relies heavily on observations from time $t-d$ to $t-1$. So we define $d$ as the lookback step and a subject at time $t$ to be denoted as $X^t = \{x^{t-d}, x^{t-d+1}, \cdots, x^t\}$, where $x^{t-j} \in \mathbb{R}^n$. The observation sequence of the $i$th variable in $\{x^t\}$ from time $t-d$ to $t$ is denoted as $X_i^t = \{x_i^{t-d}, x_i^{t-d+1}, \cdots, x_i^t\}$, where $x_i^t \in \mathbb{R}^d$, and it includes the prediction label $x_i^{t+h}$. The objective is to predict the future sequence value $x^{t+h}$ at time $t+h$ using the data collected up to time $t$, where $h$ denotes a specified forward prediction step determined by the real-time series data and task context.

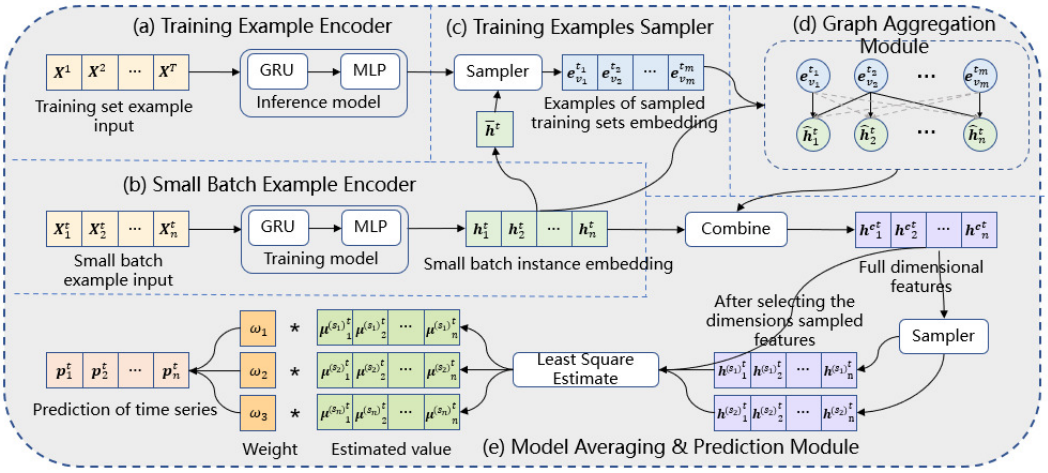


**Figure 1** The overall architecture diagram of IGMMA framework

This study is based on related work on instance-level multivariate time series prediction for graphs by Xu et al. in 2021 [25], and innovatively proposes an IGMMA framework. The specific structure of the IGMMA framework is shown in Figure 1. The main concept of this framework

is that it combines current time series information with historical information from different variables. We make predictions by weighting multiple candidate models using model averaging techniques. This process involves the following steps:

1) Learn the training and mini-batch instances and convert them into embedding forms. To address the complexity and implementation challenges of using only training instances, a sampler is used to select the most relevant historical information based on similarity calculations between each mini-batch and training instance.

2) Utilize the graph aggregation module to aggregate the information from the sampled training instances into mini-batch instances. The aggregated information, along with the mini-batch instances, is concatenated to obtain a feature representation.

3) Select the generated features and construct multiple models with different feature dimensions. An optimal linear estimation of the original and randomly selected features is obtained using the least squares method algorithm. The weights are then used to assign importance to each estimate. This strategy provides a balanced and compromised solution in terms of prediction effectiveness and model complexity, and it replaces the linear layer in the original framework.

4) Modify the objective function based on the Mallows criterion to make it more suitable for model averaging methods. This adjustment enhances the optimization process. The structure and notation definitions of this framework will be further elaborated in the following sections.

### 2.1 Feature extraction and graph module

The training instance encoder aims to learn the representations of sequence instances in the training set $\mathcal{D}_{train} = \{X^1, X^2, \cdots, X^T\}$, where $X^i \in \mathbb{R}^{n\times d}$ and $T$ represents the number of timestamps in the training set. The features of the training instances in $\mathcal{D}_{train}$ are input into a network structure consisting of a GRU and a three-layer MLP, obtaining the embeddings $E = \{E^1, E^2, \cdots, E^T\}$, where $E^i \in \mathbb{R}^{n\times l}$ and $l$ is the dimension of the output of the MLP.

For each mini-batch $\mathcal{D}_{train}$, we input the features $X^t = \{X_1^t, X_2^t, \cdots, X_n^t\}$ of $n$ instances $\{v_1^t, v_2^t, \cdots, v_n^t\}$ at time $t$ into a network with the same structure as the training instance encoder, undergoing a similar feature extraction process. The output is the embedding $H^t = \{h_1^t, h_2^t, \cdots, h_n^t\}$ of the mini-batch instances, where $h_i^t \in \mathbb{R}^l$.

To avoid the significant computational cost of directly aggregating information from training instances to mini-batch instances, we use a training instance sampler to sample the most relevant training instances $D_{train}^{sample}$ for each mini-batch from $D_{train}$. Firstly, the mean $\overline{E} = \{\overline{E}^1, \overline{E}^2, \cdots, \overline{E}^T\}$ of the embeddings of each timestamp's training instances and the mean of the embeddings of the mini-batch instances $H^t = \{\overline{h}_1^t, \overline{h}_2^t, \cdots, \overline{h}_n^t\}$ are calculated according to

$$\overline{E}^i = \frac{1}{n}\sum_{j=1}^{n} E_j^i, \quad \overline{h}^t = \frac{1}{n}\sum_{j=1}^{n} h_j^t,$$

where, $\overline{E}^i \in \mathbb{R}^l$ is the mean embedding of the training instances at time $i$, and $\overline{h}^t \in \mathbb{R}^l$ is the mean embedding of the mini-batch instances $H^t$. Subsequently, we calculate the cosine similarity between each timestamp's embedding $\overline{E}^i$ and $\overline{h}^t$ to find the most relevant instance to each mini-batch instance and select the $k$ embeddings at timestamps closest to $\overline{h}^t$ based on similarity. Subsequently, the training instances within these $k$ timestamps are sampled to create the sampled training instances with a total number $m = n \times k$.

In the graph module, a bipartite graph is constructed to capture the mutual connection and dependence between the sampled training instances $D_{train}^{sample}$ and the mini-batch instances $D_{batch}$. Given $m$ sampled training instance embeddings $E_s = \{e_{v_1}^{t_1}, e_{v_2}^{t_2}, \cdots, e_{v_m}^{t_m}\}$, and $n$ mini-batch instance embeddings $H^t = \{h_1^t, h_2^t, \cdots, h_n^t\}$. The information of the sampled training instances is aggregated with the mini-batch instances using cosine similarity between $E_s$ and $H^t$ as the initial aggregation weight:

$$A_{ij} = Cosine(h_i^t, e_{v_j}^{t_j}) = \frac{W_h h_i^t \cdot W_e e_{v_j}^{t_j}}{\|W_h h_i^t\| \cdot \|W_e e_{v_j}^{t_j}\|},$$

where $W_h$ and $W_e$ are two mapping matrices.

Due to the potential issues of over-smoothing in fully connected graphs, the aggregation weights $A_{ij}$ from all training instances to the mini-batch instances $v_i$, where $j \in [1, m]$, retain only the top $N$ weights and mask all other weights as 0. Finally, using the masked adjacency matrix $\widehat{A}$, information is aggregated from the closest $N$ training instances to each mini-batch instance:

$$\widehat{h}_i^t = \frac{1}{\mathcal{N}_i} \sum_{j \in \mathcal{N}_i} \widehat{A}_{ij} W_e e_{v_j}^{t_j},$$

where $\mathcal{N}_i$ is the set of top $N$ closest training instances for the mini-batch instance $v_i^t$. The $\widehat{h}_i^t$ is the information aggregation from the historical training instances to the instance $v_i^t$ at time $t$.

### 2.2 Model averaging module

The goal of the model averaging module is to enhance the efficiency of converting extracted feature information into final prediction results. This is achieved by combining the least squares estimates of multiple features to obtain the outcome. This approach is known to enhance the accuracy and robustness of the model, particularly in situations where overfitting may occur. Weighted averaging, simple averaging, and voting are methods that are commonly used in model averaging. In this study, model averaging module employs the weighted averaging method.

We combine the information from the graph aggregation module with small batch instances to jointly make predictions about future time series. The aggregated information obtained from the training set instances, $\widehat{h}_i^t$, serves as the first half, and the small batch instances, $h_i^t$, serve as the second half of the filling. By connecting these two halves, we obtain the concatenated feature vector $h_c = Concat(\widehat{h}_i^t, h_i^t)$, where Concat denotes the concatenation operation of multiple vectors. Since both $\widehat{h}_i^t$ and $h_i^t$ are $l$-dimensional vectors, i.e., $\widehat{h}_i^t, h_i^t \in \mathbb{R}^l$, where $l$ is the output dimension of MLP, the feature vector $h_c$ becomes a $2l$-dimensional vector, i.e., $h_c = \mathbb{R}^{2l}$. This

vector captures the feature information obtained from aggregating the current small batch instances and the training set instances up to the current point in the network structure.

### 2.2.1 Model averaging estimator for graph models

Following the idea of model averaging, let $S$ denote the total number of candidate models. We randomly select $\frac{p_s}{2}$ features from each of the two components $\widehat{h}_i^t$ and $h_i^t$ to obtain multiple features corresponding to different $p_s$ by combining them. Here, $p_s$ represents the scale of the $s$th candidate model in the model averaging process, where $1 \leq s \leq S$. We denote the sampled features as $h^{(s)}$.

For multiple different features $h^{(s)}$, which correspond to models with different numbers of dimensions and randomly selected dimensions, we perform weighted averaging of the models to increase the diversity of model features. Let $\widehat{\mu}^{(s)}$ be the estimate based on the $s$th candidate model, and use the least squares method for multiple sequences to estimate its parameters. Specifically, the least squares estimate of $\beta^{(s)}$ is calculated as $\beta^{(s)} = (h'^{(s)}h^{(s)})^{-}h'^{(s)}Y$. Since it is strictly required that no future information should be used during model training, and no label values should be used as feature information to participate in the training process, according to the least squares estimation method, we obtain the current small batch of sequence instances $v^t = \{v_1^t, v_2^t, \cdots, v_n^t\}$ under all variables, where $v^t \in \mathbb{R}^n$, and slice its features $X^t = \{x^{t-d}, x^{t-d+1}, \cdots, x^t\}$, $X^t \in {}^{d\times n}$, to obtain the observation value $x^t$ at the last time stamp $t$, i.e., the $n$-dimensional vector at the end of $X^t \in \mathbb{R}^n$. Let $X'$ be the matrix stacked according to the time stamp, $X' = \{x^{d+h-1}, x^{d+h}, \cdots, x^T\}$, whose shape should be the same as $Y$, and use it to replace $Y$ in the least squares estimation method. It is worth noting that the input data in this case is the observation value of the original sample, rather than $h^{(s)}$ as shown in the least squares estimation formula. At this point, the calculation method of the estimated value is $\beta^{(s)} = (h'^{(s)}h^{(s)})^{-}h'^{(s)}X'$, then $\widehat{\mu}^{(s)} = h^{(s)}\widehat{\beta}^{(s)}$.

In the definition process of the network model based on model averaging, in addition to the trainable parameters defined in the GRU, multilayer MLP, and other linear layers, an additional vector is added that needs to participate in gradient updates. Its representation is shown as

$$w = (w_1, w_2, \cdots, w_S)' \in \mathcal{W} = w \in [0,1]^S : \sum_{s=1}^{S} w_s = 1. \tag{1}$$

Then the corresponding average model estimate is

$$\widehat{\mu}(w) = \sum_{s=1}^{S} w_s \widehat{\mu}^{(s)} = \sum_{s=1}^{S} w_s P^{(s)} X' \triangleq P^{(w)} X'.$$

Finally, the one-dimensional output we obtain is the predicted value of the time series, and its calculation method is shown in the formula

$$\mathbf{p}_i^t = \sum_{s=1}^{S} w_s h^{(s)} \widehat{\beta}^{(s)} = \sum_{s=1}^{S} w_s h^{(s)} P^{(s)} X' \equiv \sum_{s=1}^{S} w_s h^{(s)} (h'^{(s)} h^{(s)})^{-} h'^{(s)} X'.$$

### 2.2.2 Mallows type model averaging criterion for graph models

Based on the model averaging module proposed in the previous section, it is employed to replace the conventional approach of utilizing linear layers for dimensionality reduction and mapping high-dimensional features to the final output in deep learning methods. However, solely relying on basic loss terms such as mean absolute error (MAE), mean squared error (MSE), and the summation of $L_2$ regular terms as the model's loss function fails to accurately represent the optimization objective. Although the $L_2$ norm scales all parameters in the model by predetermined weight decay coefficients to prevent excessively large parameter values and constrain the model's size and complexity, it uniformly penalizes all parameters and lacks targeted penalization for relatively large-scale candidate models. Consequently, the optimization outcome tends to favor larger models. Hence, customization of the objective function is imperative to better align the optimization objective with the Mallows criterion.

Qiu et al.[24] applied the Mallows-type model averaging method to machine learning. The weight vector is estimated by minimizing the Mallows criterion, which is defined as follows:

$$C_n(w) = \|\mathrm{y} - P(w)y\|^2 + 2\widehat{\sigma}^2(w)\mathrm{tr}P(w)$$

where $P_{tt}(w)$ is matrix $P(w)$'s $t$ diagonal term and $\widehat{\sigma}^2(w)$ is the estimator of the mean error variance.

To make the loss function more consistent with the actual module we introduced and the specific meaning of this deep learning multivariate time series prediction task, the loss function based on the Mallows criterion is defined as follows:

$$\mathcal{L}' = \frac{\sum_{v_i^t \in \mathcal{D}_{batch}} \left(\mathbf{p}_i^t - \mathbf{x}_i^{t+h}\right)^2}{|\mathcal{D}_{batch}|} + 2\frac{\widehat{\sigma}^2(w)\mathrm{tr}P(w)}{|\mathcal{D}_{batch}|} + \lambda \|\Theta\|_2^2 . \tag{2}$$

Here, $\mathcal{D}_{batch}$ is the sequence instances in each small batch, $\mathrm{p_i^t}$ and $x_i^{t+h}$ are the predicted values at time $t$ and the labels of the variables $v_i$, respectively. In the penalty term, $\lambda$ represents the regularization coefficients, $\Theta$ represents all the parameters in the model except $w$, and $\mathrm{tr}P(w)$ is the trace of the matrix $P(w)$.

The overall structure of the objective function consists of a loss term and a penalty term. The loss term uses mean squared error (MSE) to penalize prediction errors at the squared level, providing a sensitive measure of model prediction deviation. The penalty term includes penalties for the size of the candidate model and all parameters of the model. The penalty for the model's parameter sizes employs $L_2$ regularization to constrain overall parameter magnitudes.

In model averaging based on the Mallows criterion, the trace of the matrix is commonly used to characterize model complexity. It estimates the rank of the covariance matrix, which determines the number of free parameters in the model and reflects the information extractable from data without compromising goodness of fit. The objective function combines the trace of the matrix with the sum of squared residuals, weighing model complexity and goodness of fit. This approach selects a relatively simple model with good explanatory power. Introducing a penalty term for candidate model complexity in the objective function helps mitigate overfitting or underfitting, improving the model's predictive ability.

The objective function, as shown in (2), is minimized to find the optimal weight vector based on the Mallows criterion. The weight vector $w$ remains subject to the constraints described in (1). In PyTorch, the loss function is typically defined as part of the model for calculating and optimizing the loss during training. However, when modifying the built-in loss function to enhance model performance, it is necessary to customize the loss function to align it closely with the intended optimization target. This is achieved by creating the MallowLoss class, inheriting from the base nn.Module, and implementing the forward() method to calculate custom Mallows loss values. The method takes multiple input tensors (usually observed value $y$ and predicted value $\widehat{y}$), measures the error degree between them using established methods, and returns a scalar representing the loss value. The MallowLoss class defined in this paper accepts $P(w)$ as an additional parameter for loss value calculation. $P(w)$ is one of the return values from the model's forward propagation and is passed to the MallowLoss instance to compute the user-defined Mallows loss.

### 2.3 Algorithm of instance-wise graph-based Mallows model averaging framework

We employed a small-batch stochastic gradient descent algorithm to train the model while utilizing the Adam algorithm [26] to adjust the learning rate.

**Algorithm 1** IGMMA framework algorithm

**Require:** Training data $D\_train$, initial parameter $\Theta$, learning rate $\gamma$

**Ensure:** Time series prediction $p$

**while** The algorithm termination criteria have not been met **do**
The features of the training instance in $D\_train$ are encoded as training instance embeddings $E$.
    **for** $D_{batch}$ in $D_{train}$ **do**
Encode the characteristics of the small-batch $D_{batch}$ instance as embeddings $H_t$;
Sample the most relevant training instance $D_{train}^{sample}$ from $D_{train}$ for $D_{batch}$;
The information in $D_{train}^{sample}$ is aggregated into $D_{batch}$ to form $h^c = Concat(\widehat{h^t}, h^t)$;
$\widehat{h^t}$ and $h^t$in $h^c$ are randomly sampled with the same dimension to obtain multiple $h^c$;
Using $h^c$ to calculate its corresponding $P^{(s)}$ matrix for parameter estimation, several least squares estimates are obtained;
The calculation results are weighted by $w$;
Predict future time series $p^t$;
The random gradient of parameter $\Theta$ is calculated according to the formula method;
Update parameter $\Theta$ according to gradient and learning rate $\gamma$;
    **end for**
**end while**

To ensure that the weights $w$ of each model align with common sense and their inherent meaning, we incorporated the WeightConstraint class during training. Within this class, we screened the parameters involved in gradient updates based on their logical definitions. Specif-

ically, we iterated over the model's named parameters and applied constraints to the weight $w$ of the model. We constrained its components to fall within the range of 0 and 1, and then normalized the vector. This resulted in constraints where each component is greater than 0 and less than 1, and the sum of the components equals 1. In each iteration, we enforced this constraint on the model, effectively treating the problem as a constrained optimization problem. Algorithm 1 illustrates the training process.

## 3 Empirical analysis

In the previous section, we built upon the model averaging method by modifying the loss function to achieve better prediction performance. We defined an instance-level model averaging framework based on the Mallows criterion and graph. We plan to conduct empirical research on a multi-variety futures dataset using this framework. In addition, we defined the instance-wise graph-based model averaging (IGMA) framework, which solely incorporates the model averaging module without modifying the objective function.

Ablation experiments were conducted to demonstrate the effectiveness of the model averaging module. We mainly implemented IGMA, IGMMA, and other frameworks proposed in the literature based on PyTorch [30]1.11.0. We also reproduced other necessary baseline models based on this deep learning framework. All experiments were conducted using a single NVIDIA GeForce RTX 3090 graphics card. The software required for the experiments included the programming language Python 3.9.12, data processing libraries NumPy 1.24.2 and Pandas 1.4.2, machine learning framework scikit-learn 1.1.0, deep learning frameworks PyTorch 1.11.0 and TensorFlow 2.8.0, CUDA 11.6/cuDNN 8.4.0, and hyperparameter optimization framework Ray [31] 2.3.0.

### 3.1 Multivariable future time series data collection

Futures trading is a derivative trading method that mainly involves the trading of future commodities or financial assets. This trading method can be broadly classified into two categories: commodity futures and financial futures. Commodity futures are typically divided into three major categories: agricultural, metal, and energy. Financial futures mainly consist of foreign exchange futures, interest rate futures, and stock index futures.

To ensure the accuracy, consistency, and completeness of the data, we collected from multiple data sources and cross-validated (including unified format conversion) historical and real-time data from various data providers. The primary data flow is shown in Figure 2.

Cross-validation among multiple data sources can be achieved through mutual support, supplementation, and correction of each data source, thus providing more accurate, consistent, and comprehensive information. At the same time, because of the differences in data source links and processing methods, speed comparisons can be made on real-time data to improve timeliness. For historical and real-time futures data, we compared the above data providers in terms of data quality, required workload, cost, etc., and we selected Tonghuashun (iFind) and JQData as historical data sources based on their data interface and cost suitability. For real-time data, we accessed real-time data and minute data integration from Macro Source Futures

and Nanhua Futures through the Comprehensive Transaction Platform (CTP). Algorithm 2 outlines the concatenation method, where tick data refers to tick-by-tick data, and bar data refers to candlestick data generated from high-frequency tick data for clearer display and easier storage of information.

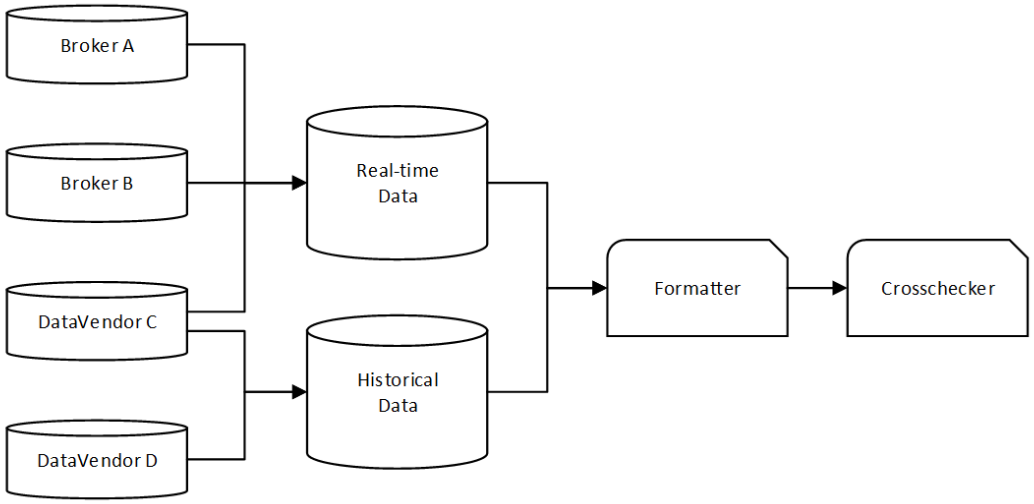


**Figure 2** The main flow direction of data

**Algorithm 2** Tick data concatenation bar data algorithm

**Require:** $tickdata, last\_volume, last\_amount$

**Ensure:** $open, close, high, low, volume, amount$

$open, close, high, low, volume, amount = 0$;
**while** $min \quad in \quad cur\_min$ **do**
  **if** open=0 **then**
    $open = tick$;
  **end if**
  **if** high=0 or $high < tick$ **then**
    $high = tick$;
  **end if**
  **if** low=0 or $low > tick$ **then**
    $low = tick$;
  **end if**
close=tick; volume+=last_volume; amount+=last_amount;
**end while**

In this study, when we select a variety of futures data, we need to consider its market liquidity and trading activity, and it should have the characteristics of a longer listing time, so we excluded some varieties such as bean II, rapeseed, Pumai, wire, and other varieties with

poor liquidity and inactive trading. Thus, forty-one futures varieties were finally determined: soybean, soybean meal, rapeseed meal, soybean oil, rapeseed oil, palm oil, sugar, egg, corn, corn starch, cotton, natural rubber, glass, methanol, polyethylene, polypropylene, polyvinyl chloride, PTA, asphalt, ferrosilicon, manganese silicon, fuel oil, rebar, iron ore, coke, coking coal, hot rolled coil, steel, iron ore. Crude oil contract, silver, gold, aluminum, copper, zinc, nickel, lead, tin, Shanghai 50, Shanghai 300, China 500, 5 Treasury bonds, 10 Treasury bonds, and other contracts. The dataset used in this paper contains the price data for the above futures contracts for the period from January 1, 2015, to January 28, 2022, and the data sampling frequency is 5min. The data were collected according to the method described in the previous section after the database was extracted as a comma-separated values file (comma-separated values, CSV) and then read.

We take the multivariate futures dataset in Section 3 as an example. Using $d = 40$, a variable dimension of $n = 41$, and a prediction step of $h = 1$, we obtain the price series or numerical matrix of the past 40 days for the forty-one futures at time $t$. This information allows us to predict the price cross-section information of the forty-one futures at a future time. Thus, the problem involves a continuous rolling window approach to forecasting. The basic statistical information for the dataset is shown in Table 1.

**Table 1** Statistical information of dataset

| Data set | Time | Variables | Sampling frequency |
|---|---|---|---|
| modified_futures_5min | 204792 | 41 | 5min |

We split this dataset into the training set (60%), verification set (20%), and test set (20%) in strict order of the timestamps. All models uniformly use the same lookback window and forecasting step size. The input feature lookback window size for each instance window is 40, i.e., the input value at each time is the time series of the past 40 timestamps for each variable, and the prediction step horizon is 1, i.e., the value at the next timestamp.

### 3.2 Comparison experiments of IGMTF and IGMA

To validate the effectiveness of the model averaging module, this experiment compares and analyzes two scenarios: the IGMA framework (IGMTF hyperparameter variant with the model averaging module) and the IGMTF (IGMTF hyperparameter variant without the model averaging module). The objective is to investigate the impact of the model averaging module on model performance. Specifically, the IGMTF hyperparameter variant framework serves as a reference, and it includes cases in which the hidden dimensions of the training instance and the small-batch instance GRU layer are set to 8, 32, 64, 128, and 256. Consequently, the total number of feature dimensions are 16, 64, 128, 256, and 512.

To ensure consistent empirical analysis, as described in Section 2, it is essential to maintain the same hyperparameter settings except for the aforementioned variations. Specifically, we use a batch size of 64, a learning rate of 0.0001, and employ an Adam optimizer for 50 training iterations. This ensures that only the specified variables change, which lends credibility to

the conclusions drawn. The model with the smallest relative square root error (RRSE) on the validation set is considered to be the optimal model, and its RRSE, relative absolute error (RAE), and correlation coefficient (CORR) metrics are evaluated on the test set to assess the model's performance beyond the training samples.

We apply the RRS, RAE, and CORR as evaluation indicators to evaluate the performance of each model on the validation and testing sets of the futures benchmark dataset. The "↑" at the end of the evaluation indicator indicates that the higher the value, the better the overall performance of the model. If the arrow is shown as "↓", the value decreases with better performance of the model. IGMTF_ 512D, IGMTF_ 256D, IGMTF_ 128D, IGMTF_ 64D, IGMTF_ 16D refers to the scenarios where the hidden layer dimensions of the IGMTF framework training instance and the GRU layer in small batch instances are 256, 128, 64, 32, and 8, respectively. Therefore, the dimensions of the feature vectors Concat($\widehat{\mathrm{h}^{\mathrm{t}}}, \mathrm{h}^{\mathrm{t}}$) are input into the linear layer before the prediction results correspond to 512, 256, 128, 64, and 16 are obtained. In the IGMA framework defined in this article, except for the model averaging module, the hyperparameters follow the IGMTF framework setting with a total feature vector dimension of 16. Table 2 shows the performance results under the above assessment indicators for each framework. A paired t-test was carried out between IGMA and a reference method at a significance level of 0.05, and * is appended when the enhancement of IGMA over the method achieves statistical significance. The numbers in bold indicate the better result in the comparison.

**Table 2** Forecasting results of multi-variate time-series models 1

| Model | RRSE(↓) | RAE(↓) | CORR(↑) |
|---|---|---|---|
| IGMTF_16D | $*6.853 \times 10^{-3}$ | $*5.432 \times 10^{-3}$ | 0.999808 |
| IGMTF_64D | $*6.321 \times 10^{-3}$ | $*4.662 \times 10^{-3}$ | 0.999811 |
| IGMTF_128D | $*6.187 \times 10^{-3}$ | $*5.495 \times 10^{-3}$ | 0.999842 |
| IGMTF_256D | $*6.042 \times 10^{-3}$ | $*4.170 \times 10^{-3}$ | 0.999848 |
| IGMTF_512D | $*6.029 \times 10^{-3}$ | $*3.509 \times 10^{-3}$ | 0.999854 |
| IGMA | $\mathbf{5.491 \times 10^{-3}}$ | $\mathbf{2.888 \times 10^{-3}}$ | **0.999881** |

To make the results more intuitive and easier to compare, the RRSE and CORR of the test set in the above table were extracted as the data of the stacked line bar chart, and the predicted results are displayed in Figure 3. Among the results, the RRSE and RAE indicators correspond to the left coordinate axis in the figure, and the CORR indicators correspond to the right coordinate axis.

From the chart, we can infer that the prediction performance (measured by the RRSE, RAE, and CORR indicators) improves as the hidden feature dimension increases within the IGMTF framework. It can be concluded that the total feature dimension of the IGMTF framework is a hyperparameter that exhibits sensitivity within the range of 16 to 512 dimensions. Moreover, higher hidden feature dimensions allow for the capture of more information, thereby enhancing the predictive capability.

By augmenting the IGMTF_16D framework with the model averaging module, the IGMA framework achieves the best performance of all frameworks, as evidenced by its superior performance on the three test set indices. An ablation analysis of the individual module reveals that the model averaging module significantly enhances the performance of the IGMA framework on the test set, reducing the RRSE to 0.005491, the RAE to 0.002888, and yielding a CORR value of 0.999881. Compared with the prediction performance of IGMTF_16D, these results show a substantial 19.87% improvement in RRSE and a 46.83% decrease in RAE. Because the CORR has a high baseline value, its increase ratio is not discussed. Compared with the IGMTF_512D framework, which closely follows the IGMA framework, the same indices exhibit a respective reduction in error of 8.92% and 17.70%. In other words, the predictive performance of IGMTF_512D, which is obtained by implementing the model averaging module in IGMTF_16D, surpasses that of IGMTF_512D, which differs from the original structure and merely generates feature dimensions.

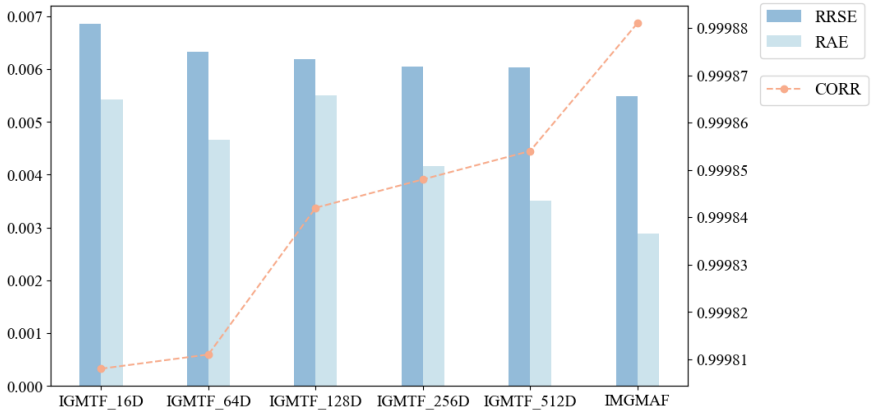


**Figure 3** Visualization of prediction results of multivariate time series model

On this basis, we compared the time required for each iteration round of the IGMTF of different dimensions, as shown in Table 3. The comparison of the training time for each hyperparameter model is analyzed. When the hidden layer dimension is small, the training time increases with an increase in dimensions, but the change range is small. When the hidden layer dimension reaches 256 dimensions, i.e., the total feature dimension is 512, the duration of each cycle of iteration by the model increases significantly from 70s to 85s to 120s, and the time is not proportional to the dimension scale but is positively correlated.

**Table 3** Training time of multivariate time-series models 1

| | IGMTF_16D | IGMTF_64D | IGMTF_128D | IGMTF_256D | IGMTF_512D | IGMA |
|---|---|---|---|---|---|---|
| Time | ∼70s | ∼72s | ∼80s | ∼85s | ∼120s | ∼85s |

Therefore, its predictive performance and training time are basically positively correlated with the hidden layer dimension, i.e., the parameter size of the model. As a result, the IGMA

framework performs better on both the verification and the test sets than other IGMTF models with settings for the hidden layer dimension. That is, when the model averaging module is added, a relatively small number of parameters can be used to achieve greater prediction accuracy. Thus, the model averaging module added to the multivariate time series dataset significantly improves the performance of multivariate time series forecasting.

### 3.3 Comparison experiments with IGMMA and discussion

We will compare the proposed IGMMA framework with the multivariate time series prediction method as follows:

1) LSTNet: A deep neural network architecture framework for modeling long- and short-term time series patterns. It was proposed by Lai et al. [27]. It uses CNN to extract short-term local dependency patterns, whereas RNN is used to capture complex long-term dependency relationships.

2) TCN: A convolutional neural network architecture framework proposed by Bai et al. [28]. It uses a series of expanded convolutional layers and residual blocks to construct the network structure, which can expand the receptive field of the convolutional kernel without increasing the number of parameters. This enables the TCN to better capture long-term dependencies in sequences.

3) LightCTS: A prediction framework for correlated time series (CTS) proposed by Lai et al. [29]. It aims to explore a different direction of implementing more efficient and lightweight models that can maintain accuracy while being deployed on resource-constrained devices. The framework adopts pure time and space operation stacking instead of alternating stacking with higher computational costs.

4) IGMA framework: A framework that only adds the model averaging module without modifying the loss function to explore the impact of modifying the calculation method of the target function to be optimized on the model's prediction performance.

For the TCN, LSTNet, and LightCTS frameworks, we chose to retain most parameter settings in the original open-source code. For the lookback and prediction step sizes, all models were uniformly set to 40 and 1, while the other parameters remained default settings (if there are obvious hyperparameters that need to be modified to conform to the dataset structure, they should be modified accordingly).

Using the same preprocessing method for multiple future datasets, we replicated the baseline model based on the corresponding open-source code available on GitHub. To evaluate the performance on the validation set, we employed Ray to search the framework's hyperparameters to identify the optimal combination. The parameters involved in the grid search included the learning rate (0.0001, 0.0005, 0.001), the number of samples per batch for small-batch instances (non-training instances) (16, 32, 64), and the weight decay coefficient, which represents the coefficient of the $L_2$ regularization term in $\mathcal{L}'$ (0.0001, 0.001). This yielded 18 hyperparameter combinations. Because LSTNet and TCN models exhibited exceptionally short training times

per iteration, we set their iteration rounds to 100. For the other models (LightCTS, IGMA, IGMMA), the number of iteration rounds was set to 50, and the final result was obtained by averaging the outcomes of five experiments.

We also applied the same indicators, namely RRSE, RAE, and CORR to evaluate the performance of each model on the validation and test set of the futures dataset. Similarly, the value "↑" beside the evaluation indicator means that the higher the value, the better the overall performance of the model; if the arrow is shown as" ↓", the value decreases as the model performance increases. Table 4 shows the performance results under the above assessment indicators in each framework.

**Table 4** Forecasting results of multi-variate time-series models 2

| Model | RRSE(↓) | RAE(↓) | CORR(↑) |
|---|---|---|---|
| LSTNet | $*9.800 \times 10^{-3}$ | $*6.220 \times 10^{-3}$ | 0.999604 |
| TCN | $*6.126 \times 10^{-3}$ | $*3.656 \times 10^{-3}$ | 0.999843 |
| LightCTS | $*6.560 \times 10^{-3}$ | $*4.327 \times 10^{-3}$ | 0.999809 |
| IGMA | $5.491 \times 10^{-3}$ | $2.888 \times 10^{-3}$ | 0.999881 |
| IGMMA | $\mathbf{4.951 \times 10^{-3}}$ | $\mathbf{2.627 \times 10^{-3}}$ | **0.999885** |

The RRSE and CORR of the test set in the above table were visualized using a stacked line bar chart, and the results are shown in Figure 4.

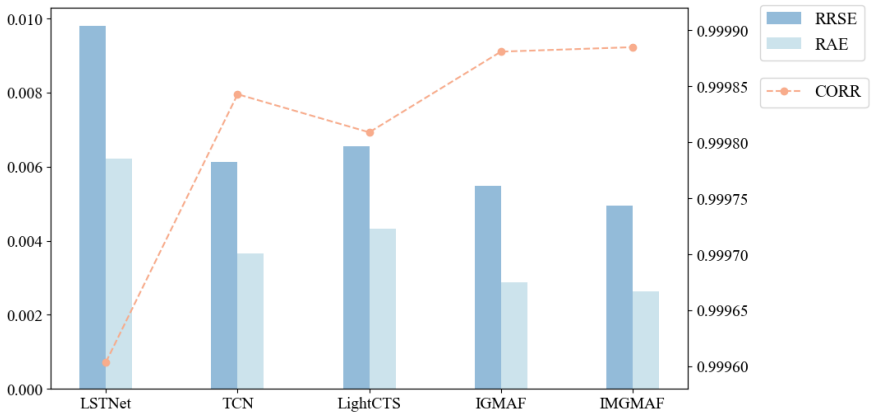


**Figure 4** Visualization of prediction results of multivariate time series model 1

Upon observing the prediction results chart, it becomes clear that LSTNet's performance on this particular future dataset deviates significantly from that of other time series prediction frameworks. This divergence can be attributed to LSTNet's model structure, which consists of relatively simplistic CNNs and RNNs. Such a structure may impose limitations on its performance when it encounters time series data that contain long-term dependencies. Insufficient feature extraction by CNNs may also impede the comprehensive representation of the inherent

complexity and patterns during subsequent modeling. In comparison to LSTNet, TCN demonstrates slightly superior prediction capabilities. However, similar to LSTNet, it may face minor challenges stemming from CNN feature extraction problems. Of the five frameworks, LightCTS stands out as a novel model with outstanding empirical performance and moderate complexity. Nevertheless, it surpasses only LSTNet in terms of effectiveness. One possible reason for this disparity may be the modeling of CTS. In general, although objective correlations exist between specific objects in the multi-variety futures price series, these correlations tend to attenuate over a long-term series. Conversely, the IGMA framework enhances its alignment with the dynamics of multi-variety futures prices by extracting the most relevant information from historical series through sampling and aggregation techniques.

The IGMMA framework presents innovative enhancements to the widely employed linear layer in deep learning methods. It replaces this layer with the model averaging approach and includes a more suitable penalty term for selecting model weight parameters. By modifying the original MAE and $L_2$ regularization methods using the Mallows criterion as a basis, the model achieves substantial improvements in prediction accuracy and generalization performance.

Based on the results in Table 4, we can conclude that the IGMMA framework outperforms all other frameworks on the three indices of the test set by incorporating the model averaging module and modifying the loss function based on IGMTF_16D. The IGMMA framework achieves a low RRSE on the test set of 0.004951, an RAE of 0.002627, and a CORR of 0.999885. After the calculation method for the loss function is optimized, the prediction performance of the IGMMA framework is only marginally inferior to that of the IGMA framework proposed in the previous section, and this results in a reduction of 9.83% and 9.04% in the respective index errors. In other words, optimizing the objective function can enhance the prediction effectiveness of the framework based on model averaging. Considering that the IGMMA, which incorporates the model averaging module, was compared with the IGMTF multi-hyperparameter variant model without the module in the previous section, it can be concluded that the IGMMA framework outperforms all the other models used for comparison in terms of prediction accuracy and generalization performance.

It is essential to consider the complexity of the model, training time, and other factors, given their importance in practical applications. Therefore, we conducted an empirical study to analyze the time required for each iteration of different multivariate time series forecasting methods, as presented in Table 5. Based on this analysis, we derive relevant conclusions regarding the complexity of the models.

**Table 5** Training time of multivariate time series models 2

| | LSTNet | TCN | LightCTS | IGMA | IGMMA |
|---|---|---|---|---|---|
| Time | ∼10s | ∼20s | ∼130s | ∼85s | ∼120s |

According to Table 5, there is generally no clear and absolute positive or negative correlation between the performance and training time of the various models. However, we can make the following observations for each type of model (except IGMA):

1) The LSTNet and TCN models exhibited the shortest time per epoch, approximately 10 seconds and 20 seconds, respectively, which indicates relatively high training speeds. However, as mentioned earlier, their performance was relatively poor, which could be attributed to insufficient model complexity and inadequate feature extraction.

2) The LightCTS model required the longest time per epoch, around 130 seconds, which suggests relatively slow training. However, based on the experimental results, the model performed relatively well on the test set, which indicates certain advantages for time series data processing.

3) The IGMMA model took approximately 120 seconds for each epoch. Although its prediction accuracy was superior to that of the IGMA, which was the best-performing framework, the modification of the loss function increased the time required for each round of iteration. Unlike IGMTF_512D and LightCTS described in the previous section, in which each round of iteration lasted around 120 seconds, the overall training time for deep learning tasks fell within the medium range. In addition, considering that the number of parameters was not high, it becomes necessary to strike a balance between accuracy and training duration based on the practical application scenario. Alternatively, modifying the hyperparameters could make the model more suitable for the real-world task at hand.

Based on a comprehensive comparison across multiple dimensions, the iterative generalized model averaging (the IGMMA) framework proposed in this study significantly outperformed other baseline models in terms of prediction accuracy, deviation, and correlation performance. This superiority was observed consistently on both the validation and test sets. Thus, we can confidently conclude that for the task of multivariate time series prediction, the incorporation of the model averaging module and the iterative modification of the objective function to align it with practical problems and mathematical logic greatly enhanced the predictive capabilities of the framework for multivariate time series problems. In summary, the IGMMA framework, which is based on the model averaging method and Mallows' criterion, proved to be a highly effective approach for the prediction of multi-variety futures in time series analysis.

### 3.4 Hyperparameter analysis

For deep learning tasks, the choice of hyperparameters is crucial as it significantly affects the performance of the model. In this section, we conduct a systematic experimental test to analyze the sensitivity and specific mode of action of the model's hyperparameters.

Some hyperparameters in the model, including the hidden layer dimension of the GRU+MLP layer feature extraction module, the number of time stamps sampled from the training instance, and the number of selected nearest neighbors in the graph aggregation module, etc., play vital roles in the experimental setup. It is crucial to ensure a fair comparison between the proposed IGMA framework and the baseline model by maintaining fixed variables. Therefore, in this section, we focus on three specific hyperparameters for empirical analysis: the learning rate of the model, the batch size of the samples per batch, and the weight decay coefficient ($L_2$

regularization term coefficient). The learning rate is set to {0.0001, 0.0005, 0.001}, the number of samples per batch is set to {16, 32, 64}, and the weight decay coefficient is set to {0.0001, 0.001}. Figures 5 to 7 illustrate the relative square root error (RRSE), relative absolute error (RAE), and correlation coefficient (CORR) under different hyperparameter settings.

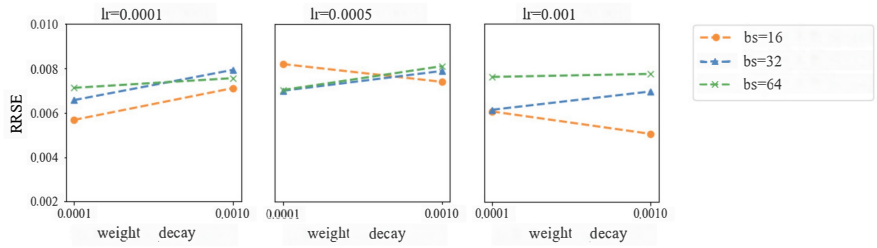


**Figure 5** Relative square root error RRSE under hyperparameter adjustment

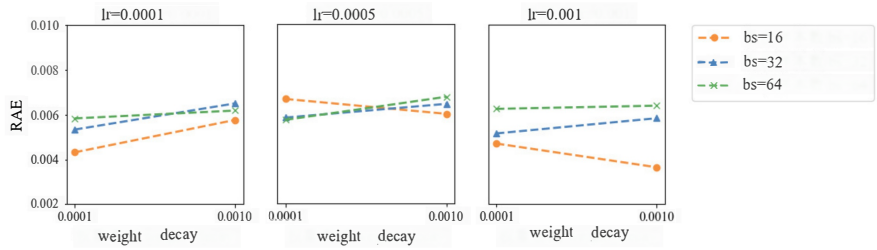


**Figure 6** Relative absolute error RAE under hyperparameter adjustment

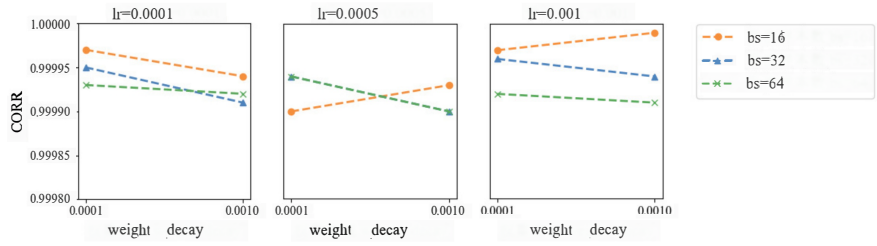


**Figure 7** The correlation coefficient CORR under hyperparameter regulation

Figures 5 – 7 illustrate the RRSE, RAE, and CORR under different hyperparameter settings. The RRSE and RAE trends in Figures 5–7 exhibit similarity but differ from the CORR trends, aligning with the definition of the evaluation indicators. From the visualization of the error measurement indexes in Figures 5 and 6, it can be observed that the RRSE and RAE error values of the model generally exhibit an increasing trend as the number of samples in each batch gradually increases from 16 to 64, regardless of the values of the learning rate or weight decay coefficient. Generally, a smaller number of batch samples leads to a smaller error value of the model. This is attributed to the improved stability of the gradient calculation, smoother error value curve during model training, and reduced training time achieved with a larger number

of batch samples. However, a smaller batch size may compromise the model's generalization ability and introduce noise that can steer the solution away from the global optimum.

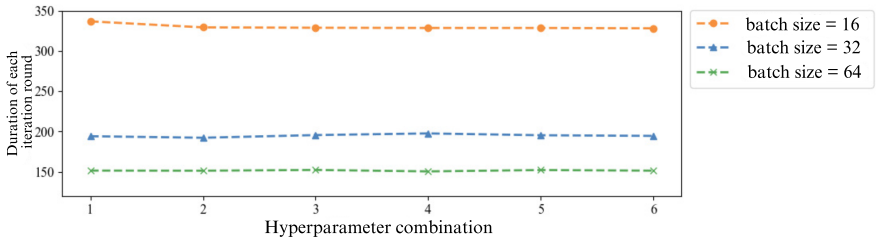


**Figure 8** Duration of each iteration round for each hyperparameter combination

Figure 8 provides a summary of the time required for each iteration round and the classification of 18 hyperparameter combinations into 6 categories, considering batch sample sizes of 16, 32, and 64. The time required for each iteration round varies due to the different settings of the learning rate and weight attenuation coefficient. The top line in the figure represents the time required when the batch sample size is 16, which is approximately 350 seconds, reaching its highest point when both the learning rate and weight attenuation coefficient are set to 0.0001. The bottom line represents the time required when the batch sample size is 64, which is approximately 150 seconds. The shortest time is when the learning rate and weight attenuation coefficient are set to 0.001. It is worth noting that additional variable calculations are required for hyperparameter adjustment using Ray, which slightly increases the time required for a model with the same parameter settings in the hyperparameter experiment compared with the data obtained in the main results section of the experiment. Although the specific position order of the corresponding points for each hyperparameter combination is determined by the specific parameter setting, the general trend is that the batch sample size increases from small to large, and the required time increases from short to long. Therefore, when the batch sample size is increased from 16 to 64, the time required for each iteration round significantly increases. Considering both the model's forecasting ability and training time, the change in batch sample size results in significant fluctuations in the corresponding index. Thus, the batch sample size is considered to be a sensitive hyperparameter.

Under the setting of the same number of samples per batch and the same weight attenuation coefficient, the RRSE tends to be high when the number of samples per batch is 16 and the weight attenuation coefficient is set to 0.0001. Conversely, in other cases where the learning rate is increased from 0.0001 to 0.0005, the RRSE generally decreases because of the doubling of the learning rate. When the batch size is relatively large, specifically with 64 samples per batch, increasing the learning rate does not significantly affect the RRSE value. However, as the batch size decreases, particularly when the number of samples per batch is relatively small, the RRSE shows a significant decrease. Using a larger batch size enhances the stability and robustness of the model during the training process, thereby reducing its sensitivity to changes in the hyperparameters. When the learning rate and the number of samples per batch are fixed, the

effect of the weight attenuation coefficient on the error index depends on the specific training situation and experimental results. In general, a higher weight attenuation coefficient imposes a stronger penalty on network structures with high complexity. However, an excessively high coefficient can lead to the underfitting of the model. According to the experimental results, when the number of samples per batch is 16 and the learning rate is relatively high, specifically 0.0005 and 0.001, increasing the weight attenuation coefficient from 0.0001 to 0.001 significantly reduces the error value. However, when the learning rate is low and the number of samples per batch is relatively large, increasing the weight attenuation coefficient can increase the error index. This phenomenon may be attributed to the fact that, in the latter case, the higher weight attenuation coefficient overly penalizes the model parameters, thereby constraining the model's learning capacity. The analysis method for the CORR is similar to that of the error index, but it exhibits the opposite trend. This will not be further discussed in this context.

In summary, we determined the optimal hyperparameter combination for the sample number per batch, learning rate, and weight decay as {16, 0.001, 0.001}. However, in practical applications, particularly when forecasting with real-time futures data, it becomes necessary to consider the training and forecasting time of the model. Therefore, trade-offs and compromises should be made between prediction accuracy and factors such as time, resources, and manpower.

## 4 Conclusions

We introduced an innovative approach that combines model averaging with deep learning methods, and we distinguish it from traditional model averaging techniques. The model averaging module is an integral component of the IGMA network and is not treated as a separate entity. Considering the characteristics and preferences of the model averaging method, a new objective function was defined for optimization, and empirical research was conducted on the proposed framework algorithms.

The experimental results demonstrated that the model averaging module achieves higher prediction accuracy with a relatively small parameter scale and betters the performance of the original model on the multi-variety futures dataset. The IGMMA framework, which incorporates the model averaging modules and modifies the objective function, outperforms all other baseline models on the test set, despite having a slightly higher model complexity. In conclusion, the IGMMA framework effectively addresses the discussed multi-variety futures time series prediction problem, showcasing the potential application of model averaging in deep learning.

In future studies, we aim to widen the scope of multivariate time series prediction beyond the limited use of price index series for various futures varieties under a specific time stamp. We plan to enhance the analysis to encompass multi-feature and multi-variable scenarios by incorporating multidimensional information, including but not limited to trading volume. Furthermore, we intend to explore the integration of alternative data sources, such as public opinion data, to address multivariate time series forecasting challenges without significantly increasing model complexity or deviating from real-world conditions.

## References


[1] Li Y, Yu R, Shahabi C, et al. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *International Conference on Learning Representations*, Vancouver, 2018.

[2] Wu Y, Liu Y, Ahmed S H, et al. Dominant data set selection algorithms for electricity consumption time-series data analysis based on affine transformation. *IEEE Internet Things J.*, 2020, **7**(5): 4347–4360.

[3] Cheng D, Yang F, Xiang S, et al. Financial time series forecasting with multi-modality graph neural network. *Pattern Recogn.*, 2022, **121**: 108218.

[4] Yu L and Gao X. Improve robustness and accuracy of deep neural network with $L_{2,\infty}$ normalization. *J. Syst. Sci. Com plex.*, 2023, **36**(1): 3-28.

[5] Wang C and Gao Q. High and low prices prediction of soybean futures with LSTM neural network. *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, 2018.

[6] Sun T, Wang J, Ni J, et al. Predicting futures market movement using deep neural networks. *2019 18th IEEE International Conference On Machine Learning and Applications (ICMLA)*, Boca Raton, 2019.

[7] Ji L, Zou Y, He K, et al. Carbon futures price forecasting based with ARIMA-CNN-LSTM model. *Procedia Computer Science*, 2019, **162**: 33–38.

[8] Yang Y. Adaptive regression by mixing. *J. Amer. Statist Assoc.*, 2001, **96**(454): 574-588.

[9] Yuan Z and Yang Y. Combining linear regression models: When and how?. *J. Amer. Statist Assoc.*, 2005, **100**(472): 1202–1214.

[10] Bates J M and Granger C W. The combination of forecasts. *J. Oper. Res. Soc.*, 1969, **20**(4): 451–468.

[11] Longford N T. Model selection and efficiency–is 'Which model...?'the right question?. *J. Roy. Statist. Soc. Ser. A*, 2005, **168**(3): 469–472.

[12] Zhang X and Zou G. Model averaging method and its application in forecast. *Statistical Research*, 2011, **28**(6): 97–102.

[13] Leung G and Barron A R. Information theory and mixing least-squares regressions. *IEEE Trans. Inform. Theory*, 2006, **52**(8): 3396–3410.

[14] Hansen B E. Least squares model averaging. *Econometrica*, 2007, **75**(4): 1175–1189.

[15] Wan A T K, Zhang X, and Zou G. Least squares model averaging by Mallows criterion. *J. Econometrics*, 2010, **156**(2): 277–283.

[16] Liu Q and Okui R. Heteroscedasticity-robust $C_p$ model averaging. *Econom. J.*, 2013, **16**(3): 463–472.

[17] Gao Y, Zhang X, Wang S, et al. Frequentist model averaging for threshold models. *Ann. Inst. Statist. Math.*, 2019, **71**: 275–306.

[18] Zhu R, Wan A T K, Zhang X, et al. A Mallows-type model averaging estimator for the varying-coefficient partially linear model. *J. Amer. Statist Assoc.*, 2019, **114**(526): 882–892.

[19] Wang J, He J, Liang H, et al. Optimal model average prediction in orthogonal kriging models. *J. Syst. Sci. Com plex.*, 2023.

[20] Gu S, Kelly B, and Xiu D. Empirical asset pricing via machine learning. *Rev. Financ. Stud.*,

2020, **33**(5): 2223–2273.

[21] Goulet Coulombe P, Leroux M, Stevanovic D, et al. How is machine learning useful for macroeconomic forecasting?. *J. Appl. Econometrics*, 2022, **37**(5): 920–964.

[22] Mullainathan S and Spiess J. Machine learning: An applied econometric approach. *J Econ. Perspect*, 2017, **31**(2): 87–106.

[23] Xie T, Yu J, and Zeng T. Econometric methods and data science techniques: A review of two strands of literature and an introduction to hybrid methods. *Working Paper*, 2020.

[24] Qiu Y, Xie T, Yu J, et al. Mallows-type averaging machine learning techniques. *Working Paper*, 2020.

[25] Xu W, Liu W, Bian J, et al. Instance-wise graph-based framework for multivariate time series forecasting. *Working Paper*, 2021.

[26] Konur O, Kingma D, and Ba J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, San Diego, 2015.

[27] Lai G, Chang W C, Yang Y, et al. Modeling long-and short-term temporal patterns with deep neural networks. *The 41st international ACM SIGIR conference on Research & development in Information Retrieval*, Ann Arbor MI, 2018.

[28] Bai S, Kolter J Z, and Koltun V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *Working Paper*, 2018.

[29] Lai Z, Zhang D, Li H, et al. LightCTS: A lightweight framework for correlated time series forecasting. *Proceedings of the ACM on Management of Data*, 2023, **1**, 125.

[30] Paszke A, Gross S, Massa F, et al. PyTorch: An imperative style, high-performance deep learning library. *33rd Conference on Neural Information Processing System*, Vancouver, 2019.

[31] Liaw R, Liang E, Nishihara R, et al. Tune: A research platform for distributed model selection and training. *Working Paper*, 2018.