Hierarchical Joint Graph Learning and Multivariate Time Series Forecasting

Juhyeon Kim, Hyungeun Lee, Seungwon Yu, Ung Hwang, Wooyul Jung, Miseon Park
Department of Electronic Engineering
Hanyang University, Seoul, Korea
{wngus1310,didls1228,ahddl1324,gogowinner,jung6231}@hanyang.ac.kr
{misun9631}@naver.com

Kijung Yoon
Department of Electronic Engineering
Department of Artificial Intelligence
Hanyang University, Seoul, Korea
kiyoon@hanyang.ac.kr

Abstract

Multivariate time series is prevalent in many scientific and industrial domains. Modeling multivariate signals is challenging due to their long-range temporal dependencies and intricate interactions–both direct and indirect. To confront these complexities, we introduce a method of representing multivariate signals as nodes in a graph with edges indicating interdependency between them. Specifically, we leverage graph neural networks (GNN) and attention mechanisms to efficiently learn the underlying relationships within the time series data. Moreover, we suggest employing hierarchical signal decompositions running over the graphs to capture multiple spatial dependencies. The effectiveness of our proposed model is evaluated across various real-world benchmark datasets designed for long-term forecasting tasks. The results consistently showcase the superiority of our model, achieving an average 23% reduction in mean squared error (MSE) compared to existing models.

1 Introduction

Multivariate time series forecasting is a primary machine learning task in both scientific research and industrial applications [1, 2]. The interactions and dependencies between many time series data govern how they evolve, and these can range from simple linear correlations to complex relationships such as the traffic flows underlying intelligent transportation systems [3–6] or physical forces affecting the trajectories of objects in space [7–9].

Accurately predicting future values of the time series may require understanding their true relationships, which can provide valuable insights into the system represented by the time series. Recent studies aim to jointly infer these relationships and learn to forecast in an end-to-end manner, even without prior knowledge of the underlying graph [5, 10]. However, inferring the graph from numerous time series data has a quadratic computational complexity, making it prohibitively expensive to scale to a large number of time signals.

Another important aspect of time series forecasting is the presence of non-stationary properties, such as seasonal effects, trends, and other structures that depend on the time index [11]. Such properties may need to be eliminated before modeling, and a recent line of work aims to incorporate trend and seasonality decomposition into the model architecture to simplify the prediction process [12, 13].

Temporal Graph Learning Workshop @ NeurIPS 2023, New Orleans.

Therefore, it is natural to ask whether one can leverage deep neural networks to combine the strength of both worlds: 1) using a latent graph structure that aids in time series forecasting with each signal represented as a node and the interactions between them as edges, and 2) using end-to-end training to model the time series by decomposing it into multiple levels, which enables separate modeling of different patterns at each level, and then combining them to make accurate predictions. Existing works have not addressed both of these strengths together in a unified framework, and this is precisely the research question we seek to address in our current study.

To address this, we propose the use of graph neural networks (GNN) and a self-attention mechanism that efficiently infers latent graph structures with a time complexity and memory usage of $\mathcal{O}(N \log N)$ where N is the number of time series. We further incorporate hierarchical residual blocks to learn backcast and forecast outputs. These blocks operate across multiple inferred graphs, and the aggregated forecasts contribute to producing the final prediction. By implementing this approach, we have achieved a superior forecasting performance compared to baseline models, with an average enhancement of 23%. For an overview, this paper brings the following contributions:

1. We introduce a novel approach that extends hierarchical signal decomposition, merging it with concurrent hierarchical latent graphs learning. This is termed as hierarchical joint graph learning and multivariate time series forecasting (HGMTS).
2. Our method incorporates a sparse self-attention mechanism, which we establish as a good inductive bias when learning on latent graphs and addressing long sequence time series forecasting (LSTF) challenges.
3. Through our experimental findings, it is evident that our proposed model outperforms traditional transformer networks in multivariate time series forecasting. The design not only sets a superior standard for direct multi-step forecasting but also establishes itself as a promising spatio-temporal GNN benchmark for subsequent studies bridging latent graph learning and time series forecasting.

2 Related Work

Until recently, deep learning methods for time series forecasting have primarily focused on utilizing recurrent neural networks (RNN) and their variants to develop a sequence-to-sequence prediction approach [14–17], which has shown remarkable outcomes. Despite significant progress, however, these methods are yet to achieve accurate predictions for long sequence time series forecasting due to challenges such as the accumulation of errors in many steps of unrolling, as well as vanishing gradients and memory limitations [18].

Self-attention based transformer models proposed recently for LSTF tasks have revolutionized time series prediction and attained remarkable success. In contrast to traditional RNN models, transformers have exhibited superior capability in capturing long-range temporal dependencies. Still, recent advancements in this domain, as illustrated by LongFormer [19], Reformer [20], Informer [21], AutoFormer [22], and ETSformer [23], have predominantly zeroed in on improving the efficiency of the self-attention mechanism, particularly for handling long input and output sequences. Concurrently, there has been a rise in the development of attention-free architectures, as seen in Oreshkin et al. [12] and Challu et al. [13], which present a computationally efficient alternative for modeling extensive input-output relationships by using deep stacks of fully connected layers. However, such models often overlook the intricate interactions between signals in multivariate time series data, tending to process each time series independently.

Spatio-temporal graph neural networks (ST-GNNs) are a specific type of GNNs that are tailored to handle both time series data and their interactions. They have been used in a wide range of applications such as action recognition [24, 25] and traffic forecasting [26–28]. These networks integrate sequential models for capturing temporal dependencies with GNNs employed to encapsulate spatial correlations among distinct nodes. However, a caveat with ST-GNNs is that they necessitate prior information regarding structural connectivity to depict the interrelations in time series data. This can be a limitation in cases where the structural information is not available.

Accordingly, GNNs that include structure learning components have been developed to learn effective graph structures suitable for time series forecasting. Two such models, NRI [8] and GTS [6], calculate the probability of an edge between nodes using pairwise scores, resulting in a discrete adjacency

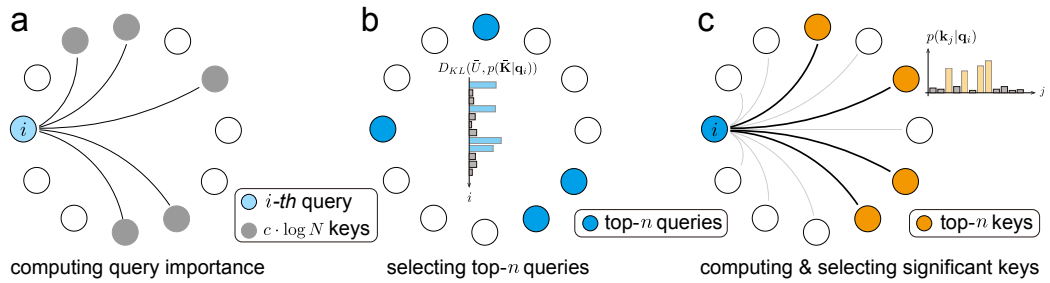

Figure 1: **Overview of the latent graph structure learning (L-GSL). (a)** Key nodes chosen at random (depicted as gray circles) are used to measure the significance of a query node (shown as a blue circle). **(b)** Top-n query nodes (blue circles) are picked according to the importance distribution across all query nodes. **(c)** Key nodes, colored in orange, that hold sufficient relevance to be linked with the chosen query node.

matrix. Nonetheless, this approach can be computationally intensive with a growing number of nodes. In contrast, MTGNN [5] and GDN [10] utilize a randomly initialized node embedding matrix to infer the latent graph structure. While this approach is less taxing on computational resources, it might compromise the accuracy of predictions.

3 Methods

In this section, we detail our proposed method, HGMTS. The overarching framework and core operational principles of this approach can be viewed in Figures 1 and 2.

3.1 Preliminaries

Let $\mathbf{X} \in \mathbb{R}^{N \times T \times M}$ represents a multivariate time series, where N signifies the count of signals originating from various sensors, T denotes the length of the sequence, and M represents the dimension of the signal input (usually M = 1). We depict this multivariate time series as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{A}\}$, wherein the collection of nodes denoted by $\mathcal{V}$ corresponds to the sensors, the set $\mathcal{E}$ pertains to the edges, and $\mathcal{A}$ represents the adjacency matrix. Notably, the precise composition of $\mathcal{E}$ and $\mathcal{A}$ is not known initially; however, our model will acquire this knowledge through the learning process.

3.2 Latent Graph Structure Learning (L-GSL)

We embrace the concept of self-attention (introduced by [29]) and employ the attention scores in the role of edge weights. The process of learning the adjacency matrix of the graph, denoted as $\mathcal{A} \in \mathbb{R}^{N \times N}$ unfolds as follows:

$$\mathbf{Q} = \mathbf{H}\mathbf{W}^{Q},\ \mathbf{K} = \mathbf{H}\mathbf{W}^{K},\ \mathcal{A} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{T}}{\sqrt{D}}\right) \tag{1}$$

where $\mathbf{H} \in \mathbb{R}^{N \times D}$ corresponds to node embeddings[1], $\mathbf{W}^{Q} \in \mathbb{R}^{D \times D}$ and $\mathbf{W}^{K} \in \mathbb{R}^{D \times D}$ are weight matrices that project $\mathbf{H}$ into query $\mathbf{Q}$ and key $\mathbf{K}$, respectively. The main limitation in estimating latent graph structures in Eq.(1) for a large value of N is the necessity to perform quadratic time dot-product computations along with the utilization of $\mathcal{O}(N^2)$ memory. In an effort to achieve a self-attention mechanism complexity of $\mathcal{O}(N \log N)$, our approach involves identifying pivotal query nodes and their associated significant key nodes in a sequential manner.

3.2.1 Identifying pivotal query nodes

For the purpose of determining which query nodes will establish connections with other nodes, our initial step involves evaluating the significance of queries. Recent studies [30, 31, 19, 21] have

[1]The methodology for computing node embeddings from multivariate time series is detailed in Section 3.3

highlighted the existence of sparsity in the distribution of self-attention probabilities. Drawing inspiration from these findings, we establish the importance of queries based on the Kullback-Leibler (KL) divergence between a uniform distribution and the attention probability distribution of query nodes.

Let $\mathbf{q}_i$ and $\mathbf{k}_i$ represent the i-th row in matrices $\mathbf{Q}$ and $\mathbf{K}$ respectively. For a given query node, $p(\mathbf{k}_j|\mathbf{q}_i) = \exp(\mathbf{q}_i\mathbf{k}_j^\top)/\sum_\ell \exp(\mathbf{q}_i\mathbf{k}_\ell^\top)$ denotes the attention probability of the i-th query towards the j-th key node. Then, $p(\mathbf{K}|\mathbf{q}_i) = [p(\mathbf{k}_1|\mathbf{q}_i)\dots p(\mathbf{k}_N|\mathbf{q}_i)]$ indicates the probability distribution of how the i-th query allocates its attention/weight across all nodes. In this context, $D_{KL}(U, p(\mathbf{K}|\mathbf{q}_i))$ quantifies the deviation of a query node's attention probabilities from a uniform distribution $\mathcal{U}\{1, N\}$. This divergence measurement serves as a metric for identifying significant query nodes; a higher KL divergence suggests that a query's attention is mainly directed towards particular key nodes, rather than being evenly distributed. As a result, these query nodes are postulated to be suitable candidates for establishing sparse connections.

The traversal of all query nodes for this measurement, however, still entails a quadratic computational requirement. It is worth noting that a recent study demonstrated that the relative magnitudes of query importance remain unchanged even when the divergence metric is calculated using randomly sampled keys [21]. Building on this idea, we determine the importance of query nodes through the computation of $D_{KL}(\bar{U}, p(\bar{\mathbf{K}}|\mathbf{q}_i))$ instead, where $\bar{U} = \mathcal{U}\{1, n\}$, $\bar{\mathbf{K}}$ represents a matrix containing randomly sampled n row vectors from $\mathbf{K}$, and $n = \lfloor c \cdot \log N \rfloor$ denotes the number of random samples based on a constant sampling factor c (Figure 1a). Given this measurement of query importance, we select top-n query nodes and denote it as $\bar{\mathbf{Q}}$ (Figure 1b).

3.2.2 Identifying associated key nodes

Using the selected set of n query nodes, our subsequent step involves identifying the corresponding key nodes to establish connections. In pursuit of achieving this objective, we initiate by computing the attention probabilities $p(\mathbf{K}|\mathbf{q}_i)$ of the i-th query across all keys nodes; this procedure is reiterated for each of the n query nodes. Next, we choose the top-n key nodes for each query based on their attention scores (Figure 1c), and we designate this collection as $\bar{\mathbf{K}}$. The ultimate adjacency matrix, adhering to the sparsity constraint, is defined by the equation:

$$\bar{\mathcal{A}} = \mathrm{softmax}\left(\frac{\bar{\mathbf{Q}}\bar{\mathbf{K}}^T}{\sqrt{D}}\right) \tag{2}$$

In this equation, $\bar{\mathbf{Q}}$ and $\bar{\mathbf{K}}$ possess the same dimensions as $\mathbf{Q}$ and $\mathbf{K}$, except that the row vectors corresponding to insignificant query and key nodes are replaced with zeros. To sum up, the complexity of all the necessary computations for evaluating the significance of a query node and determining which key nodes to establish connections with, considering the top-n chosen queries, amounts to $\mathcal{O}(N \log N)$.

3.3 Hierarchical Signal Decomposition

This section provides an overview of the proposed approach shown in Figure 2 and discusses the overall design principles. Our approach builds upon N-BEATS [12], enhancing its key elements significantly. Our main methodology comprises of three primary elements: signal decomposition, latent graph structure learning, and constructing forecasts and backcasts in a hierarchical manner. Much like the N-BEATS approach, every block is trained to generate signals for both backcast and forecast outputs. Here, the backcast output is designed to be subtracted from the input of the subsequent block, whereas the forecasts are combined to produce the final prediction (Figure 2). These blocks are arranged in stacks, each focusing on a distinct spatial dependency through a unique set of graph structures.

3.3.1 Signal decomposition module

Recent research has witnessed a surging interest in disentangling time series data into its trend and seasonal components. These components respectively represent the overall long-term pattern and the seasonal fluctuations within the time signals. However, when it comes to future time series, directly performing this decomposition becomes impractical due to the inherent uncertainty of the future. To address this challenge, we propose the incorporation of a signal decomposition module within

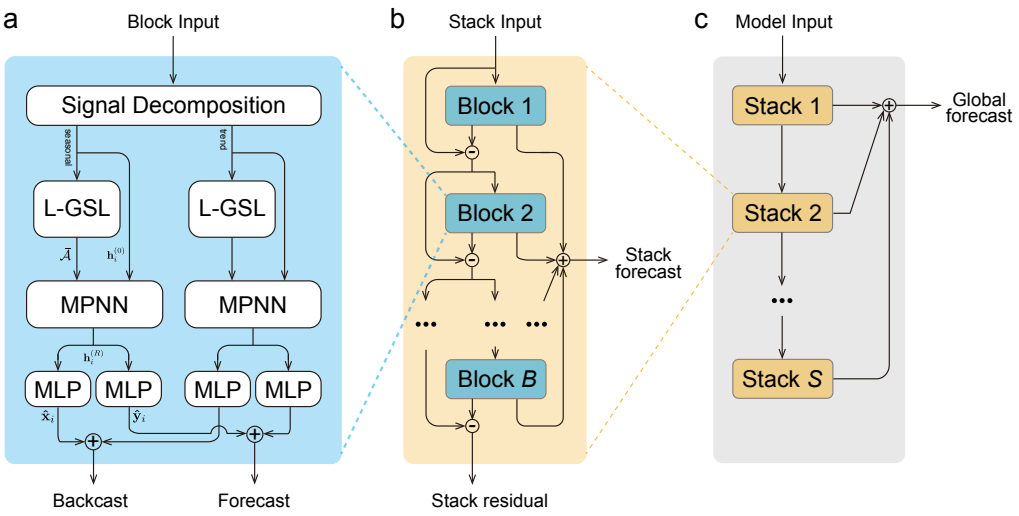

Figure 2: **Overview of the proposed HGMTS model architecture. (a)** The hierarchical residual block is marked by signal decomposition and GNN-centric L-GSL modules. **(b)** The combination of multiple blocks forms a stack, **(c)** culminating in the entire model design to ultimately produce a global forecasting output.

a single block (Figure 2a). This module enables the gradual extraction of the consistent, long-term trend from intermediate forecasting signals. Specifically, we employ the moving average technique to smooth out recurring fluctuations and uncover the underlying long-term trends as outlined below:

$$\mathbf{X}^{\texttt{trend}} = \mathrm{AvgPool}(\mathrm{Padding}(\mathbf{X})), \quad \mathbf{X}^{\texttt{seas}} = \mathbf{X} - \mathbf{X}^{\texttt{trend}} \tag{3}$$

where $\mathbf{X}^{\texttt{trend}}, \mathbf{X}^{\texttt{seas}}$ denote the trend and seasonal components respectively. We opt for the $\mathrm{AvgPool}(\cdot)$ for the moving average, accompanied by the zero padding operation to maintain the original series length intact.

3.3.2 Message-passing module

The message-passing module receives as input the past L time steps of both seasonal and trend outputs $\mathbf{X}^{\texttt{seas}}_{t-L:t}, \mathbf{X}^{\texttt{trend}}_{t-L:t} \in \mathbb{R}^{N \times L}$ obtained from the signal decomposition. As the two components go through the same set of distinct parameterized network modules, their differentiation will be disregarded henceforth. At each time step t, the input consisting of N multivariate time series with L lags are transformed into embedding vectors $\mathbf{H} \in \mathbb{R}^{N \times D}$ using a multilayer perceptron (MLP). Each row vector $\mathbf{h}_i$ in this matrix represents an individual node embedding. Subsequently, these node embeddings are employed in Eq.1 of the latent graph structure learning module to create a sparse adjacency matrix $\bar{\mathcal{A}}$. This matrix, in conjunction with the node embedding matrix, serves as the input for the message-passing neural network (Figure 2a). To be more specific, the r-th round of message passing in the GNN is executed using the following equations:

$$\mathbf{h}_i^{(0)} = f(\mathbf{x}_{i,t-L:t}) \tag{4}$$

$$\bar{\mathcal{A}} = \text{L-GSL}(\mathbf{H}) \tag{6}$$

$$\mathbf{m}_{ij}^{(r)} = g(\mathbf{h}_i^{(r)} - \mathbf{h}_j^{(r)}) \tag{5}$$

$$\mathbf{h}_i^{(r+1)} = \mathrm{GRU}(\mathbf{h}_i^{(r)}, \textstyle\sum_{j \in \mathcal{N}(i)} \bar{a}_{ij} \cdot \mathbf{m}_{ij}^{(r)}) \tag{7}$$

where $\mathbf{h}_i^{(r)}$ refers to the i-th node embedding after round r, and $\mathbf{m}_{ij}^{(r)}$ represents the message vector from node i to j. The interaction strength associated with the edge (i, j), denoted as $\bar{a}_{ij}$, corresponds to the entry in $\bar{\mathcal{A}}$ at the i-th row and j-th column. Both the encoding function $f(\cdot)$ and the message function $g(\cdot)$ are implemented as two-layer MLPs with ReLU nonlinearities. Finally, the node embeddings are updated using a GRU after aggregating all incoming messages through a weighted sum over the neighborhood $\mathcal{N}(i)$ for each node i. This sequence of operations is repeated separately for the seasonal and trend inputs, with no sharing of parameters (Figure 2a).

To enhance both the model's expressivity and its capacity for generalization, we employ a multi-module GNN framework [32]. More specifically, the next hidden state $\mathbf{h}_i^{(r+1)}$ is computed by blending two intermediate node states, $\mathbf{h}_{i,1}^{(r)}$ and $\mathbf{h}_{i,2}^{(r)}$, through a linear combination defined as follows:

$$\mathbf{h}_i^{(r+1)} \;=\; \beta_i^{(r)}\mathbf{h}_{i,1}^{(r)} + (1-\beta_i^{(r)})\mathbf{h}_{i,2}^{(r)} \tag{8}$$

where the two intermediate representations $\mathbf{h}_{i,1}^{(r)}$ and $\mathbf{h}_{i,2}^{(r)}$ are derived from Eq. (7) using two distinct GRUs. The value of the gating variable $\beta_i^{(r)}$ is determined by another processing unit employing a gating function ξ_g, which is a neural network producing a scalar output through a sigmoid activation.

3.3.3 Forecast and backcast module

Following the completion of the last R round of message passing (3 rounds in total), the backcast $\hat{\mathbf{x}}$ and forecast outputs $\hat{\mathbf{y}}$ are generated in this procedure. This is achieved by mapping the final node embeddings through separate two MLPs. These MLPs are responsible for handling the generation of backcast and forecast outputs individually (Figure 2a). It is important to note that the last layer of these MLPs is designed as a linear layer. This process of generating backcast and forecast outputs is applied to both the seasonal and trend pathways, and the ultimate backcast and forecast outputs are obtained by summing up the respective outputs from the seasonal and trend components (Figure 2a):

$$\hat{\mathbf{x}}_{i,t-L:t}^{\texttt{seas}} = \phi_{\texttt{seas}}(\mathbf{h}_{i,\texttt{seas}}^{(R)}) \qquad \hat{\mathbf{y}}_{i,t+1:t+K}^{\texttt{seas}} = \psi_{\texttt{seas}}(\mathbf{h}_{i,\texttt{seas}}^{(R)}) \tag{9}$$

$$\hat{\mathbf{x}}_{i,t-L:t}^{\texttt{trend}} = \phi_{\texttt{trend}}(\mathbf{h}_{i,\texttt{trend}}^{(R)}) \qquad \hat{\mathbf{y}}_{i,t+1:t+K}^{\texttt{trend}} = \psi_{\texttt{trend}}(\mathbf{h}_{i,\texttt{trend}}^{(R)}) \tag{10}$$

$$\hat{\mathbf{x}}_{i,t-L:t} = \hat{\mathbf{x}}_{i,t-L:t}^{\texttt{seas}} + \hat{\mathbf{x}}_{i,t-L:t}^{\texttt{trend}} \qquad \hat{\mathbf{y}}_{i,t+1:t+K} = \hat{\mathbf{y}}_{i,t+1:t+K}^{\texttt{seas}} + \hat{\mathbf{y}}_{i,t+1:t+K}^{\texttt{trend}} \tag{11}$$

Here, $\phi_\square$ and $\psi_\square$ represent two-layer MLPs designed to acquire the predictive decomposition of the partial backcast $\hat{\mathbf{x}}_{i,t-L:t}$ of the preceding L time steps, and the forecast $\hat{\mathbf{y}}_{i,t+1:t+K}$ of the subsequent K time steps. These MLPs operate on components denoted as $\square$, which can be either the `seasonal` or `trend` aspects. Note that the indexing related to block or stack levels has been excluded for clarity. The resulting global forecast is constructed by summing the outputs of all blocks (Figure 2b-c).

4 Experimental Setup

We first provide an overview of the datasets (Table 1), evaluation metrics, and baselines employed to quantitatively assess our model's performance. The main results are summarized in Table 2, demonstrating the competitive predictive performance of our approach in comparison to existing works. We then elaborate on the specifics of our training and evaluation setups followed by detailing the ablation studies.

4.1 Datasets

Our experimentation extensively covers six real-world benchmark datasets. Conforming to the standard protocol [33, 21], the split of all datasets into training, validation, and test sets has been conducted chronologically, following a split ratio of 60:20:20 for the ETTm_2 dataset and a split ratio of 70:10:20 for the remaining datasets.

- **ETTm_2 (Electricity Transformer Temperature)**: This dataset encompasses data obtained from electricity transformers, featuring load and oil temperatures recorded every 15 minutes during the period spanning from July 2016 to July 2018.
- **ECL (Electricity Consuming Load)**: The ECL dataset compiles hourly electricity consumption (in Kwh) data from 321 customers, spanning the years 2012 to 2014.
- **Exchange**: This dataset aggregates daily exchange rates of eight different countries relative to the US dollar. The data spans from 1990 to 2016.
- **Traffic**: The Traffic dataset is a collection of road occupancy rates from 862 sensors situated along San Francisco Bay area freeways. These rates are recorded every hour, spanning from January 2015 to December 2016.

- **Weather**: This dataset comprises 21 meteorological measurements, including air temperature and humidity. These measurements are recorded every 10 minutes throughout the entirety of the year 2020 in Germany.
- **ILI (Influenza-Like Illness)**: This dataset provides a record of weekly influenza-like illness (ILI) patients and the total patient count, sourced from the Centers for Disease Control and Prevention of the US. The data covers the extensive period from 2002 to 2021. It represents the ratio of ILI patients versus the total count for each week.

4.2 Evaluation metrics

We evaluate the effectiveness of our approach by measuring its accuracy using the mean squared error (MSE) and mean absolute error (MAE) metrics. These evaluations are conducted for various prediction horizon lengths $K \in \{96, 192, 336, 720\}$ given a fixed input length $L = 96$, except for ILI where $L = 36$:

$$\text{MSE} = \frac{1}{NK}\sum_{i=1}^{N}\sum_{\tau=t}^{t+K}\left(\mathbf{y}_{i,\tau} - \hat{\mathbf{y}}_{i,\tau}\right)^2, \quad \text{MAE} = \frac{1}{NK}\sum_{i=1}^{N}\sum_{\tau=t}^{t+K}\left|\mathbf{y}_{i,\tau} - \hat{\mathbf{y}}_{i,\tau}\right| \tag{12}$$

4.3 Baselines

We evaluate our proposed model by comparing it with seven baseline models. These include: (1) N-BEATS [12], which aligns with the external structure of our model, (2) Autoformer [33], (3) Informer [21], (4) Reformer [20], (5) LogTrans [31] – latest transformer-based models. Additionally, we compare with two conventional RNN-based models: (6) LSTNet [34] and (7) LSTM [35].

4.4 Hyperparameters

Our model is trained using the ADAM optimizer, starting with a learning rate of 10^{-4} that gets reduced by half every two epochs. We employ early stopping during training, stopping the process if there is no improvement after 10 epochs. The training is carried out with a batch size of 32. We have configured our model with 3 stacks, each containing 1 block. All tests are conducted three times, making use of the PyTorch framework, and are executed on a single NVIDIA RTX 3090 with 24GB GPU.

5 Experimental Results

5.1 Multivariate time series forecasting

In the multivariate setting, our proposed model, HGMTS, consistently achieves state-of-the-art performance across all benchmark datasets and prediction length configurations (Table 2). Notably, under the input-96-predict-192 setting, HGMTS demonstrates significant improvements over previous state-of-the-art results, with a 34% (0.273→0.180) reduction in MSE for ETT, 19% (0.180→0.146) reduction for ECL, 53% (0.225→0.105) reduction for Exchange, 5% (0.409→0.389) reduction for Traffic, and 10% (0.229→0.207) reduction for Weather. In the case of the input-36-predict-60 setting for ILI, HGMTS achieves 17% (2.547→2.118) reduction in MSE. Overall, HGMTS delivers an average MSE reduction of 23% across these settings. It is particularly striking how HGMTS drastically improves predictions for the Exchange dataset, where it records an average MSE reduction of 52% for all prediction lengths. Moreover, HGMTS stands out for its outstanding long-term stability, an essential attribute for real-world applications.

5.2 Effect of sparsity in graphs on forecasting

Within the HGMTS model framework, a key hyperparameter is the sampling factor in L-GSL. This factor determines how many query nodes are selected and subsequently linked to key nodes. For the sake of simplicity, we ensure that the number of chosen query and key nodes remains the same. We then measure the sparsity of the latent graphs by computing the proportion of selected pivotal query or key nodes relative to the total time series count. This proportion is denoted as $\gamma = \lfloor c \cdot \log N \rfloor / N$ and acts as an indicator of the sparsity in building these latent graphs.

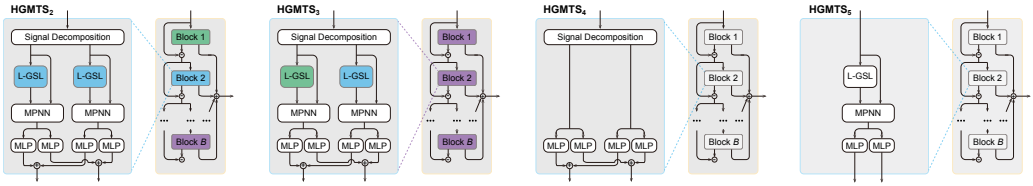

Figure 3: **Ablation study overview.** Displayed are four distinct model architectures explored to understand the impact of specific components on overall LSTF performance.

To understand the impact of sparsity in the learned graphs, we modify γ values between 0.2 and 0.7 and then document the findings from the multivariate forecasting studies. As detailed in Table 2, there is a consistent trend: all the graphs lean towards sparse interactions ($\gamma \leq 0.5$), targeting optimal predictive outcomes in LSTF tasks. Additionally, different benchmark datasets exhibit unique preferences regarding the optimal sparsity level for predictive performance, as displayed in Table 2.

5.3 Ablation studies

We posit that the strengths of the HGMTS architecture stem from its ability to hierarchically model the interplay between time series, particularly in the realms of trend and seasonality components. To delve deeper into this proposition, we present a series of control models for a comparative analysis:

- HGMTS_1: The model as showcased in Figure 2.
- HGMTS_2: A model that has shared latent graphs between trend and seasonality channels, but not across different blocks and stacks.
- HGMTS_3: A model where latent graphs are shared throughout all blocks and stacks but remain distinct between trend and seasonality channels.
- HGMTS_4: This model omits the L-GSL and MPNN modules.
- HGMTS_5: A model focusing solely on either the trend or seasonality channel, essentially lacking the signal decomposition module.
- HGMTS_6: A model that has used a single GRU module in Eq (8).

Under the same multivariate setting, the evaluation metrics for each control model, averaged over all benchmark datasets excluding ILI, are detailed in Table 4. The HGMTS_4, which forgoes the L-GSL and MPNN modules, experiences a noticeable average MSE surge of 30% (0.258→0.336) across all horizons. This rise is the most significant among all controls, indicating that capturing interdependencies between multivariate signals is vital in our suggested model. HGMTS_5, which emphasizes solely on a single channel between trend and seasonality, registers the second most pronounced MSE growth (18%: 0.258→0.305), suggesting that signal decomposition is also instrumental in LSTF tasks. Sharing the latent graphs – whether between the trend and seasonality pathways (as in HGMTS_2) or among blocks (as in HGMTS_3) – does elevate the average MSE, but the rise is modest when compared with the first two control models. Additionally, our findings highlight that incorporating multiple node update mechanisms in MPNN, as seen in HGMTS_6, brings about a slight enhancement in forecasting precision.

The information presented in Table 4 robustly supports the idea that best performance is achieved by integrating both suggested components: the latent graph structure and hierarchical signal decomposition. This emphasizes their synergistic role in enhancing the accuracy of long sequence time series predictions. Furthermore, it is confirmed that crafting distinct latent associations between time series hierarchically, spanning both trend and seasonal channels, is instrumental in attaining improved prediction outcomes.

6 Conclusions

In this paper, we delved into the challenge of long-term multivariate time series forecasting, an area that has seen notable progress recently. However, the intricate temporal patterns often impede models from effectively learning reliable dependencies. In response, we introduce HGMTS, a spatio-temporal

multivariate time series forecasting model that incorporates a signal decomposition module and employs a latent graph structure learning as intrinsic operators. This unique approach allows for the hierarchical aggregation of long-term trend and seasonal information from intermediate predictions. Furthermore, we adopt a multi-module message-passing framework to enhance our model's capacity to capture diverse time series data from a range of heterogeneous sensors. This approach distinctly sets us apart from previous neural forecasting models. Notably, HGMTS naturally achieves a computational complexity of $\mathcal{O}(N \log N)$ and consistently delivers state-of-the-art performance across a wide array of real-world datasets.

Learning a latent graph typically poses considerable challenges. Even though our model leverages the top-k pooling method to infer the latent graph, there are many other deep learning techniques that could be investigated in upcoming studies to uncover hidden structural patterns. Enhancements related to both representation capacity and computational efficiency might expand its broader adoption.

References

[1] Fotios Petropoulos, Daniele Apiletti, Vassilios Assimakopoulos, Mohamed Zied Babai, Devon K Barrow, Souhaib Ben Taieb, Christoph Bergmeir, Ricardo J Bessa, Jakub Bijak, John E Boylan, et al. Forecasting: theory and practice. *International Journal of Forecasting*, 2022.

[2] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Alexander Pritzel, Suman Ravuri, Timo Ewalds, Ferran Alet, Zach Eaton-Rosen, et al. Graphcast: Learning skillful medium-range global weather forecasting. *arXiv preprint arXiv:2212.12794*, 2022.

[3] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, 2021.

[4] Saeed Rahmani, Asiye Baghbani, Nizar Bouguila, and Zachary Patterson. Graph neural networks for intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[5] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 753–763, 2020.

[6] Chao Shang, Jie Chen, and Jinbo Bi. Discrete graph structure learning for forecasting multiple time series. *arXiv preprint arXiv:2101.06861*, 2021.

[7] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

[8] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learningachine learning*, pages 2688–2697. PMLR, 2018.

[9] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[10] Ailin Deng and Bryan Hooi. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4027–4035, 2021.

[11] Robert B Cleveland, William S Cleveland, Jean E McRae, and Irma Terpenning. Stl: A seasonal-trend decomposition. *Journal of Official Statistics*, 6(1):3–73, 1990.

[12] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2020.

[13] Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza Ramirez, Max Mergenthaler Canseco, and Artur Dubrawski. Nhits: Neural hierarchical interpolation for time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6989–6997, 2023.

[14] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. Long-term forecasting using tensor-train rnns. *Arxiv*, 2017.

[15] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *International Joint Conference on Artificial Intelligence*, 2017.

[16] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

[17] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3): 1181–1191, 2020.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[19] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[20] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgNKkHtvB`.

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

[22] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12270–12280, 2021.

[23] Gerald Woo, Chenghao Liu, Doyen Sahoo, Akshat Kumar, and Steven Hoi. Etsformer: Exponential smoothing transformers for time-series forecasting. *arXiv preprint arXiv:2202.01381*, 2022.

[24] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[25] Zhen Huang, Xu Shen, Xinmei Tian, Houqiang Li, Jianqiang Huang, and Xian-Sheng Hua. Spatio-temporal inception graph convolutional networks for skeleton-based action recognition. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 2122–2130, 2020.

[26] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

[27] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I 25*, pages 362–373. Springer, 2018.

[28] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems*, 21(9):3848–3858, 2019.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[30] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[31] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in Neural Information Processing Systems*, 32, 2019.

[32] HyunGeun Lee and Kijung Yoon. Towards better generalization with flexible representation of multi-module graph neural networks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=EYjfLeJL4l`.

[33] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.

[34] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 95–104, 2018.

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

7 Supplementary Material

Table 1: Summary statistics for the benchmark datasets used in our empirical study.

Dataset	Frequency	# Time Series	Input Length (L)	Horizon (K)
$ETTm_2$	15 Minute	7	96	$\{96, 192, 336, 720\}$
ECL	Hourly	321	96	$\{96, 192, 336, 720\}$
Exchange	Daily	8	96	$\{96, 192, 336, 720\}$
Traffic	Hourly	862	96	$\{96, 192, 336, 720\}$
Weather	10 Minute	21	96	$\{96, 192, 336, 720\}$
ILI	Weekly	7	36	$\{24, 36, 48, 60\}$

Table 2: Multivariate forecasting results for different prediction length $K \in \{96, 192, 336, 720\}$. For the ILI dataset, we set the input length (L) to 36, while for the other datasets, we set it to 96. A prediction is considered more precise if it has a lower MSE or MAE value. The metrics are the average of three trials, with the best results highlighted in bold for emphasis.

Models		HGMTS		N-BEATS[12]		Autoformer[33]		Informer[21]		LongTrans[31]		Reformer[20]		LSTNet[34]		LSTM[35]	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
$ETTm_2$	96	**0.145**	**0.232**	0.184	0.263	0.255	0.339	0.365	0.453	0.768	0.642	0.658	0.619	3.142	1.365	2.041	1.073
	192	**0.180**	**0.311**	0.273	0.337	0.281	0.340	0.533	0.563	0.989	0.757	1.078	0.827	3.154	1.369	2.249	1.112
	336	**0.227**	**0.349**	0.309	0.355	0.339	0.372	1.363	0.887	1.334	0.872	1.549	0.972	3.160	1.369	2.568	1.238
	720	**0.280**	**0.398**	0.411	0.425	0.422	0.419	3.379	1.388	3.048	1.328	2.631	1.242	3.171	1.368	2.720	1.287
ECL	96	**0.128**	**0.226**	0.145	0.247	0.201	0.317	0.274	0.368	0.258	0.357	0.312	0.402	0.680	0.645	0.375	0.437
	192	**0.146**	**0.249**	0.180	0.283	0.222	0.334	0.296	0.386	0.266	0.368	0.348	0.433	0.725	0.676	0.442	0.473
	336	**0.175**	**0.277**	0.200	0.308	0.231	0.338	0.300	0.394	0.280	0.380	0.350	0.433	0.828	0.727	0.439	0.473
	720	**0.238**	**0.332**	0.266	0.362	0.254	0.361	0.373	0.439	0.283	0.376	0.340	0.420	0.957	0.811	0.980	0.814
Exchange	96	**0.055**	**0.172**	0.098	0.206	0.197	0.323	0.847	0.752	0.968	0.812	1.065	0.829	1.551	1.058	1.453	1.049
	192	**0.105**	**0.242**	0.225	0.329	0.300	0.369	1.204	0.895	1.040	0.851	1.188	0.906	1.477	1.028	1.846	1.179
	336	**0.182**	**0.334**	0.493	0.482	0.509	0.524	1.672	1.036	1.659	1.081	1.357	0.976	1.507	1.031	2.136	1.231
	720	**0.560**	**0.609**	1.108	0.804	1.447	0.941	2.478	1.310	1.941	1.127	1.510	1.016	2.285	1.243	2.984	1.427
Traffic	96	**0.371**	**0.264**	0.398	0.282	0.613	0.388	0.719	0.391	0.684	0.384	0.732	0.423	1.107	0.685	0.843	0.453
	192	**0.389**	**0.281**	0.409	0.293	0.616	0.382	0.696	0.379	0.685	0.390	0.733	0.420	1.157	0.706	0.847	0.453
	336	**0.439**	**0.302**	0.449	0.318	0.622	0.377	0.777	0.420	0.733	0.408	0.742	0.420	1.216	0.730	0.853	0.455
	720	**0.577**	**0.386**	0.589	0.391	0.660	0.408	0.864	0.472	0.717	0..396	0.755	0.423	1.481	0.805	1.500	0.805
Weather	96	**0.146**	**0.185**	0.167	0.203	0.266	0.336	0.300	0.384	0.458	0.490	0.689	0.596	0.594	0.587	0.369	0.406
	192	**0.207**	**0.236**	0.229	0.261	0.307	0.367	0.598	0.544	0.658	0.589	0.752	0.638	0.560	0.565	0.416	0.435
	336	**0.268**	**0.291**	0.287	0.304	0.359	0.395	0.578	0.523	0.797	0.652	0.639	0.596	0.597	0.587	0.455	0.454
	720	**0.348**	**0.351**	0.368	0.359	0.419	0.428	1.059	0.741	0.869	0.675	1.130	0.792	0.618	0.599	0.535	0.520
ILI	24	**1.827**	**0.839**	1.879	0.886	3.483	1.287	5.764	1.677	4.480	1.444	4.400	1.382	6.026	1.770	5.914	1.734
	36	**2.034**	**0.903**	2.210	1.018	3.103	1.148	4.755	1.467	4.799	1.467	4.783	1.448	5.340	1.668	6.631	1.845
	48	**2.102**	**0.915**	2.440	1.088	2.669	1.085	4.763	1.469	4.800	1.468	4.832	1.465	6.080	1.787	6.736	1.857
	60	**2.118**	**0.956**	2.547	1.057	2.770	1.125	5.264	1.564	5.278	1.560	4.882	1.483	5.548	1.720	6.870	1.879

Table 3: The HGMTS performance evaluated under various selections of the graph sparsity hyper-parameter γ. The forecasting setup remains consistent with what is presented in Table 2.

Sparsity (γ)		0.2		0.3		0.4		0.5		0.6		0.7	
Metric		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
$ETTm_2$	96	0.151	0.240	0.149	0.238	0.146	0.234	**0.145**	**0.232**	–	–	0.146	0.235
	192	0.187	0.324	0.183	0.315	0.181	0.313	**0.180**	**0.311**	–	–	1.182	0.314
	336	0.239	0.364	0.234	0.358	0.230	0.353	**0.227**	**0.349**	–	–	0.229	0.352
	720	0.292	0.418	0.286	0.407	0.283	0.402	**0.280**	**0.398**	–	–	0.282	0.401
ECL	96	**0.128**	**0.226**	0.130	0.229	0.133	0.234	0.138	0.241	0.145	0.249	0.156	0.263
	192	**0.146**	**0.249**	0.149	0.253	0.152	0.257	0.158	0.266	0.164	0.274	0.169	0.280
	336	**0.175**	**0.277**	0.177	0.280	0.181	0.285	0.189	0.294	0.193	0.301	0.198	0.307
	720	**0.238**	**0.332**	0.240	0.335	0.243	0.338	0.247	0.345	0.252	0.351	0.256	0.359
Exchange	96	–	–	0.056	0.173	**0.055**	**0.172**	0.057	0.174	0.058	0.176	0.061	0.180
	192	–	–	0.107	0.244	**0.105**	**0.242**	0.106	0.244	0.107	0.246	0.109	0.249
	336	–	–	0.184	0.336	**0.182**	**0.334**	0.183	0.336	0.184	0.338	0.187	0.341
	720	–	–	0.563	0.613	**0.560**	**0.609**	0.562	0.611	0.564	0.613	0.567	0.617
Traffic	96	**0.371**	**0.264**	0.374	0.268	0.377	0.272	0.381	0.276	0.386	0.280	0.391	0.287
	192	**0.389**	**0.281**	0.394	0.286	0.398	0.291	0.405	0.299	0.412	0.308	0.423	0.316
	336	**0.439**	**0.302**	0.445	0.309	0.451	0.316	0.460	0.327	0.463	0.332	0.466	0.338
	720	**0.577**	**0.386**	0.581	0.392	0.584	0.395	0.589	0.401	0.594	0.407	0.598	0.414
Weather	96	0.147	0.186	**0.146**	**0.185**	0.147	0.187	0.149	0.190	0.150	0.191	0.152	0.193
	192	0.208	0.238	**0.207**	**0.236**	0.209	0.238	0.212	0.240	0.215	0.244	0.218	0.248
	336	0.270	0.293	**0.268**	**0.291**	0.269	0.292	0.271	0.294	0.274	0.298	0.277	0.301
	720	0.350	0.354	**0.348**	**0.351**	0.349	0.353	0.352	0.356	0.354	0.359	0.357	0.364
ILI	24	1.832	0.845	1.830	0.842	1.828	0.840	**1.827**	**0.839**	–	–	1.829	0.842
	36	2.041	0.911	2.037	0.906	2.036	0.905	**2.034**	**0.903**	–	–	2.035	0.905
	48	2.113	0.926	2.108	0.922	2.105	0.918	**2.102**	**0.915**	–	–	2.104	0.918
	60	2.123	0.964	2.122	0.962	2.120	0.959	**2.118**	**0.956**	–	–	2.119	0.959

Table 4: Empirical evaluation of long sequence time series forecasts for HGMTS. MAE and MSE are averaged over three runs and five datasets, with the best result highlighted in bold and the second best in blue.

		$HGMTS_1$	$HGMTS_2$	$HGMTS_3$	$HGMTS_4$	$HGMTS_5$	$HGMTS_6$
A. MSE	96	**0.168**	0.171	0.170	0.195	0.183	0.169
	192	**0.205**	0.209	0.208	0.261	0.232	0.206
	336	**0.258**	0.263	0.271	0.344	0.309	0.264
	720	**0.401**	0.412	0.428	0.545	0.496	0.414
A. MAE	96	**0.214**	0.219	0.218	0.237	0.229	0.216
	192	**0.264**	0.268	0.266	0.296	0.286	0.265
	336	**0.311**	0.316	0.328	0.349	0.337	0.318
	720	**0.415**	0.418	0.421	0.464	0.435	0.420