A Notation

Table 5: Notations

X_t	multivariate time series input at timestamps t, $X \in \mathbb{R}^{N \times T}$
x_t	the multivariate values of N series at timestamp t, $x_t \in \mathbb{R}^N$
Y_t	the next τ timestamps of multivariate time series, $Y_t \in \mathbb{R}^{N \times \tau}$
$\hat{Y}_t$	the prediction values of multivariate time series for next τ timestamps, $\hat{Y}_t \in \mathbb{R}^{N \times \tau}$
N	the number of series
T	the lookback window size
τ	the prediction length of multivariate time series forecasting
$\mathcal{G}_t$	the hypervariate graph, $\mathcal{G}_t = \{X_t^{\mathcal{G}}, A_t^{\mathcal{G}}\}$ attributed to $X_t^{\mathcal{G}}$
$X_t^{\mathcal{G}}$	the nodes of the hypervariate graph, $X_t^{\mathcal{G}} \in \mathbb{R}^{NT \times 1}$
$A_t^{\mathcal{G}}$	the adjacency matrix of $\mathcal{G}_t$, $A_t^{\mathcal{G}} \in \mathbb{R}^{NT \times NT}$
$\mathcal{S}$	the Fourier Graph Operator
d	the embedding dimension
$\mathbf{X}_t^{\mathcal{G}}$	the embedding of $X_t^{\mathcal{G}}$, $\mathbf{X}_t^{\mathcal{G}} \in \mathbb{R}^{NT \times d}$
$\mathcal{X}_t^{\mathcal{G}}$	the spectrum of $\mathbf{X}_t^{\mathcal{G}}$, $\mathcal{X}_t^{\mathcal{G}} \in \mathbb{C}^{NT \times d}$
$\mathcal{Y}_t^{\mathcal{G}}$	the output of FourierGNN, $\mathcal{Y}_t^{\mathcal{G}} \in \mathbb{C}^{NT \times d}$
θ_g	the parameters of the graph network
θ_t	the parameters of the temporal network
$\theta_{\mathcal{G}}$	the network parameters for hypervariate graphs
E_ϕ	the embedding matrix, $E_\phi \in \mathbb{R}^{1 \times d}$
κ	the kernel function
W	the weight matrix
b	the complex bias weights
$\mathcal{F}$	Discrete Fourier Transform
$\mathcal{F}^{-1}$	Inverse Discrete Fourier Transform
F	the forecasting model

B Convolution Theorem

The convolution theorem [26] is one of the most important properties of the Fourier transform. It states the Fourier transform of a convolution of two signals equals the pointwise product of their Fourier transforms in the frequency domain. Given a signal $x[n]$ and a filter $h[n]$, the convolution theorem can be defined as follows:

$$\mathcal{F}((x * h)[n]) = \mathcal{F}(x)\mathcal{F}(h) \tag{9}$$

where $(x * h)[n] = \sum_{m=0}^{N-1} h[m]x[(n-m)_N]$ denotes the convolution of x and h, $(n-m)_N$ denotes $(n-m)$ modulo N, and $\mathcal{F}(x)$ and $\mathcal{F}(h)$ denote discrete Fourier transform of $x[n]$ and $h[n]$, respectively.

C Explanations and Proofs

C.1 The Explanations of the Hypervariate Graph Structure

Note that the time lag effect between time-series variables is a common phenomenon in real-world multivariate time series scenarios, for example, the time lag influence between two financial assets (e.g. dollar and gold) of a portfolio. It is beneficial but challenging to consider dependencies between different variables under different timestamps.

The hypervariate graph connecting any two variables at any two timestamps aims to encode high-resolution spatiotemporal dependencies. It embodies not only the intra-series temporal dependencies (node connections of each individual variable), inter-series spatial dependencies (node connections under each single time step), and also the time-varying spatiotemporal dependencies (node connections between different variables at different time steps). By leveraging the hypervariate graph structure, we can effectively learn the spatial and temporal dependencies. This approach is distinct from previous methods that represent the spatial and temporal dependencies separately using two network structures.

C.2 The Interpretation of n-invariant FGO

Why $\mathcal{F}(\kappa) \in \mathbb{C}^{n\times d\times d}$? From Definition 2, we know that the kernel κ is defined as a matrix-valued projection, i.e., $\kappa : [n] \times [n] \to \mathbb{R}^{d\times d}$. Note that we assume κ is in the special case of the Green's kernel, i.e., a translation-invariant kernel $\kappa[i, j] = \kappa[i - j]$. Accordingly, κ can be reduced: $\kappa : [n] \to \mathbb{R}^{d\times d}$ where we can parameterize $\mathcal{F}(\kappa)$ with a complex-valued matrix $\mathbb{C}^{n\times d\times d}$.

What is n-invariant FGO? Turning to our case of the fully-connected hypervariate graph, we can consider a special case of κ, i.e., a space-invariant kernel $\kappa[i, j] = \kappa[\varrho]$ with ϱ being a constant scalar. Accordingly, we can parameterize FGO $\mathcal{S}$ with a n-invariant complex-valued matrix $\mathbb{C}^{d\times d}$.

The interpretation of n-invariant FGO. An n-invariant FGO is similar to a shared-weight convolution kernel or filter of CNNs that slide along ($[n] \times [n]$) input features, which effectively reduces parameter volumes and saves computation costs. Note that although we adopt the same transformation (i.e., the n-invariant FGO) over NT frequency points, we embed the raw MTS inputs in the d-dimension distributive space beforehand and then perform FourierGNN over MTS embeddings, which can be analogized as d convolution kernels/filters in each convolutional layer in CNNs. This can ensure FourierGNN is able to learn informative features/patterns to improve its model capacity (See the following analysis of the effectiveness of n-invariant FGO).

The effectiveness of n-invariant FGO. In addition, the n-invariant parameterized FGO is empirically proven effective to improve model generalization and achieve superior forecasting performance (See the ablation study in Section 5.3 for more details). Although parameterizing $\mathcal{F}(\kappa) \in \mathbb{C}^{n\times d\times d}$ (i.e., an n-variant FGO) may be more powerful and flexible than the n-invariant FGO in terms of forecasting performance, it introduces much more parameters and training time costs, especially in case of multi-layer FourierGNN, and may obtain inferior performance due to inadequate training or overfitting. As indicated in Table 6, the FourierGNN with the n-invariant FGO achieves slightly better performance than that with the n-variant FGO on ECG and COVID-19, respectively. Notably, the FourierGNN with the n-variant FGO introduces a much larger parameter volume proportional to n and requires significantly more training time. In contrast, n-invariant FGO is n-agnostic and lightweight, which is a more wise and efficient alternative. These results confirm our design and verify the effectiveness and applicability of n-invariant FGO.

Table 6: Comparison between FourierGNN models with n-invariant FGO and n-variant FGO on the ECG and COVID-19 datasets.

Datasets	Models	Parameters (M)	Training (s/epoch)	MAE	RMSE	MAPE (%)
ECG	n-invariant	**0.18**	**12.45**	**0.052**	0.078	**10.97**
	n-variant	82.96	104.06	0.053	0.078	11.05
COVID-19	n-invariant	**1.06**	**0.62**	**0.123**	**0.168**	**71.52**
	n-variant	130.99	7.46	0.129	0.174	72.12

C.3 Proof of Proposition 1 and Interpretation of FourierGNN

Proposition 1. *Given a graph $G = (X, A)$ with node features $X \in \mathbb{R}^{n\times d}$ and adjacency matrix $A \in \mathbb{R}^{n\times n}$, the recursive multiplication of FGOs in Fourier space is equivalent to multi-order convolutions in the time domain:*

$$\mathcal{F}^{-1}(\mathcal{F}(X)\mathcal{S}_{0:k}) = A_{k:0}XW_{0:k}, \quad s.t.\ \mathcal{S}_{0:k} = \prod_{i=0}^{k}\mathcal{S}_i, A_{k:0} = \prod_{i=k}^{0}A_i, W_{0:k} = \prod_{i=0}^{k}W_i$$

where $A_0, \mathcal{S}_0, W_0$ are the identity matrix, $A_k \in \mathbb{R}^{n\times n}$ corresponds to the k-th diffusion step sharing the same sparsity pattern of A, $W_k \in \mathbb{R}^{d\times d}$ is the k-th weight matrix, $\mathcal{S}_k \in \mathbb{C}^{d\times d}$ is the k-th FGO satisfying $\mathcal{F}(A_kXW_k) = \mathcal{F}(X)\mathcal{S}_k$, and $\mathcal{F}$ and $\mathcal{F}^{-1}$ denote DFT and its inverse, respectively.

Proof. The proof aims to demonstrate the equivalence between the recursive multiplication of FGOs in Fourier space and multi-order convolutions in the time domain. According to $\mathcal{F}(A_kXW_k) = \mathcal{F}(X)\mathcal{S}_k$, we expand the multi-order convolutions $A_{0:K}XW_{0:K}$ in the time domain using a set of FGOs in Fourier space:

$$\begin{aligned}
\mathcal{F}(A_KA_{K-1}\cdots A_0XW_0\cdots W_{K-1}W_K) &= \mathcal{F}(A_K(A_{K-1}...A_0XW_0\cdots W_{K-1})W_K)\\
&= \mathcal{F}(A_{K-1}...A_0XW_0\cdots W_{K-1})\mathcal{S}_K\\
&= \mathcal{F}(A_{K-1}(A_{K-2}...A_0XW_0\cdots W_{K-2})W_{K-1})\mathcal{S}_K\\
&= \mathcal{F}(A_{K-2}...A_0XW_0\cdots W_{K-2})\mathcal{S}_{K-1}\mathcal{S}_K\\
&= \cdots\\
&= \mathcal{F}(X)\mathcal{S}_0\cdots\mathcal{S}_{K-1}\mathcal{S}_K\\
&= \mathcal{F}(X)\mathcal{S}_{0:K}
\end{aligned} \tag{10}$$

where it yields $\mathcal{F}^{-1}(\mathcal{F}(X)\mathcal{S}_{0:K}) = A_{K:0}XW_{0:K}$ with $\mathcal{S}_{0:K} = \prod_{i=0}^{K}\mathcal{S}_i$,$A_{K:0} = \prod_{i=K}^{0}A_i$ and $W_{0:K} = \prod_{i=0}^{K}W_i$. Proved. □

Thus, the FourierGNN can be rewritten as (for convenience, we exclude the non-linear activation function σ and learnable bias parameters b):

$$\mathcal{F}^{-1}(\sum_{k=0}^{K}\mathcal{F}(X)\mathcal{S}_{0:K}) = A_0XW_0 + A_1(A_0XW_0)W_1 + ... + A_{K:0}XW_{0:K} \tag{11}$$

From the right part of the above equation, we can observe that it assigns different weights to weigh the information of different neighbors in each diffusion order. This property enable FourierGNN to capture time-varying correlations, which is empirically verified in our experiments (See Appendix H.2 for more details).

D Compared with Other Graph Neural Networks

Graph Convolutional Networks. Graph convolutional networks (GCNs) depend on the Laplacian eigenbasis to perform the multi-order graph convolutions over a given graph structure. Compared with GCNs, FourierGNN as an efficient alternative to multi-order graph convolutions has three main differences: 1) No eigendecompositions or similar costly matrix operations are required. FourierGNN transforms the input into Fourier domain by discrete Fourier transform (DFT) instead of graph Fourier transform (GFT); 2) Explicitly assigning various importance to nodes of the same neighborhood with different diffusion steps. FourierGNN adopts different Fourier Graph Operators $\mathcal{S}$ in different diffusion steps corresponding to the time-varying dependencies among nodes; 3) FourierGNN is invariant to the discretization N, T. It parameterizes the graph convolution via Fourier bases invariant graph structure and graph scale.

Graph Attention Networks. Graph attention networks (GATs) are non-spectral attention-based graph neural networks. GATs use node representations to calculate the attention weights (i.e., edge weights) varying with different graph attention layers. Accordingly, both GATs and FourierGNN do

not depend on eigendecompositions and adopt varying edge weights with different diffusion steps (layers). However, FourierGNN can efficiently perform graph convolutions in the Fourier space. For a complete graph, the time complexity of the attention calculation of K layers is proportional to Kn^2 where n is the number of nodes, while a K-layer FourierGNN infers the graph structure in Fourier space with the time complexity proportional to $n \log n$. In addition, compared with GATs that implicitly achieve edge-varying weights with different layers, FourierGNN adopts different FGOs in different diffusion steps explicitly.

E Experiment Details

E.1 Datasets

We use seven public multivariate benchmarks for multivariate time series forecasting and these benchmark datasets are summarized in Table 7.

Table 7: Summary of datasets.

Datasets	Solar	Wiki	Traffic	ECG	Electricity	COVID-19	METR-LA
Samples	3650	803	10560	5000	140211	335	34272
Variables	592	2000	963	140	370	55	207
Granularity	1hour	1day	1hour	-	15min	1day	5min
Start time	01/01/2006	01/07/2015	01/01/2015	-	01/01/2011	01/02/2020	01/03/2012

Solar[1]: This dataset is about solar power collected by National Renewable Energy Laboratory. We choose the power plant data points in Florida as the data set which contains 593 points. The data is collected from 2006/01/01 to 2016/12/31 with the sampling interval of every 1 hour.

Wiki[2]: This dataset contains a number of daily views of different Wikipedia articles and is collected from 2015/7/1 to 2016/12/31. It consists of approximately $145k$ time series and we randomly choose $2k$ from them as our experimental data set.

Traffic[3]: This dataset contains hourly traffic data from 963 San Francisco freeway car lanes. The traffic data are collected since 2015/01/01 with the sampling interval of every 1 hour.

ECG[4]: This dataset is about Electrocardiogram(ECG) from the UCR time-series classification archive [34]. It contains 140 nodes and each node has a length of 5000.

Electricity[5]: This dataset contains the electricity consumption of 370 clients and is collected since 2011/01/01. The data sampling interval is every 15 minutes.

COVID-19[6]: This dataset is about COVID-19 hospitalization in the U.S. states of California (CA) from 01/02/2020 to 31/12/2020 provided by the Johns Hopkins University with the sampling interval of every one day.

METR-LA[7]: This dataset contains traffic information collected from loop detectors in the highway of Los Angeles County from 01/03/2012 to 30/06/2012. It contains 207 sensors and the data sampling interval is every 5 minutes.

E.2 Baselines

In experiments, we conduct a comprehensive comparison of the forecasting performance between our FourierGNN and representative and state-of-the-art (SOTA) models as follows.

[1] https://www.nrel.gov/grid/solar-power-data.html
[2] https://www.kaggle.com/c/web-traffic-time-series-forecasting/data
[3] https://archive.ics.uci.edu/ml/datasets/PEMS-SF
[4] http://www.timeseriesclassification.com/description.php?Dataset=ECG5000
[5] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014
[6] https://github.com/CSSEGISandData/COVID-19
[7] https://github.com/liyaguang/DCRNN

VAR [29]: VAR is a classic linear autoregressive model. We use the Statsmodels library (`https://www.statsmodels.org`) which is a Python package that provides statistical computations to realize the VAR.

DeepGLO [31]: DeepGLO models the relationships among variables by matrix factorization and employs a temporal convolution neural network to introduce non-linear relationships. We download the source code from: `https://github.com/rajatsen91/deepglo`. We follow the recommended configuration as our experimental settings for wiki, electricity, and traffic datasets. For covid datasets, the vertical and horizontal batch size is set to 64, the rank of the global model is set to 64, the number of channels is set to [32, 32, 32, 1], and the period is set to 7.

LSTNet [30]: LSTNet uses a CNN to capture inter-variable relationships and an RNN to discover long-term patterns. We download the source code from: `https://github.com/laiguokun/LSTNet`. In our experiment, we use the recommended configuration where the number of CNN hidden units is 100, the kernel size of the CNN layers is 4, the dropout is 0.2, the RNN hidden units is 100, the number of RNN hidden layers is 1, the learning rate is 0.001 and the optimizer is Adam.

TCN [6]: TCN is a causal convolution model for regression prediction. We download the source code from: `https://github.com/locuslab/TCN`. We utilize the same configuration as the polyphonic music task exampled in the open source code where the dropout is 0.25, the kernel size is 5, the number of hidden units is 150, the number of levels is 4 and the optimizer is Adam.

Reformer [32]: Reformer combines the modeling capacity of a Transformer with an architecture that can be executed efficiently on long sequences and with small memory use. We download the source code from: `https://github.com/thuml/Autoformer`. We follow the recommended configuration as the experimental settings.

Informer [7]: Informer leverages an efficient self-attention mechanism to encode the dependencies among variables. We download the source code from: `https://github.com/zhouhaoyi/Informer2020`. We follow the recommended configuration as our experimental settings where the dropout is 0.05, the number of encoder layers is 2, the number of decoder layers is 1, the learning rate is 0.0001, and the optimizer is Adam.

Autoformer [8]: Autoformer proposes a decomposition architecture by embedding the series decomposition block as an inner operator, which can progressively aggregate the long-term trend part from intermediate prediction. We download the source code from: `https://github.com/thuml/Autoformer`. We follow the recommended configuration as our experimental settings with 2 encoder layers and 1 decoder layer.

FEDformer [20]: FEDformer proposes an attention mechanism with low-rank approximation in frequency and a mixture of expert decomposition to control the distribution shifting. We download the source code from: `https://github.com/MAZiqing/FEDformer`. We use FEB-f as the Frequency Enhanced Block and select the random mode with 64 as the experimental mode.

SFM [22]: On the basis of the LSTM model, SFM introduces a series of different frequency components in the cell states. We download the source code from: `https://github.com/z331565360/State-Frequency-Memory-stock-prediction`. We follow the recommended settings where the learning rate is 0.01, the frequency dimension is 10, the hidden dimension is 10 and the optimizer is RMSProp.

StemGNN [4]: StemGNN leverages GFT and DFT to capture dependencies among variables in the frequency domain. We download the source code from: `https://github.com/microsoft/StemGNN`. We use the recommended configuration of stemGNN as our experiment setting where the optimizer is RMSProp, the learning rate is 0.0001, the number of stacked layers is 5, and the dropout rate is 0.5.

MTGNN [11]: MTGNN proposes an effective method to exploit the inherent dependency relationships among multiple time series. We download the source code from: `https://github.com/nnzhan/MTGNN`. Because the experimental datasets have no static features, we set the parameter load_static_feature to false. We construct the graph by the adaptive adjacency matrix and add the graph convolution layer. Regarding other parameters, we adopt the recommended settings.

GraphWaveNet [15]: GraphWaveNet introduces an adaptive dependency matrix learning to capture the hidden spatial dependency. We download the source code from: `https://github.com/`

nnzhan/Graph-WaveNet. Since our datasets have no prior defined graph structures, we use only adaptive adjacent matrix. We add a graph convolution layer and randomly initialize the adjacent matrix. We adopt the recommended configuration as our experimental settings where the learning rate is 0.001, the dropout is 0.3, the number of epochs is 50, and the optimizer is Adam.

AGCRN [2]: AGCRN proposes a data-adaptive graph generation module for discovering spatial correlations from data. We download the source code from: https://github.com/LeiBAI/AGCRN. We follow the recommended configuration as our experimental settings where the embedding dimension is 10, the learning rate is 0.003, and the optimizer is Adam.

TAMP-S2GCNets [9]: TAMP-S2GCNets explores the utility of MP to enhance knowledge representation mechanisms within the time-aware DL paradigm. We download the source code from: https://www.dropbox.com/sh/n0ajd5l0tdeyb80/AABGn-ejfV1YtRwjf_L0AOsNa?dl=0. TAMP-S2GCNets requires predefined graph topology and we use the California State topology provided by the source code as input. We adopt the recommended configuration as our experimental settings on COVID-19.

DCRNN [16]: DCRNN uses bidirectional graph random walk to model spatial dependency and recurrent neural network to capture the temporal dynamics. We download the source code from: https://github.com/liyaguang/DCRNN. We follow the recommended configuration as our experimental settings with the batch size is 64, the learning rate is 0.01, the input dimension is 2 and the optimizer is Adam. DCRNN requires a pre-defined graph structure and we use the adjacency matrix as the pre-defined structure provided by the METR-LA dataset.

STGCN [1]: STGCN integrates graph convolution and gated temporal convolution through spatial-temporal convolutional blocks. We download the source code from:https://github.com/VeritasYin/STGCN_IJCAI-18. We use the recommended configuration as our experimental settings where the batch size is 50, the learning rate is 0.001 and the optimizer is Adam. STGCN requires a pre-defined graph structure and we leverage the adjacency matrix as the pre-defined structures provided by the METR-LA dataset.

CoST [19]: CoST separates the representation learning and downstream forecasting task and proposes a contrastive learning framework that learns disentangled season-trend representations for time series forecasting tasks. We download the source code from: https://github.com/salesforce/CoST. We set the representation dimension to 320, the learning rate to 0.001, and the batch size to 32. Inputs are min-max normalization, we perform a 70/20/10 train/validation/test split for the METR-LA dataset and 60/20/20 for the COVID-19 dataset.

E.3 Evaluation Metrics

We use MAE (Mean Absolute Error), RMSE (Root Mean Square Error), and MAPE (Mean Absolute Percentage Error) as the evaluation metrics in the experiments.

Specifically, given the groudtruth at timestamps t, $Y_t = [\boldsymbol{x}_{t+1}, ..., \boldsymbol{x}_{t+\tau}] \in \mathbb{R}^{N\times\tau}$, and the predictions $\hat{Y}_t = [\hat{\boldsymbol{x}}_{t+1}, ..., \hat{\boldsymbol{x}}_{t+\tau}] \in \mathbb{R}^{N\times\tau}$ for future τ steps at timestamp t, the metrics are defined as follows:

$$MAE = \frac{1}{\tau N}\sum_{i=1}^{N}\sum_{j=1}^{\tau}|x_{ij} - \hat{x}_{ij}| \tag{12}$$

$$RMSE = \sqrt{\frac{1}{\tau N}\sum_{i=1}^{N}\sum_{j=1}^{\tau}(x_{ij} - \hat{x}_{ij})^2} \tag{13}$$

$$MAPE = \frac{1}{\tau N}\sum_{i=1}^{N}\sum_{j=1}^{\tau}\left|\frac{x_{ij} - \hat{x}_{ij}}{x_{ij}}\right| \times 100\% \tag{14}$$

with $x_{ij} \in Y_t$ and $\hat{x}_{ij} \in \hat{Y}_t$.

E.4 Experimental Settings

We summarize the implementation details of the proposed FourierGNN as follows. Note that the details of the baselines are introduced in their corresponding descriptions (see Section E.2).

Network details. The fully connected feed-forward network (FFN) consists of three linear transformations with $LeakyReLU$ activations in between. The FFN is formulated as follows:

$$\begin{aligned}\mathbf{X}_1 &= \text{LeakyReLU}(\mathbf{Y}_t^{\mathcal{G}}\mathbf{W}_1 + \mathbf{b}_1)\\ \mathbf{X}_2 &= \text{LeakyReLU}(\mathbf{X}_1\mathbf{W}_2 + \mathbf{b}_2)\\ \hat{Y} &= \mathbf{X}_2\mathbf{W}_3 + \mathbf{b}_3\end{aligned} \tag{15}$$

where $\mathbf{W}_1 \in \mathbb{R}^{(Td)\times d_1^{ffn}}$, $\mathbf{W}_2 \in \mathbb{R}^{d_1^{ffn}\times d_2^{ffn}}$ and $\mathbf{W}_3 \in \mathbb{R}^{d_2^{ffn}\times\tau}$ are the weights of the three layers respectively, and $\mathbf{b}_1 \in \mathbb{R}^{d_1^{ffn}}$, $\mathbf{b}_2 \in \mathbb{R}^{d_2^{ffn}}$ and $\mathbf{b}_3 \in \mathbb{R}^{\tau}$ are the biases of the three layers respectively. Here, d_1^{ffn} and d_2^{ffn} are the dimensions of the three layers. In addition, we adopt a $ReLU$ activation function in Equation 6.

Training details. We carefully tune the hyperparameters, including the embedding size, batch size, d_1^{ffn} and d_2^{ffn}, on the validation set and choose the settings with the best performance for FourierGNN on different datasets. Specifically, the embedding size and batch size are tuned over $\{32, 64, 128, 256, 512\}$ and $\{2, 4, 8, 16, 32, 64, 128\}$ respectively. For the COVID-19 dataset, the embedding size is 256, and the batch size is set to 4. For the Traffic, Solar, and Wiki datasets, the embedding size is 128, and the batch size is set to 2. For the METR-LA, ECG, and Electricity datasets, the embedding size is 128, and the batch size is set to 32.

To reduce the number of parameters, we adopt a linear transform to reshape the original time domain representation $\mathbf{Y}_t^{\mathcal{G}} \in \mathbb{R}^{NT\times d}$ to $\mathbf{Y}_t \in \mathbb{R}^{N\times T\times d}$, and map $\mathbf{Y}_t$ to a low-dimensional tensor $\mathbf{Y}_t \in \mathbb{R}^{N\times l\times d}$ with $l < T$. We then reshape $\mathbf{Y}_t \in \mathbb{R}^{N\times(ld)}$ and feed it to FFN. We perform a grid search on the dimensions of FFN, i.e., d_1^{ffn} and d_2^{ffn}, over $\{32, 64, 128, 256, 512\}$ and tune the intermediate dimension l over $\{2, 4, 6, 8, 12\}$. The settings of the three hyperparameters over all datasets are shown in Table 8. By default, we set the diffusion step (layers) $K = 3$ for all datasets.

Table 8: Dimension settings of FFN on different datasets. $*$ denotes that we feed the original time domain representation to FFN without the dimension reduction.

Datasets	Solar	Wiki	Traffic	ECG	Electricity	COVID-19	META-LR
l	6	2	2	$*$	4	8	4
d_1^{ffn}	64	64	64	64	64	256	64
d_2^{ffn}	256	256	256	256	256	512	256

E.5 Details for Visualization Experiments

To verify the effectiveness of FourierGNN in learning the spatiotemporal dependencies on the hypervariate graph, we obtain the output of FourierGNN as the node representation, denoted as $\mathbf{Y}_t^{\mathcal{G}} = \text{IDFT}(\text{FourierGNN}(\mathbf{X}_t^{\mathcal{G}})) \in \mathbb{R}^{NT\times d}$ with Inverse Discrete Fourier Transform (IDFT). Then, we visualize the adjacency matrix $\mathbf{A}$ calculated based the flatten node representation $\mathbf{Y}_t^{\mathcal{G}} \in \mathbb{R}^{NT\times d}$, formulated as $\mathbf{A} = \mathbf{Y}_t^{\mathcal{G}}(\mathbf{Y}_t^{\mathcal{G}})^T \in \mathbb{R}^{NT\times NT}$, to show the variable correlations. Note that $\mathbf{A}$ is normalized via $\mathbf{A}/\max(\mathbf{A})$. Since it is not feasible to directly visualize the huge adjacency matrix $\mathbf{A}$ of the hypervariate graph, we visualize its different subgraphs in Figures 3, 4, 9, and 10 to better verify the learned spatiotemporal information on the hypervariate graph from different perspectives.

Figure 3. We select 8 counties and visualize the correlations between 12 consecutive time steps for each selected county respectively. Figure 3 reflects the temporal correlations within each variable.

Figure 4: On the METR-LA dataset, we average its adjacency matrix $\mathbf{A}$ over the temporal dimension (i.e., marginalizing T) to $\mathbf{A}' \in \mathbb{R}^{N\times N}$. Then, we randomly select 20 detectors out of all $N = 207$ detectors and obtain their corresponding sub adjacency matrix ($\mathbb{R}^{20\times 20}$) from $\mathbf{A}'$ for visualization. We further compare the sub-adjacency with the real road map (generated by the Google map tool) to verify the learned dependencies between different detectors.

Figure 9. Since we adopt a 3-layer FourierGNN, we can calculate four adjacency matrices based on the spectrum input $\mathcal{X}_t^{\mathcal{G}}$ of FourierGNN and the outputs of each layer in FourierGNN. Following the

way of visualization in Figure 4, we select 10 counties and two timestamps on the four adjacency matrices for visualization. Figure 9 shows the effects of each layer of FourierGNN in filtering or enhancing variable correlations.

Figure 10. On the COVID-19 dataset, we randomly select 10 counties out of $N = 55$ counties and obtain their four sub-adjacency matrices of four consecutive days for visualization. Each of the four sub adjacency matrices $\mathbb{R}^{10\times10}$ embodies the dependencies between counties in one day. Figure 10 reflects the time-varying dependencies between counties (i.e., variables).

F Additional Results

To further evaluate the performance of our model FourierGNN in multi-step forecasting, we conduct more experiments on the Wiki, METR-LA, and ECG datasets, respectively. We compare our model FourierGNN with five models (including StemGNN [4], AGCRN [2], GraphWaveNet [15], MTGNN [11], and Informer [7]) on the Wiki dataset under different prediction lengths, and the results are shown in Table 9. From the table, we observe that FourierGNN outperforms other models on MAE, RMSE, and MAPE metrics for all the prediction lengths. On average, FourierGNN improves MAE, RMSE, and MAPE by 6.8%, 3.2%, and 22.9%, respectively. Among these models, AGCRN shows promising performances since it captures the spatial and temporal correlations adaptively. However, it fails to simultaneously capture spatiotemporal dependencies, limiting its forecasting performance. In contrast, our model captures comprehensive spatiotemporal dependencies simultaneously on a hypervariate graph for multivariate time series forecasting.

Table 9: Accuracy comparison under different prediction lengths on the Wiki dataset.

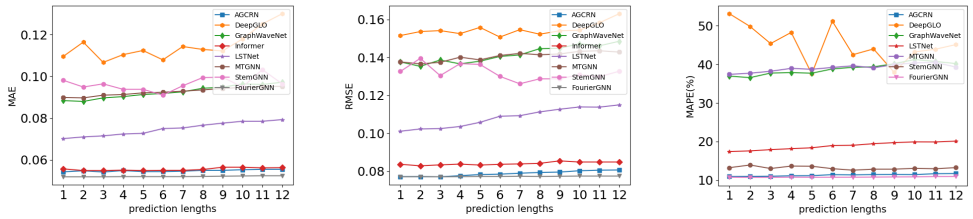
Length	3			6			9			12		
Metrics	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)
GraphWaveNet [15]	0.061	0.105	138.60	0.061	0.105	135.32	0.061	0.105	132.52	0.061	0.104	136.12
StemGNN [4]	0.157	0.236	89.00	0.159	0.233	98.01	0.232	0.311	142.14	0.220	0.306	125.40
AGCRN [2]	0.043	0.077	73.49	0.044	0.078	80.44	0.045	0.079	81.89	0.044	0.079	78.52
MTGNN [11]	0.102	0.141	123.15	0.091	0.133	91.75	0.074	0.120	85.44	0.101	0.140	122.96
Informer [7]	0.053	0.089	85.31	0.054	0.090	84.46	0.059	0.095	93.80	0.059	0.095	95.09
FourierGNN	**0.040**	**0.075**	**58.18**	**0.041**	**0.075**	**60.43**	**0.041**	**0.076**	**60.95**	**0.042**	**0.077**	**62.62**

Table 10: Accuracy comparison under different prediction lengths on the METR-LA dataset.

Horizon	3			6			9			12		
Metrics	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)	MAE	RMSE	MAPE(%)
DCRNN [16]	0.160	0.204	80.00	0.191	0.243	83.15	0.216	0.269	85.72	0.241	0.291	88.25
STGCN [1]	0.058	0.133	59.02	0.080	0.177	60.67	0.102	0.209	62.08	0.128	0.238	63.81
GraphWaveNet [15]	0.180	0.366	21.90	0.184	0.375	22.95	0.196	0.382	23.61	0.202	0.386	24.14
MTGNN [11]	0.135	0.294	**17.99**	0.144	0.307	**18.82**	0.149	0.328	**19.38**	0.153	0.316	**19.92**
StemGNN [4]	0.052	0.115	86.39	0.069	0.141	87.71	0.080	0.162	89.00	0.093	0.175	90.25
AGCRN [2]	0.062	0.131	24.96	0.086	0.165	27.62	0.099	0.188	29.72	0.109	0.204	31.73
Informer [7]	0.076	0.141	69.96	0.088	0.163	70.94	0.096	0.178	72.26	0.100	0.190	72.54
CoST [19]	0.064	0.118	88.44	0.077	0.141	89.63	0.088	**0.159**	90.56	0.097	0.171	91.42
FourierGNN	**0.050**	**0.113**	86.30	**0.066**	**0.140**	87.97	**0.076**	**0.159**	88.99	**0.084**	**0.165**	89.69

Furthermore, we compare our model FourierGNN with seven MTS models (including STGCN [1], DCRNN [16], StemGNN [4], AGCRN [2], GraphWaveNet [15], MTGNN [11], Informer [7], and CoST [19]) on the METR-LA dataset which has a predefined graph topology in the data, and the results are shown in Table 10. On average, we improve 5.7% on MAE and 2.5% on RMSE. Among these models, StemGNN achieves competitive performance because it combines GFT to capture the spatial dependencies and DFT to capture the temporal dependencies. However, it is also limited to simultaneously capturing spatiotemporal dependencies. CoST learns disentangled seasonal-trend representations for time series forecasting via contrastive learning and obtains competitive results. But, our model still outperforms CoST. Because, compared with CoST, our model not only can learn the dynamic temporal representations, but also capture the discriminative spatial representations. Besides, STGCN and DCRNN require pre-defined graph structures. But StemGNN and our model outperform them for all steps, and AGCRN outperforms them when the prediction lengths are 9 and 12. This also shows that a novel adaptive graph learning can precisely capture the hidden spatial dependency. In addition, we compare FourierGNN with the baseline models under the different

prediction lengths on the ECG dataset, as shown in Figure 5. It reports that FourierGNN achieves the best performances (MAE, RMSE, and MAPE) for all prediction lengths.



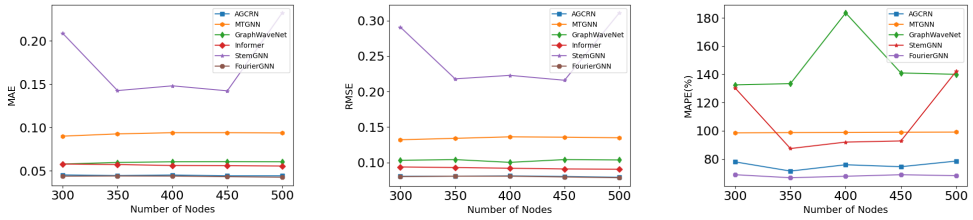
(a) MAE (b) RMSE (c) MAPE

Figure 5: Performance comparison in multi-step prediction on the ECG dataset.

G Further Analyses

G.1 Scalability Analysis

We further conduct experiments on the Wiki dataset to investigate the performance of FourierGNN under different graph sizes ($N \times T$). The results are shown in Figure 6, where Figure 6(a), Figure 6(b) and Figure 6(c) show MAE, RMSE, and MAPE at the different number of nodes, respectively. From these figures, we observe that FourierGNN keeps a leading edge over the other state-of-the-art MTS models as the number of nodes increases. The results demonstrate the superiority and scalability of FourierGNN on large-scale datasets.



(a) MAE (b) RMSE (c) MAPE

Figure 6: Scalability analyses in terms of MAE, RMSE, and MAPE under different number of nodes on the Wiki dataset.

G.2 Parameter Analysis

We evaluate the forecasting performance of our model FourierGNN under different diffusion steps (layers) on the COVID-19 dataset, as illustrated in Table 11. The table shows that FourierGNN achieves increasingly better performance from $K = 1$ to $K = 4$ and achieves the best results when $K = 3$. With the further increase of K, FourierGNN obtains inferior performance. The results indicate that high-order diffusion information is beneficial for improving forecasting accuracy, but the diffusion information may gradually weaken the effect or even bring noises to forecasting with the increase of the order.

Table 11: Performance at different diffusion steps (layers) on the COVID-19 dataset.

	K=1	K=2	K=3	K=4
MAE	0.136	0.133	<u>0.129</u>	0.132
RMSE	0.181	0.177	<u>0.173</u>	0.176
MAPE(%)	72.30	71.80	<u>71.52</u>	72.59

In addition, we conduct additional experiments on the ECG dataset to analyze the effect of the input lookback window length T and the embedding dimension d, as shown in Figure 7 and Figure 8, respectively. Figure 7 shows that the performance (including RMSE and MAPE) of FourierGNN

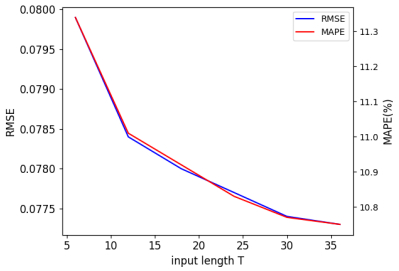


Figure 7: Influence of input window.

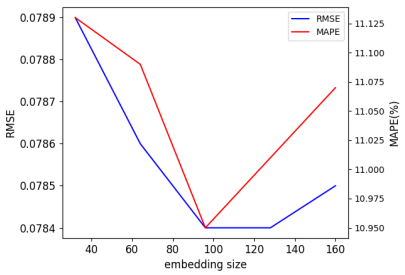


Figure 8: Influence of embedding size.

gets better as the input lookback window length increases, indicating that FourierGNN can learn a comprehensive hypervariate graph from long MTS inputs to capture the spatial and temporal dependencies. Moreover, Figure 8 shows that the performance (RMSE and MAPE) first increases and then decreases with the increase of the embedding size, which is attributed that a large embedding size improves the fitting ability of FourierGNN but it may easily lead to the overfitting issue especially when the embedding size is too large.

G.3 Ablation Study

We provide more details about each variant used in this section and Section 5.3.

- **w/o Embedding**. A variant of FourierGNN feeds the raw MTS input instead of its embeddings into the graph convolution in the Fourier space.
- **w/o Dynamic FGO**. A variant of FourierGNN uses the same FGO for all diffusion steps instead of applying different FGOs in different diffusion steps. It corresponds to a vanilla graph filter.
- **w/o Residual**. A variant of FourierGNN does not have the $K = 0$ layer output, i.e., $\mathcal{X}_t^{\mathcal{G}}$, in the summation.
- **w/o Summation**. A variant of FourierGNN adopts the last order (layer) output as the final frequency output of the FourierGNN.

We conduct another ablation study on the COVID-19 dataset to further investigate the effects of the different components of our FourierGNN. The results are shown in Table 12, which confirms the results in Table 4 and further verifies the effectiveness of each component in FourierGNN. Both Table 12 and Table 4 report that the embedding and dynamic FGOs in FourierGNN contribute more than the design of residual and summation to the state-of-the-art performance of FourierGNN.

Table 12: Ablation studies on the COVID-19 dataset.

Metric	w/o Embedding	w/o Dynamic FGO	w/o Residual	w/o Summation	FourierGNN
MAE	0.157	0.138	0.131	0.134	<u>0.129</u>
RMSE	0.203	0.180	0.174	0.177	<u>0.173</u>
MAPE(%)	76.91	74.01	72.25	72.57	<u>71.52</u>

H Visualizations

H.1 Visualization of the Diffusion Process in FourierGNN

To gain insight into the operation of the FGO, we visualize the frequency output of each layer in our FourierGNN. We select 10 counties from the COVID-19 dataset and visualize their adjacency matrices at two different timestamps, as shown in Figure 9. From left to right, the results correspond to the original spectrum of the input, as well as the outputs of the first, second, and third layers of the FourierGNN. From the top, we can find that as the number of layers increases, some correlation

values are reduced, indicating that some correlations are filtered out. In contrast, the bottom case illustrates some correlations are enhanced as the number of layers increases. These results show that FGO can adaptively and effectively capture important patterns while removing noises, enabling the learning of a discriminative model.

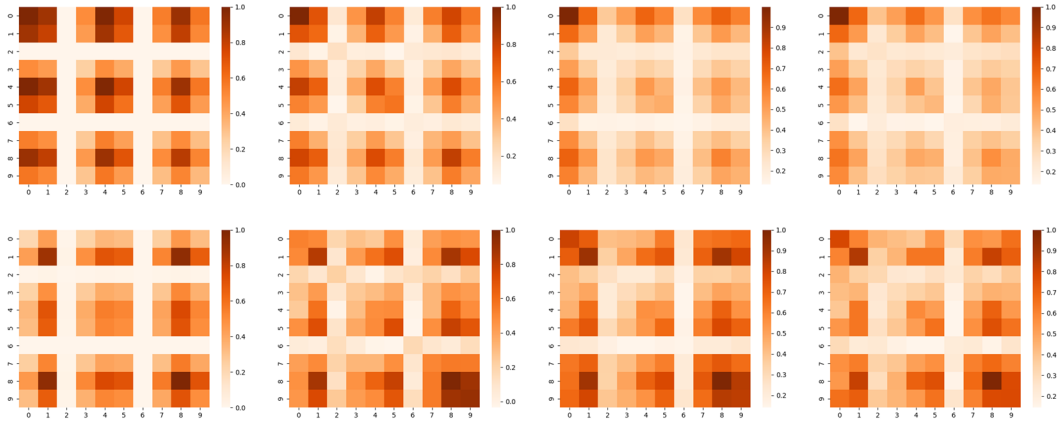


Figure 9: The diffusion process of FourierGNN at two timestamps (top and bottom) on COVID-19.

H.2 Visualization of Time-Varying Dependencies Learned by FourierGNN

Furthermore, we explore the capability of FourierGNN in capturing time-varying dependencies among variables. To investigate this, we perform additional experiments to visualize the adjacency matrix of 10 randomly-selected counties over four consecutive days on the COVID-19 dataset. The visualization results, displayed as a heatmap in Figure 10, reveal clear spatial patterns that exhibit continuous evolution in the temporal dimension. This is because FourierGNN can attend to the time-varying variability of the spatiotemporal dependencies. These results verify that our model enjoys the feasibility of exploiting the time-varying dependencies among variables.

Based on the insights gained from these visualization results, we can conclude that the hypervariate graph structure exhibits strong capabilities to encode spatiotemporal dependencies. By incorporating FGOs, FourierGNN can effectively attend to and exploit the time-varying dependencies among variates. The synergy between the hypervariate graph structure and FGOs empowers FourierGNN to capture and model intricate spatiotemporal relationships with remarkable effectiveness.

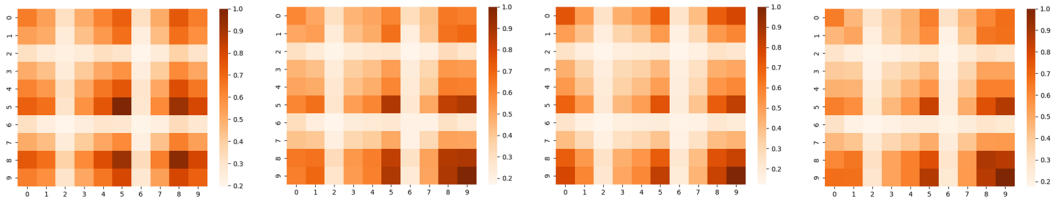


Figure 10: The adjacency matrix for four consecutive days on the COVID-19 dataset.