METRO: A Generic Graph Neural Network Framework for Multivariate Time Series Forecasting

Yue Cui[13†], Kai Zheng[1*], Dingshan Cui[2], Jiandong Xie[2], Liwei Deng[1†], Feiteng Huang[2], and Xiaofang Zhou[3]
[1]University of Electronic Science and Technology of China, China
[2]Huawei Cloud Database Innovation Lab, China
[3]The Hong Kong University of Science and Technology, China
{cuiyue,zhengkai}@uestc.edu.cn,{cuidingshan1,xiejiandong,huangfeiteng}@huawei.com
deng_liwei@std.uestc.edu.cn,zxf@cse.ust.hk

ABSTRACT

Multivariate time series forecasting has been drawing increasing attention due to its prevalent applications. It has been commonly assumed that leveraging latent dependencies between pairs of variables can enhance prediction accuracy. However, most existing methods suffer from static variable relevance modeling and ignorance of correlation between temporal scales, thereby failing to fully retain the dynamic and periodic interdependencies among variables, which are vital for long- and short-term forecasting. In this paper, we propose METRO, a generic framework with multi-scale temporal graphs neural networks, which models the dynamic and cross-scale variable correlations simultaneously. By representing the multivariate time series as a series of temporal graphs, both intra- and inter-step correlations can be well preserved via message-passing and node embedding update. To enable information propagation across temporal scales, we design a novel sampling strategy to align specific steps between higher and lower scales and fuse the cross-scale information efficiently. Moreover, we provide a modular interpretation of existing GNN-based time series forecasting works as specific instances under our framework. Extensive experiments conducted on four benchmark datasets demonstrate the effectiveness and efficiency of our approach. METRO has been successfully deployed onto the time series analytics platform of Huawei Cloud, where a one-month online test demonstrated that up to 20% relative improvement over state-of-the-art models w.r.t. RSE can be achieved.

PVLDB Reference Format:
Yue Cui, Kai Zheng, Dingshan Cui, Jiandong Xie, Liwei Deng, Feiteng Huang, and Xiaofang Zhou. METRO: A Generic Graph Neural Network Framework for Multivariate Time Series Forecasting. PVLDB, 15(2): XXX-XXX, 2022.
doi:10.14778/3489496.3489503

Proceedings of the VLDB Endowment, Vol. 15, No. 2 ISSN 2150-8097.
doi:10.14778/3489496.3489503

† The research was done when Yue Cui and Liwei Deng interned at Huawei Cloud Database Innovation Lab.
*Kai Zheng is the corresponding author.

1 INTRODUCTION

Multivariate time series (MTS) forecasting has greatly shaped our modern societies in tremendous scenarios ranging from stock markets, traffic flows to urban electricity consumption, and so forth, where people are interested in learning the dynamic tendencies or making predictive maintenance.

Traditionally, statistical methods, e.g., autoregressive integrated moving average (ARIMA) [1], vector autoregression (VAR), Gaussian process models (GP) [7, 35] and their variants [2], are used in time series forecasting. Recently, advanced machine learning methods significantly improved the accuracy of prediction by their flexibility in modeling nonlinear patterns [15, 34, 45]. Even more recently, to capture the interdependencies and dynamic trends over a group of variables, Graph Neural Networks (GNNs) have become increasingly popular in MTS forecasting where the variables are taken as nodes and interactions of variables can be modeled by edges [11, 22, 42]. It has also been proven that even for tasks where explicit graph structures do not naturally exist, there can be significant improvements when the hidden graph structures are extracted [42].

Despite the advance of GNNs in exploring potential interdependencies of variables, most existing works assume the graph structure of MTS to be static [29]. However, as depicted in Figure 1, the interdependencies between variables can be temporally dynamic. State-of-the-art MTS forecasting models MTGNN [42] and TEGNN [46] learn/compute adjacency matrices of graphs from the whole input time series and hold them for subsequent temporal modeling. ASTGCN [11], which is proposed for a more specialized task, i.e., traffic prediction, where connections between nodes are known in advance, considers adaptive adjacency matrix (attention value) but on a relatively large scale that only changes with regard to input series sample. In a similar problem setting, STDN [47] proposes a flow gating mechanism to learn dynamic similarity by time step, but the approach can only be applied on consecutive, and grid partitioned regional maps.

In real-world scenarios, time series can be expressed in multiple scales, e.g., daily, weekly, and monthly. Generally, mainstream models dealing with multi-scale patterns in two ways. One group of works, e.g, [52], treat time stamps of MTS as side-information and combine the embedded timestamps with MTS's embedding through operations such as summation. Another line of works, e.g., [42, 46], first adopt filters with different kernel sizes to the MTS

and then sum or concatenate the multi-scale features to get final predictions. Though to a certain degree, above mentioned models can incorporate information across different scales, they suffer from a common problem that the methods are essentially simple linear-weight transformation and thus the relation between pairs of scales are ignored. However, interacted scale features can have a much stronger influence than independent ones. For example, suppose one wants to know the flow of a metro station at 6 pm on July 15th, 2021 (a Thursday). The scales of time-related features are [hour: 6 pm, day: 15th, week: Thursday, month: July, year: 2021]. Intuitively, when making such predictions one will primarily consider the features **6 pm AND Thursday** jointly as they imply a rush hour on weekdays. In other words, single-scale features and features of monthly or yearly scales are less relevant to this prediction task.

To address the problems of 1) the dependencies between variables are dynamic and 2) temporal patterns in MTS occur in multiple scales, we propose a generic, end-to-end framework: **M**ulti-scal**E** **T**emporal g**R**aph neural netw**O**rks (METRO). It has four key components, a temporal graph embedding (TGE) unit, which obtains node (variable) embeddings under multiple time scales, a single-scale graph update (SGU) unit, which dynamically leverages graph structures of nodes in each single scale and propagates information from a node's intra- and inter-step neighbors, a cross-scale graph fusion (CGF) kernel, which walks along the time axis and fuses features across scales, and a predictor, which serves as a decoder that decodes node embeddings and gives final predictions. Overall, our contributions can be briefly summarized as follows:

- We propose METRO, a generic multi-scale temporal graph neural network framework that leverages both dynamic and cross-scale variable correlations for MTS forecasting and show that previous GNN-based MTS forecasting models can be interpreted as specific instances of our framework.
- To achieve effectiveness and maintain efficiency, a light-weight implementation of the framework is presented based on the attention mechanism.
- We show the effectiveness of the proposed framework through extensive experiments on four benchmark MTS datasets. Comprehensive ablation studies comparing different instances of our framework are conducted to verify the effectiveness of each key component.
- A one-month online prediction test in production environment showed that METRO brought up to 20 % relative improvements over state-of-the-art models w.r.t root square error (RSE).

2 PRELIMINARIES

Multivariate Time Series Forecasting: A multivariate time series (MTS) can be represented as $\mathcal{X} = \{X(1), X(2), ..., X(t)\}$, where $X(i) = [x_{i,1}, .., x_{i,n}] \in \mathbb{R}^{n \times m}, i \in [1, ..., t]$, is the observed multivariate variable at time step i, and $x_{i,k} \in \mathbb{R}^m, k \in [1, ..., n]$, is the representation value of the k^{th} variable at time step i, which is of dimension m. The goal of multivariate time series forecasting is to predict the representation value of the multivariate variable Δ steps ahead, i.e., $\hat{Y}(t+\Delta) = X(t+\Delta) = [x_{t+\Delta,1}, .., x_{t+\Delta,n}]$, using its historical sequence of window size w, i.e., $\{X(t-w+1), ..., X(t)\}$.

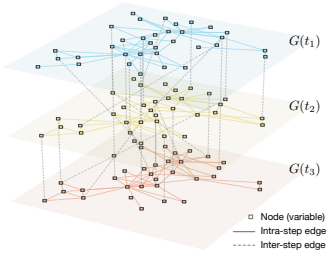


Figure 1: An example of MTS represented in temporal graphs. The variables are taken as nodes and the interaction between variables are taken as edges. The relation between nodes are temporally dynamic.

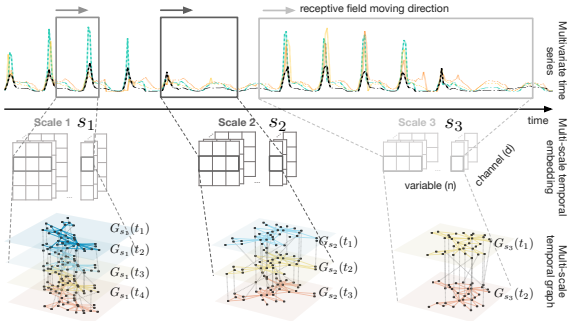


Figure 2: Examples of multi-scale temporal graphs. The node embeddings are obtained through multi-scale temporal graph embedding unit.

Definition 2.1. **Graph.** A graph $G = (V, E)$ is comprised of a set of vertices V and a set of edges E, which can be described by an adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$, where $A_{ij} > 0$ if there is an edge e_{ij} between vertex v_i and v_j and $A_{ij} = 0$ if there is no edge.

In our problem setting, variables are taken as vertices and the interactions between variables are modeled as edges. The above-defined graph can also be referred to as *Static Graph*, where vertices and edges are invariant over time. We further define a temporal graph as:

Definition 2.2. **Temporal Graph.** A temporal graph $G(t) = (V(t), E(t))$ is a graph as a function of time. It models a series of multivariate variable observations and can be viewed as a sequence of static graphs, where $V(t)$ and $E(t)$ denote the sets of vertices and edges at a particular time step t respectively.

There can be two types of temporal graphs in MTS forecasting. In *Intra-step Temporal Graph*, $V(t)$ is made up of variables within time step t and in *Inter-step Temporal Graph*, $V(t)$ consists variables of time step t and its previous time step $t-1$ ($t \geq 2$). Figure 1 illustrates examples of the two types of temporal graphs.

Definition 2.3. **Multi-scale Temporal Graph.** A temporal graph of scale s, denoted as $G_s(t) = (V_s(t), E_s(t))$, is a temporal graph where each node in the graph represents an observation of time-length w_s. Examples of multi-scale temporal graphs are depicted in Figure 2.

3 METRO

3.1 Temporal Graph Embedding Unit

The temporal graph embedding (TGE) unit serves as an encoder where embeddings of temporal features of each variable a under certain time scale are obtained.

For a multivariate time sequence of w length, i.e., $\{X(t_0), \ldots, X(t_0 + w - 1)\}$, its temporal feature at time step t, $t \in [t_0, ..., t_0 + w - 1]$, under scale s_i is obtained as:

$$\mathbf{H}^0_{s_i}(t) = emb(s_i, t) = f(X(t), ..., X(t + w_{s_i} - 1)), \tag{1}$$

where $\mathbf{H}^0_{s_i}(t) \in \mathbb{R}^{n \times d}$ is the obtained temporal features, scale $s_i \in \{s_1, .., s_p\}$ is a hyper-parameter to control the granularity of the receptive field, n is the number of variables, d denotes the embedding dimension, and $f(\cdot)$ is the embedding function. There can be multiple choices for implementation of such a function, e.g., simple concatenation of time steps followed by a linear transformation layer, RNNs such as LSTMs [14] or GRUs [3], where the scales can be controlled by feeding the network with input of corresponding length. In other alternatives such as CNNs, kernels with multiple receptive fields can be applied to obtain multi-scale temporal features. Since in our datasets, $X(t) \in \mathbb{R}^{n \times 1}$, for simplicity, we choose the embedding function as unpadded convolution with multiple filters with kernel size $1 \times w_{s_i}$, $i = 1, .., p$ and a shared stride S. Mathematically, the convolutional function can be expressed as:

$$\mathbf{H}^0_{s_i,j}(t) = \sigma(W_{s_i,j} * [X(t), ..., X(t + w_{s_i} - 1)] + b_j), \tag{2}$$

where $\sigma()$ denotes the sigmoid function, $W_{s_i,j}$ is the kernel of the j^{th} channel, $*$ is the convolution operator, [ ] denotes the concatenation operation and b_j is the convolution bias. Then the outputs of multiple convolution channels are concatenated to obtain $\mathbf{H}^0_{s_i}(t)$.

3.2 Single-scale Graph Update Unit

As proven by some of the recent works [42, 46], modeling hidden relations among variables by training and updating adjacency matrices of the graphs is promising. However, it is problematic to perform message aggregation over time with fixed global graph structures.

To deal with this problem, in the single-scale graph update (SGU) unit, we adaptively learn intra- and inter-step adjacency matrix of variables and propose a graph-based temporal message passing strategy that computes and aggregates historical temporal information of nodes which is then utilized to update node embeddings. Taking the multi-scale temporal graphs in Figure 2 as an example, SGU operates s_1, s_2, s_3 separately. For the temporal graphs of a single scale, e.g, $G_{s_1}(:)$ SGU jointly considers message passing within a time step, e.g., the connection between pairs nodes of $G_{s_1}(t_i)$ and between time steps, e.g., the connection between pairs of nodes of $G_{s_1}(t_1)$ and $G_{s_1}(t_2)$. Intuitively, in real-world scenarios such as traffic flow prediction, the first kind of message passing can simulate spatial correlations of sensors and the second resembles the flow transition.

Since the following descriptions in this subsection are all for a single scale, for simplicity, we omit the scale variable s in all related notations.

The SGU unit and the to-be-introduced cross-scale graph fusion (CGF) kernel can be combined and stacked multiple times, a.k.a. the multiple layers of the deep model. Since the network structures are the same for each of the stacked layers, in the following sections, we will take layer l as an example. For layer l, given the node embeddings of the most recent previous $k \in \mathbb{N}^+$ steps, i.e., $\mathbf{H}^l(t-k), ..., \mathbf{H}^l(t-1)$, and the to-be-updated node embeddings of current time step t, i.e., $\mathbf{H}^l(t)$, SGU unit first learns messages between nodes of two adjacent time steps as:

$$\begin{aligned} \mathbf{m}^l(t-j) &= msg(\mathbf{H}^l(t-j-1), \mathbf{H}^l(t-j), A^l_{t-j-1,t-j}), \\ A^l_{t-j-1,t-j} &= g_m(\mathbf{H}^l(t-j-1), \mathbf{H}^l(t-j)), \end{aligned} \tag{3}$$

where $\mathbf{m}^l(t-j)$ denotes learned message, $j \in [1, ..., k-1]$ msg is a learnable message function that models information be passed from $V(t-j-1)$ to $V(t-j)$, e.g., GCNs [20], and g_m is the graph learning function, which computes the adjacent matrix $A^l_{t-j-1,t-j} \in \mathbb{R}^{N_{t-j} \times N_{t-j}}$, where $N_{t-j} = |V(t-j-1)| + |V(t-j)|$. g_m can be implemented as transfer entropy [46], or node-embedding-based deep layer model [22] etc.

A message here can be viewed as intermediate information between two consecutive time step in range $[t-k, t-1]$. To pass all the messages to the current step, we take the learned message as a sequence and apply an aggregation function:

$$\widetilde{\mathbf{m}}^l(t) = agg(\mathbf{m}^l(t-k-1), .., \mathbf{m}^l(t-1)), \tag{4}$$

where agg is the learnable message aggregation function, which can be implemented as simple concatenation, or more complex models such as RNN and its variants or Transformer [39] based models, $\widetilde{\mathbf{m}}^l(t)$ denotes the aggregated message for the current layer, where all nodes appearing in $G(t-k-1), ..., G(t-1)$ are included and represented.

The embeddings of nodes in the current step t are then be updated based on the aggregated message and the memory of the previous layer:

$$\begin{aligned} \hat{\mathbf{H}}^{l+1}(t) &= upd(\widetilde{\mathbf{m}}^l(t), \mathbf{H}^l(t), A^l_t), \\ A^l_t &= g_u(\widetilde{\mathbf{m}}^l(t), \mathbf{H}^l(t)), \end{aligned} \tag{5}$$

the update function upd can be any memory update function e.g., GRUs [3] or LSTMs [14], g_u is the graph learning function that adaptively computes the adjacent matrix A^l_t. In DMGNN [22], a graph-based GRU is also proposed for such an update.

3.2.1 Contextual Update. Note that the above message passing and embedding update strategy is one-way: from previous steps to the current step, which is analogous to traditional memory update models such as GRUs and LSTMs and is efficient to be implemented. However, the dynamic change holds contextually, and for a specific time step in the given series, future information is often available. It is beneficial if not only previous information but future information is made use of. This can be achieved by extending $\mathbf{H}^l(t)$'s temporal neighbors to future $v \in \mathbb{N}^+$ steps. In this case, Equation (3) - (4) are still valid with $j \in [-v, ..., -1]$. We name this strategy as SGU-Contextual (SGU-C).

For the sake of efficiency, in the representative implementation of METRO, i.e., MTTRO-attn, we jointly implement Equation 3- 5 in SGU-C mode with multi-head attention (MultiHeadAttn) [39]. The adjacency matrix is accompanied with attention matrix, which dynamically adjusts the impacting weights between nodes. We use

a threshold to measure the significance of the attention, by which the attention values smaller than the threshold are set to 0s. Note that this is different from the selection strategy used in MTGNN [42], where nodes are connected with their top-k most relevant neighbors. Since top-k nodes are always connected regardless of the relevance scores, noisy messages may be propagated from a node's irrelevant neighbors (have low relevance score but in the top-k shortlist). Formally, the node embedding can be updated as:

$$\begin{aligned} \hat{\mathbf{H}}^{l+1}(t) &= MultiHeadAttn(\mathbf{Q}^l(t), \mathbf{K}^l, \mathbf{V}^l, r_1), \\ \mathbf{Q}^l(t) &= \mathbf{H}^l(t), \\ \mathbf{K}^l &= \mathbf{V}^l = [\mathbf{H}^l(t-k), ..., \mathbf{H}^l(t), ..., \mathbf{H}^l(t+v)], \end{aligned} \tag{6}$$

where r_1 is the attention value cutoff threshold.

3.3 Cross-scale Graph Fusion Kernel

To understand and diffuse information across scales, we propose the cross-scale graph fusion (CGF) kernel, which walks along the obtained embedding features of multiple scales and enables information fusion.

Scales are defined by the size of respective fields. Larger scales capture global but coarse features of the raw time series while smaller scales capture local but fine features. Taking METRO-attn's embedding function as an example, i.e., convolution, when the convolutional kernel walking down the time axis with a certain stride, each primitive time period, i.e., some continuous time-steps of the raw time series, is mapped to a certain time step in the output feature matrix, which is not temporally aligned across scales. It is desirable to conduct cross-scale fusion on features from multiple scales describing the same object. Thus, given the updated embeddings from the SGU unit, we need to first decide which time steps in which scales can interact.

We solve this problem by taking the time steps of higher scale as the standard and then sampling fine features (time steps) from lower scales. It is expected that the primitive time periods of the selected time steps of lower scales 1) can cover the standard's primitive time period; 2) are not overlapped.

A toy example of how to do the sampling can be found in the left part of Figure 4. Suppose the raw time series has been embedded in three scales with a shared stride S, i.e., s_1, s_2, s_3, where $4w_{s_1} = 2w_{s_2} = w_{s_3}$ and $S = w_{s_1}$. Since the SUG unit is omit, we only plot the updated embeddings in the second most left column, i.e., $\hat{\mathbf{H}}^{l+1}_{s_1}(:), \hat{\mathbf{H}}^{l+1}_{s_2}(:), \hat{\mathbf{H}}^{l+1}_{s_3}(:)$, which are illustrated as cubes and each of the cubic layer inside denotes the embeddings of variables of certain time steps. The CGF kernel takes the highest scale as standard, i.e., s_3. For a specific time step t, the sampling function $samp(\cdot)$ samples time steps from $\hat{\mathbf{H}}^{l+1}_{s_1}(:), \hat{\mathbf{H}}^{l+1}_{s_2}(:)$ such that the corresponding primitive time periods (circled by dashed grey lines in the most left column) of lower-scale time steps can cover the standard's (circled by solid red line) while are not overlapped.

Formally, for time step t of scale s_i, we sample the features from lower scales by:

$$\mathbf{Z}^l_{s_i^-}(t) = samp(\hat{\mathbf{H}}^{l+1}_{s_1}(:), ..., \hat{\mathbf{H}}^{l+1}_{s_{i-1}}(:), \hat{\mathbf{H}}^{l+1}_{s_i}(t)), \tag{7}$$

where $\hat{\mathbf{H}}^{l+1}_{s_j}(:)$ is the output updated features of SGU unit, $\mathbf{Z}^l_{s_i^-}$ is the selected lower scale features concatenated in time axis and

Algorithm 1 Network Training Algorithm

Input: Historical MTS $\mathcal{X}$, $\mathcal{Y}$, Δ, scales $s_1, ..., s_p$, other hyper-parameters
Output: Model parameters
Initialize model parameters
for batch $(X(t_0), ..., X(t_0 + w - 1)) \in$ training set **do**
  /* Temporal graph embedding */
  $\forall\ s_i \in [s_1, ..., s_p],\ \forall\ t \in [t_0, ..., t_0 + w - w_{s_i}],\ \mathbf{H}^0_{s_i}(t) \leftarrow emb(s_i, t)$
  for layer l in range $(nlayers)$ **do**
    /* Single-scale graph update */
    for $s_i \in [s_1, ..., s_p]$ **do**
      for $\forall t \le t_{s_i}$, in parallel or sequentially:
      /* t_{s_i} denotes the last time steps under s_i */
      $\mathbf{m}^l(1), ..., \mathbf{m}^l(t-1), \mathbf{m}^l(t+1), ..., \mathbf{m}^l(t_{s_i}-1) \leftarrow$ Calculate between-step message w.r.t. Equation 3
      $\hat{\mathbf{H}}^{l+1}_{s_i}(t) \leftarrow$ Update node embedding to obtain w.r.t. Equation 5
    end for
    /* Cross-scale graph fusion */
    for $s_i \in [s_1, ..., s_p]$ with lower scales satisfying Equation 8 **do**
      for $\forall t \le t_{s_i}$, in parallel or sequentially:
      $\mathbf{Z}^l_{s_i^-}(t) \leftarrow$ Sample time steps from lower scales w.r.t. Equation 7
      $\mathbf{H}^{l+1}_{s_1:s_{i-1}}(:), \mathbf{H}^{l+1}_{s_i}(t) \leftarrow$ Update embedding of certain time steps w.r.t. Equation 9
    end for
  end for
  $\hat{Y}(t_0 + w - 1 + \Delta) = pre(\mathbf{H}^{l+1}_{s_1}(:), ..., \mathbf{H}^{l+1}_{s_p}(:))$
  $\mathcal{L} = MSE(\hat{Y}(t_0 + w - 1 + \Delta), Y(t_0 + w - 1 + \Delta))$
  Perform gradient descend w.r.t model parameters
end for

samp denotes the sampling function, which can be implemented as slicing operation. Mathematically, to sample time steps mapping with un-overlapped primitive time period, a selectable lower scale s_j should satisfy:

$$\begin{cases} w_{s_j} = nS, & n = 1, 2, 3, ... \\ w_{s_i} = m w_{s_j}, & m = 2, 3, 4, ... \end{cases} \tag{8}$$

Based on above conditions, the sampled time slices are $\{\mathbf{H}^l_{s_j}(t_k) | t_k = t + k * n, k \in \{0, 1, ..., n(m-1))\}\}$. Note that above conditions actually impose restrictions on stride S and the window of scales, which means one needs to take those conditions into account when determining the hyper-parameters. In our implementations, we set S as 12, and apply three scales: $w_{s_1} = 24, w_{s_2} = 48, w_{s_3} = 96$.

To enable information diffusion between $\mathbf{Z}^l_{s_i^-}(t)$ and $\hat{\mathbf{H}}^{l+1}_{s_i}(t)$, we further use a graph learning function g_f to adaptively learn the adjacency matrix $A_{(s_i^-, s_i), t}$ between nodes of two parties and then fuse the information in bidirectional way:

$$\begin{aligned} \mathbf{H}^{l+1}_{s_1:s_{i-1}}(:), \mathbf{H}^{l+1}_{s_i}(t) &\leftarrow fuse(\mathbf{Z}^l_{s^-}(t), \hat{\mathbf{H}}^{l+1}_{s_i}(t), A^l_{(s_i^-, s_i), t}), \\ A^l_{(s_i^-, s_i), t} &= g_f(\mathbf{Z}^l_{s_i^-}(t), \hat{\mathbf{H}}^{l+1}_{s_i}(t)), \end{aligned} \tag{9}$$

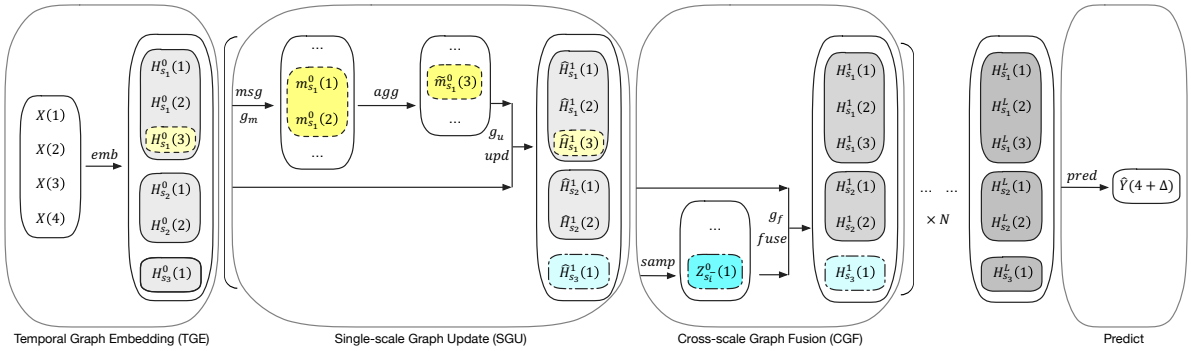


Figure 3: Framework overview of METRO (better viewed in color): a toy example with sequence length 4, i.e., {X(1), X(2), X(3), X(4)}. As illustrated in the TGE unit, three scales with stride 1 are adopted with receptive field of size 2, 3 ,4 respectively. The single-scale update process for time step 3 in scale 1 is marked in yellow and cross-scale graph fusion process for time step 1 in scale 3 is marked in blue. $\times N$ denotes that SGU and CGF can be combined and stacked multiple times, while only one layer is illustrated in the figure.

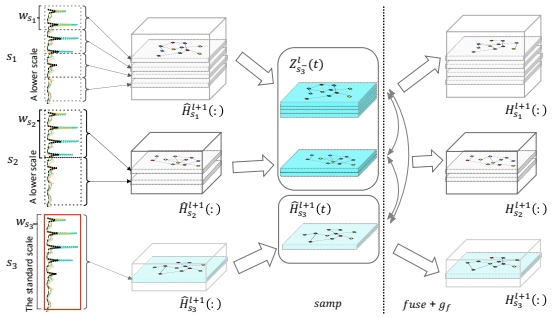


Figure 4: Examples of cross-scale fusion. Scale 3 is chosen as the standard scale.

where $\mathbf{H}^{l+1}_{s_1:s_{i-1}}(:)$ denotes the cross-scale fused embedding of the sampled lower scale features and $\mathbf{H}^{l+1}_{s_i}(t)$ denotes that of the standard high scale at time step t. $fuse$ can be implemented such as GCNs. The choices of g_f can be many, similar to what was discussed in previous sections. There can be multiple CGF kernels under different standard high scales to facilitate the interaction among lower scales where their various receptive fields may not be covered by only one kernel.

For efficiency purpose, in the representative implementation of METRO, i.e., METRO-attn, we again adopt MultiHeadAttn for $fuse$ and the attention matrix with a cut-off threshold r_2 is taken as the adjacency matrix. The cross-scale graph fusion process can be represented as:

$$\begin{aligned}&\mathbf{H}^{l+1}_{s_1:s_{i-1}}(:), \mathbf{H}^{l+1}_{s_i}(t) = MultiHeadAttn(\mathbf{Q}'^{l}(t)\\ &\mathbf{K}'^{l}(t), \mathbf{V}'^{l}(t), r_2),\\ &\mathbf{Q}'^{l}(t) = \mathbf{K}'^{l}(t) = \mathbf{V}'^{l}(t) = [\mathbf{Z}^{l}_{s^-}(t), \hat{\mathbf{H}}^{l+1}(t)].\end{aligned} \tag{10}$$

3.4 Predictor and Training Strategy

SGU unit and CGF kernel are combined and stacked as several layers. Once node embeddings of the final layer obtained, final prediction can be achieved through a predictor

$$\hat{Y}(t+\Delta) = pre(\mathbf{H}_{s_1:s_p}(:)), \tag{11}$$

where $\mathbf{H}_{s_1:s_p}(:)$ denotes the node embeddings of the final layer of the stacked network. pre can be implemented as sum of the last time steps or through learnable functions e.g., MLPs. In our implementations, we concatenate the time steps of scales s_1 to s_{p-1} w.r.t. the time period corresponding to the last step of the largest scale s_p. We then feed it to a reshaping function to the get prediction values. As a common practice [21, 33, 42, 46], we implement the reshaping function as an one-layer MLP for efficiency purpose.

MSE loss is implemented to evaluate the predicted values and the overall training strategy is described in Algorithm 1. The overall data flow is illustrated in Figure 3.

3.5 Instantiate METRO

METRO as a generic framework can have different choices of implementations at each of its key components, which results in a large number of possible instances. In Table 1 we list some of the possible implementations and illustrate how previous GNN-based models for MTS forecasting can be cast as specific instances of METRO [1]. For example, MTGNN [42] uses the graph learning function $A = ReLU(tanh(\alpha(M_1M_2^T - M_2M_1^T)))$, where $M_i = tanh(\alpha E_i W_i)$ and GCN+1dConv as graph update function is a specific case of SGU when the message passing between temporal graphs is missing. The representative implementation, i.e., the one used to present main comparison results with baseline models in Table 3, is METRO-attn.

4 EXPERIMENTS AND RESULTS

4.1 Datasets and Metrics

We conduct experiments on four benchmark datasets[2].

- **Solar Energy:** The raw dataset of Solar Energy was collected by the National Renewable Energy Laboratory (NREL) [3] as synthetic solar photovoltaic (PV) power plant data points for the United States, the year 2006. NREL's Solar Power Data was generated every 5 minutes. We use the pre-processed

[1] Technically, TPA-LSTM [33] is not claimed to be a GNN-based MTS forecasting approach. We here include it because the attention mechanism used in TPA-LSTM can be viewed as an introduction of connection matrix of graphs.
[2] https://github.com/laiguokun/multivariate-time-series-data
[3] https://www.nrel.gov/grid/solar-power-data.html

Table 1: Previous GNN-based MTS forecasting models interpreted within METRO

Model	Temporal Graph Embedding	Single-scale Graph Update					Cross-scale Graph Fusion			Predict
	emb.	msg.	g_m	agg.	upd.	g_u	samp.	fuse	g_f	
MTGNN [42]	concatenation	-	-	-	GCN+1dConv	$A = ReLU(tanh(\alpha(M_1M_2^T - M_2M_1^T)))$, $M_i = tanh(\alpha E_i W_i)$	-	concatenation	-	conv
TEGNN [46]	concatenation	-	-	-	GCN+1dConv	Transfer entropy	-	concatenation	-	MLP
ASTGCN [11]	concatenation	-	-	-	GCN+1dConv	$A = SoftMax(tanh(\mathcal{X}W_1)tanh(W_2^T\mathcal{X}^T))$	-	sum	-	linear trans.
TPA-LSTM [33]	LSTM	-	-	-	LSTM	$\alpha_i = sigmoid(Conv(M)^T WM)$, $M = LSTM(\mathcal{X})$	-	-	-	last
METRO-linear	concat. time steps +linear transform	MultiHeadAttn	attn	sum-all	MultiHeadAttn	attn	slicing	MultiHeadAttn	attn	MLP
METRO-prev	1dConv	MultiHeadAttn	attn	sum-prev	MultiHeadAttn	attn	slicing	MultiHeadAttn	attn	MLP
METRO-GRU	GRU	GRU	-	last	MultiHeadAttn	attn	slicing	MultiHeadAttn	attn	MLP
METRO-GCN	1dConv	MultiHeadAttn	attn	sum-all	GCN	attn	slicing	MultiHeadAttn	attn	MLP
METRO-cat	1dConv	MultiHeadAttn	attn	sum-all	MultiHeadAttn	attn	last	concatenation	-	MLP
METRO-sum	1dConv	MultiHeadAttn	attn	sum-all	MultiHeadAttn	attn	last	sum	-	MLP
METRO-attn	1dConv	MultiHeadAttn	attn	sum-all	MultiHeadAttn	attn	slicing	MultiHeadAttn	attn	MLP

data in the released version, representing the 10-minute variation of 137 PV plants.

- **Traffic:** The Traffic dataset was generated by the California Department of Transportation [4] and collected for the year 2015-2016, describing hourly road occupancy rates measured by 862 sensors on San Francisco Bay area freeways.
- **Electricity:** The raw dataset of Electricity is from the UCI Machine Learning Repository [5], which contains electricity consumption of 370 clients every 15 minutes from 2011 to 2014. We use the pre-processed dataset from [21], which reflects the hourly consumption of 321 clients from 2012 to 2014.
- **Exchange Rate**: First proposed in [21], the Exchange Rate dataset collects the daily exchange rates of 8 countries of the years from 1990 to 2016. The 8 countries are Australia, British, Canada, Switzerland, China, Japan, New Zealand, and Singapore.

Table 2: Statistics of dataset.

Dataset	# samples	# variables	Sample rate	Input length
traffic	17544	862	1 hour	168
solar-energy	52560	137	10 minutes	168
electricity	26304	321	1 hour	168
exchange-rate	7588	8	1 day	168

The statistic of datasets is summarized in Table 2. Input length 168 is common in many MTS forecasting works.

As a common practice, we evaluate the model using two metrics, Root Relative Squared Error (RSE) and Empirical Correlation Coefficient (CORR), which are computed as:

$$RSE = \frac{\sqrt{\sum_{t=t_0}^{t_1}\sum_{i=1}^{m}(y_{t,i} - \hat{y}_{t,i})^2}}{\sqrt{\sum_{t=t_0}^{t_1}\sum_{i=1}^{m}(y_{t,i} - \overline{y_{t_0:t_1,1:m}})^2}}, \tag{12}$$

$$CORR = \frac{1}{m}\sum_{i=1}^{m}\frac{\sum_{t=t_0}^{t_1}(y_{t,i} - \overline{y_{t_0:t_1,i}})(\hat{y}_{t,i} - \overline{\hat{y}_{t_0:t_1,i}})}{\sqrt{\sum_{t=t_0}^{t_1}(y_{t,i} - \overline{y_{t_0:t_1,i}})^2 \sum_{t=t_0}^{t_1}(\hat{y}_{t,i} - \overline{\hat{y}_{t_0:t_1,i}})^2}}, \tag{13}$$

where $\hat{y}$ is the prediction output, y is the ground-truth value, $t \in [t_0, t_1]$ is the time instance in test set, $\overline{y}$ is the mean of set y.

[4]https://pems.dot.ca.gov/

[5]https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

4.2 Baselines

We compared the proposed model with the following methods:

- AR: A standard autoregression model.
- LRidge: The most popular model for MTS forecasting, which is based on Vector Autoregression (VAR) model with L2-regularization.
- VAR-MLP: A hybrid model that combines multi-layer perception (MLP) and VAR [51].
- GP: Gaussian process model for time series forecasting [7, 28].
- LSTNet-Skip[6]: LSTNet [21] with recurrent-skip layer, where LSTNet is a deep neural network that combines convolutional neural networks and recurrent neural networks.
- TPA-LSTM[7]: An attention-based LSTM model [33].
- MTGNN[8]: A graph-based convolutional model [42].
- Informer[9]: A sparse-attention-based model [52].
- HI: A statistical model that use historical data directly as output. [4].

4.3 Implementation Details

All experiments are implemented on an Intel (R) Xeon(R) Gold 6278C CPU @ 2.60GHz and two Tesla T4 GPUs.

4.3.1 Module Architecture. For TGE unit, the channel size is set as 16. For all MultiHeadAttn parts of SGU unit and CGF kernel, the number of layers is set as 1, with queries, keys, and values of dimension 3, head number equals 3, and inner-layer of dimension 8. For a detailed description of the MultiHeadAttn-related hyper-parameters, please refer to [39]. As for the predictor, embedding dimension of the MLP is optimized by the hyper-parameter optimization framework Optuna [10] within the range *[16, 64]*, step size 16.

4.3.2 Hyper-parameter Setting. For each dataset, we split it into 6:2:2 for training, validation, and test. The batch size is chosen by Optuna in range *[8,64]* with step size 8. The optimizer and learning rate are also chosen by Optuna, searching from *Adam [19], stochastic gradient descent(SGD) [27], RMSprop [13]* and within the range *[1e-5, 1e-1]*. We apply a scheduler to control the decay of learning rate, whose step size is chosen from {5, 15, 20} and decay rate chosen

[6]https://github.com/laiguokun/LSTNet

[7]https://github.com/gantheory/TPA-LSTM

[8]https://github.com/nnzhan/MTGNN

[9]https://github.com/zhouhaoyi/Informer2020

[10]https://github.com/optuna/optuna

Table 3: Summary of comparison results w.r.t. RSE and CORR. The best results are highlighted in bold and the second best results are underlined.

Dataset		Solar Energy				Traffic				Electricity				Exchange Rate			
		Horizon				Horizon				Horizon				Horizon			
Method	Metric	3	6	12	24	3	6	12	24	3	6	12	24	3	6	12	24
HI	RSE	0.2749	0.4488	0.7436	1.1937	1.0492	1.3492	1.5224	0.6400	0.4522	0.7331	1.0024	0.1058	**0.0170**	**0.0237**	0.0388	0.0462
	CORR	0.9629	0.8977	0.7132	0.2490	0.4021	-0.0390	-0.3774	0.7401	0.5930	0.1150	-0.2979	0.8670	0.9737	0.9652	0.9463	0.9285
AR	RSE	0.2549	0.3849	0.6127	0.8585	0.6028	0.6257	0.6302	0.6400	0.1016	0.1045	0.1083	0.1164	0.0230	0.0295	0.0388	0.0462
	CORR	0.9615	0.9239	0.8086	0.5492	0.7763	0.7543	0.7519	0.7401	0.8781	0.8610	0.8506	0.8592	0.9702	0.9581	0.9463	0.9285
LRidge	RSE	0.2161	0.3084	0.4792	0.7376	0.5981	0.6065	0.6151	0.6221	0.1530	0.1463	0.2354	0.1292	0.0202	0.0288	0.0468	0.0694
	CORR	0.9771	0.9533	0.8773	0.6788	0.7968	0.8000	0.7856	0.7804	0.8868	0.8583	0.7943	0.8733	0.9739	0.9691	0.9531	0.9216
VAR-MLP	RSE	0.2144	0.2880	0.4293	0.6921	0.5686	0.6718	0.6075	0.6234	0.1485	0.1784	0.1607	0.1306	0.0270	0.0437	0.0450	0.0604
	CORR	0.9742	0.9591	0.9050	0.7097	0.8189	0.7654	0.7909	0.7852	0.8694	0.8306	0.8190	0.8596	0.8548	0.8654	0.8206	0.7644
GP	RSE	0.2409	0.3317	0.5284	0.7986	0.6175	0.6343	0.6735	0.6821	0.1500	0.2014	0.1732	0.1303	0.0218	0.0281	0.0399	0.0597
	CORR	0.9706	0.9366	0.8454	0.5918	0.7772	0.7665	0.7361	0.7241	0.8615	0.8302	0.8338	0.8756	0.8831	0.8170	0.8470	0.8224
Informer	RSE	0.2387	0.3188	0.3654	0.4950	0.5481	0.5597	0.6613	0.5590	0.1475	0.1481	0.1691	0.1489	0.1590	0.1438	0.1788	0.1637
	CORR	0.9763	0.9587	0.9417	0.8834	0.8360	0.8270	0.7470	0.8289	0.8777	0.8760	0.8659	0.8692	0.9176	0.9062	0.8478	0.8223
LSTNet-Skip	RSE	0.1978	0.2607	0.3329	0.4654	0.4880	0.4860	0.5047	0.5214	0.0925	0.1018	0.1090	0.1100	0.0231	0.0289	0.0367	0.0500
	CORR	0.9750	0.9609	0.9421	0.8850	0.8684	0.8687	0.8541	0.8523	0.9242	0.9088	0.9066	0.9072	0.9725	0.9641	0.9505	0.9285
TPA-LSTM	RSE	0.1833	0.2365	0.3347	0.4471	0.4516	0.4693	0.4793	0.4810	0.0862	0.0931	0.1005	0.1042	0.0172	0.0243	0.0361	0.0454
	CORR	0.9791	0.9662	0.9403	0.9065	0.8777	0.8708	0.8669	0.8626	0.9436	0.9302	0.9177	0.9086	**0.9794**	0.9683	0.9496	0.9323
MTGNN	RSE	0.1817	0.2405	0.3133	0.4469	0.4244	0.4784	0.4538	0.4670	0.0751	0.0927	0.0968	0.0944	0.0196	0.0265	0.0361	0.0476
	CORR	0.9799	0.9696	0.9469	0.8996	0.8896	0.8656	0.8771	**0.8812**	0.9457	0.9240	0.9259	0.9242	0.9768	0.9660	0.9502	0.9309
METRO-attn	RSE	**0.1760**	**0.2353**	**0.3092**	**0.4171**	**0.4087**	**0.4345**	**0.4380**	**0.4406**	**0.0717**	**0.0802**	**0.0909**	**0.0912**	0.0182	0.0242	**0.0339**	**0.0431**
	CORR	**0.9854**	**0.9726**	**0.9531**	**0.9090**	**0.8983**	**0.8853**	**0.8812**	0.8810	**0.9501**	**0.9409**	**0.9327**	**0.9301**	0.9790	**0.9709**	**0.9581**	**0.9407**

between [0.6, 1.0] by Optuna. The weight decay rate is set as 1e-5. The total number of epochs is set as 100, through which we save the best model based on CORR/RSE of the validation set and reload it for the evaluation of the test set. Unless explicitly stated, all reported results are on the test set. Dropout 0.1 is applied to the outputs of the MultiHeadAttn and MLP layers. Threshold r in Equation (6) and (10) is set as $r_1 = r_2 = 80\%$, 80%, 180% and 60% of the mean of attention values for Solar Energy, Traffic, Electricity and Exchange Rate dataset respectively. If not specified or tuned with Optuna, all parameters are the same on four datasets. We report the mean, or mean and standard deviation in ten runs with different random seeds.

4.4 Main Results

Table 3 compares the performance of our proposed network and baseline models for 3, 6, 12, and 24 steps ahead prediction on the four datasets. The best results are highlighted in bold and the second-best results are underlined.

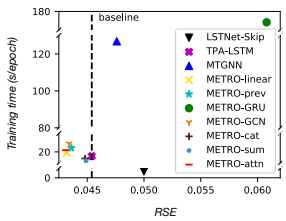
We observe that the implementation of METRO, METRO-attn, achieves state-of-the-art results in almost all cases. One interesting observation is that METRO-attn gets more significant improvement on longer-term predictions (horizon=12, 24). In other words, compared with baselines, the model is more robust when the prediction horizon becomes even larger. We credit this to the fact that METRO models dynamic graph structure and the cross-scale information fusion mechanism can enhance the pattern related to the target horizon, thus, the prediction accuracy deterioration from small to large horizons is not drastic.

Another observation is that HI achieves the best results on small horizons, i.e., horizon = 3, the Exchange Rate dataset. This is resulted from that the Exchange Rate dataset has an almost linear temporal feature, which is easier to predict for naive models but can be overfitted by more sophisticated models such as MTGNN and METRO.

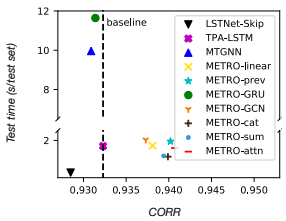
There is an overall trend that the prediction accuracy drops as the prediction horizon increases. This follows our intuition that farther future is harder to predict. However, for HI and Informer, we can observe a rebound at the prediction horizon of 24, Electricity, and Traffic datasets. We argue that this is because of the one-day (24 hours) periodic pattern of the datasets and the two models both having preference or mechanism to deal with periodic data, i.e., periodic phase similarity in historical data can greatly enhance the performance of HI [4] and there is a time-stamp feature embedding mechanism in Informer.

4.5 Detailed Study of METRO

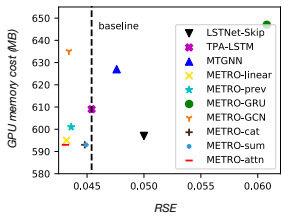
4.5.1 Choice of Modules. As METRO is designed as a loosely coupled framework, there can be many instances with different choices of implementations for each module. In this section a comprehensive analysis and comparison on different instances of METRO are performed. In the following discussions, we take the representative implementation METRO-attn as the standard and compare other variants with it. Results of some variants can be regarded as the ablation study of the framework METRO to verify the effectiveness of some key components. We are mostly interested in the trade-off between the prediction accuracy (RSE and CORR) and the model efficiency, measured by training time, prediction time, GPU memory cost and model size. The variants we experiment with are reported in Table 1 and the results are reported in Figure 5, where the dashed lines denote the best performance of RSE and CORR among all baselines. The performance of three most recent works, i.e. LSTNet-Skip [21], TPA-LSTM [33] and MTGNN [42], are also included in Figure 5. All variants run 100 epochs with the same



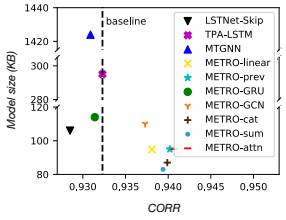
(a) Trade off between RSE and training time (time is measured by average sec per epoch)



(b) Trade off between CORR and test time (time measured by average sec per test set)



(c) Trade off between RSE and GPU memory cost (sampled every 10 seconds of the training process and then averaged over the total number of samples)



(d) Trade off between CORR and the saved model's size

Figure 5: Trade off between prediction accuracy and training time, test time, GPU memory cost and model size on Exchange Rate dataset with horizon of 24.

hyper-parameter setting discussed before. We observe a similar trend on the four datasets and report the results on the Exchange Rate dataset at horizon 24 only. The same reason applies to results reported in subsequent sections. Though convolution might not be implemented in some variants, we retain the term "stride" when describing the movement of receptive fields.

The very first observation is that most of our models significantly outperform state-of-the-art baselines in both effectiveness and efficiency. Besides, some other observations are as follows.

TGE. METRO-linear obtains node embeddings by linearly transforming the concatenation of input time series. Each concatenation is operated along the time axis of a scale-window-sized subsequence of the input series, which then strides to the next subsequence. The obtained tensor is then linearly transformed to a certain embedding dimension, which is set as 16 in our experiments. While METRO-linear is a little bit faster in training, METRO-attn performs better on RSE and CORR, indicating the importance of comprehensively modeling temporal information. It is also worth mentioning that METRO-linear is slightly smaller than METRO-attn in terms of parameters, but costs more training memory. This is because the slicing operation needs extra containers to receive the time period slices.

SGU. METRO-prev excludes the contextual update strategy and only uses information from previous steps for aggregation and update. We implement this variant by introducing a mask when performing MultiHeadAttn, where the attention values of future steps are masked as zeros. The effectiveness of the contextual update strategy is evidenced by faster and more accurate prediction of METRO-attn. The masks used in METRO-prev that prevent the model from seeing feature values also increase training memory cost. However, as they are non-parameterized, the model size of METRO-prev is the same as METRO-attn. For METRO-GRU, we feed a 1-layer GRU with scale-window-sized subsequences of the input and then take the last output hidden state as the aggregated node embeddings. METRO-GRU exhibits the slowest speed for that the embeddings are updated sequentially in GRU layers. The GRU layer also introduces much more parameters, which slows the training process and leads to poor performance under limited epochs. For the same reason, it needs more memory storage and results in larger model size. METRO-GCN replaced the MultiHeadAttn function with a 1-layer GCN, in which the adjacency matrix is computed through self-attention [39]. It can be observed that METRO-GCN slightly hurts the prediction accuracy but has a significant increase in time cost and storage-related metrics, which confirms the efficiency of MultiHeadAttn.

Table 4: Performance of METRO-attn with different scales for prediction on Exchange Rate and Electricity datasets with horizon of 3 and 24.

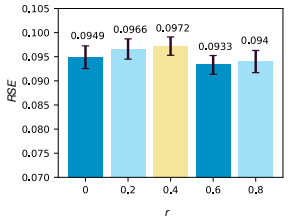
Dataset	Electricity				Exchange Rate			
Horizon	3		24		3		24	
Scales	RSE	CORR	RSE	CORR	RSE	CORR	RSE	CORR
12	0.1906	0.8358	0.1122	0.8753	0.0206	0.9784	0.0435	0.9374
24	0.1165	0.9158	0.1026	0.8847	0.0186	0.9788	0.0434	0.9384
12,24	0.0966	0.9225	0.1030	0.8857	0.0214	0.9757	0.0442	0.9367
12,24,48	0.0808	0.9377	0.0999	0.9053	0.0183	0.9789	0.0432	0.9390
12,48	0.0945	0.9251	0.1019	0.9029	0.0243	0.9748	0.0494	0.9207
24,48,96	0.0717	0.9501	0.0912	0.9301	0.0182	0.9790	0.0431	0.9407
12,24,48,96	0.0848	0.9451	0.1046	0.9240	0.0325	0.9471	0.0457	0.9369
12,24,96	0.0729	0.9486	0.0919	0.9271	0.0276	0.9741	0.0427	0.9409

CGF. METRO-cat and METRO-sum are the variants of our framework where cross-scale fusion is removed. The two variants can indeed speed up the training process but significantly underperform METRO-attn on prediction accuracy, which verifies the benefits of cross-scale information propagation in modeling node embeddings. The METRO equipped with either simplified fusion component has only slightly less training memory cost than METRO-attn since the overall storage cost of CGF kernels is negligibly small compared with data-related storage. However, CGF does take an important portion of storage if viewing the model itself, given that METRO-cat and METRO-sum both have notably smaller model sizes than METRO-attn.

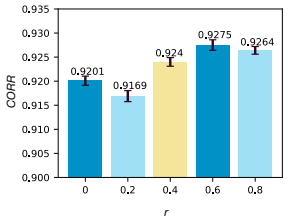
4.5.2 Multi-scale Analysis. To analyze the effect of multiple scales, we further evaluate the METRO-attn model with different scales on Exchange Rate and Electricity datasets, for short- (horizon=3) and long-term prediction (horizon=24). The stride size is fixed as 12. Table 4 depicts the results.

The first insight is that the choices of scales have an obvious impact on the performance on the Electricity dataset, as shown by the fact that for horizon 3, RSE reaches up to 0.1906 with scale (12) and can be lowered to 0.0717 with scale (24,48,96). Moreover, for the Electricity dataset, two scales are always better than one scale and three get the best results. This is because the Electricity dataset has more explicit periodical patterns (daily, weekly) that are easier to be extracted by the model with multiple scales. Another interesting observation is that the model gives better performance

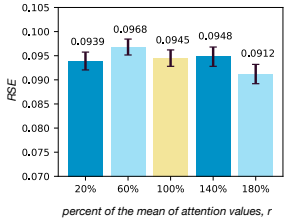
when larger scales are introduced, and the trend is more obvious on horizon 24, e.g., RSE and CORR get better from scale setting (12)→(12,24)→(12,24,48) on horizon 24 of the Electricity dataset. This is in line with the intuition that considering the larger scales enable the model to capture long-run/global patterns, which are vital for long-term prediction.



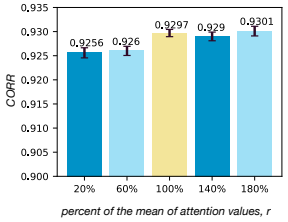
(a) Effect of fixed cut-off threshold r on RSE (Electricity)



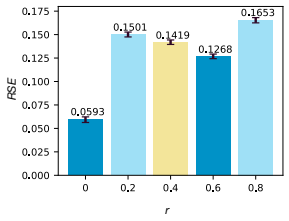
(b) Effect of fixed cut-off threshold r on CORR (Electricity)



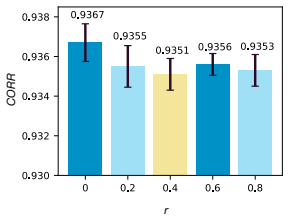
(c) Effect of adapted cut-off threshold r on RSE (Electricity)



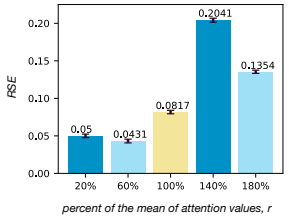
(d) Effect of adapted cut-off threshold r on CORR (Electricity)



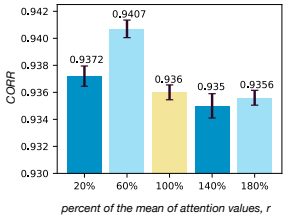
(e) Effect of fixed cut-off threshold r on RSE (Exchange Rate)



(f) Effect of fixed cut-off threshold r on CORR (Exchange Rate)



(g) Effect of adapted cut-off threshold r on RSE (Exchange Rate)

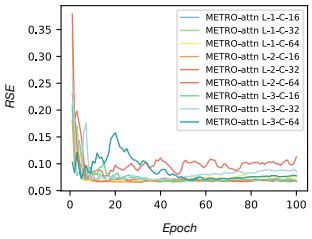


(h) Effect of adapted cut-off threshold r on CORR (Exchange Rate)

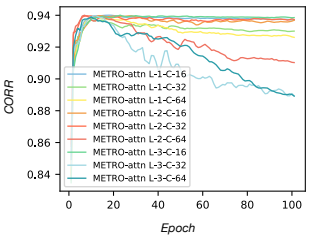
Figure 6: Effect of adapted cut-off threshold r on Exchange Rate dataset with horizon of 24.

4.5.3 Effect of threshold r. Attention matrix with thresholds plays a key role in the proposed model architecture. In this section, we evaluate the effect of the thresholds on Electricity and Exchange Rate datasets, which have 8 and 321 variables respectively. We are mostly interested in how the values of the thresholds contribute to forecasting performance. Therefore, for simplicity, we first let $r_1 = r_2 = r$ and conduct experiments on 1) manually-set values of r and 2) self-adapted values of r, which are calculated as n% of the mean value of the attention matrix. The results are shown in Figure 6.

A graph whose adjacency matrix adopts directly from an attention matrix can be seen as a fully connected graph. Removing connections with attention values lower than a fixed threshold only considers the absolute value of attention while ignores its overall distribution. Accordingly, it inevitably leads to an inferior performance, i.e., the RSE and CORR in Figure 6 (a) (b) (e) (f) are significantly worse than most cases in Figure 6 (c) (d) (g) (h), where the values of r are adapted. Another interpretation for this phenomenon can be from the perspective of training: a fixed small value of r might not be helpful enough in propagating useful information from potential neighbors, while a fixed large value of r might introduce noise information. Thus, training is harder under such an inflexible setting than using an adapted threshold. Form Figure 6 (c) (d) (g) (h), we can tell that a relatively medium-value of r benefits prediction the most. In conclusion, r has a great influence on prediction accuracy and its value needs to be carefully decided or the contribution could be negative. Although it is an empirical parameter, 100% of the mean value is a simple and generally good choice.



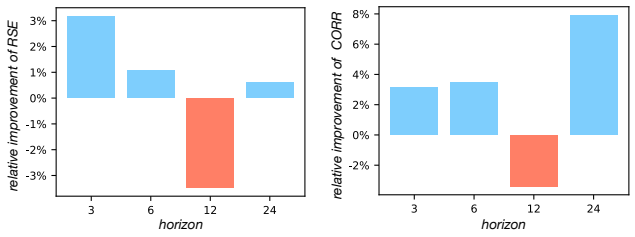
(a) RSE over training epochs on the validation set of Exchange Rate dataset, horizon of 24.



(b) CORR over training epochs on the validation set of Exchange Rate dataset, horizon of 24.

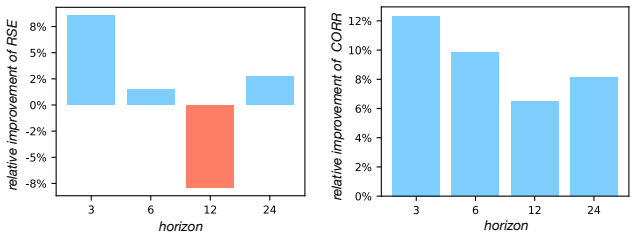
Figure 7: Effect of model layers and embedding dimensions. RSE and CORR are measured for prediction on Exchange Rate dataset at horizon 24.

4.5.4 Effect of Network Depth. We further investigate the depth of the proposed network. The results are demonstrated in Figure 7, which are obtained on the validation set at the process of network training on the Exchange Rate dataset at horizon 24. The notation in the form of METRO-attn L-x-C-y denotes the network is of x ∈ {1,2,3} layer deep with nodes of embedding dimension y ∈ {16,32,64}. Other hyper-parameters setting is in accordance with those stated at the beginning of the section. We can conclude from Figure 7 that, generally, a deeper network and larger embedding dimensions enable higher CORR and lower RSE. However, it can also be observed that with more epochs, RSE increases and CORR decrease. This phenomenon is more significant in the case of deeper networks and especially, with larger embedding dimensions, which indicates that models with high complexity can over-fit more easily. Many strategies such as early stop can be adopted to remedy this problem but it is beyond the scope of this paper.



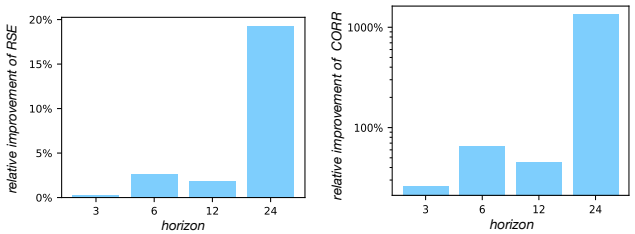
(a) Relative improvement of METRO w.r.t. RSE (A6)

(b) Relative improvement of METRO w.r.t. CORR (A6)



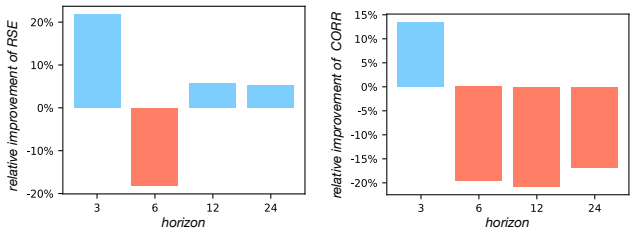
(c) Relative improvement of METRO w.r.t. RSE (B1)

(d) Relative improvement of METRO w.r.t. CORR (B1)



(e) Relative improvement of METRO w.r.t. RSE (C1)

(f) Relative improvement of METRO w.r.t. CORR (C1)



(g) Relative improvement of METRO w.r.t. RSE (C4)

(h) Relative improvement of METRO w.r.t. CORR (C4)

Figure 8: Relative improvement of METRO over MTGNN w.r.t. MSE and CORR on the online test.

4.6 Deployment and Online Experiments

Online experiments were conducted on the cloud-native time series analytics platform of Huawei Cloud to verify the superior performance of METRO. Huawei Cloud provides cloud and AI services to hundreds of thousands of individual and industry customers, including but not limited to storage, computing, and network resources. Users of Huawei Cloud generate hundreds of thousands of resource/solution queries every day. To guarantee user experience and resource security, the time series analysis system watches over the state of certain services performing tasks such as anomaly detection or predictive maintenance. To deploy METRO, we utilize one Intel (R) Xeon(R) Gold 6278C CPU @ 2.60GHz and one Tesla T4 GPU card. METRO was implemented as METRO-attn.

Table 5: Statistics of deployment dataset.

Dataset	# samples	# variables	Sample rate	Input length
A6	3168	2	1 hour	168
B1	3168	6	1 hour	168
C1	3168	3	1 hour	168
C4	3168	1	1 hour	168

The dataset for training recorded the hourly virtual CPU (VCPU) requirements of the users on Huawei Cloud. The training data was collected from July 28. 2020 to December 7, 2020. We evaluated METRO on four sub-datasets named A6, B1, C1, and C4, respectively, each corresponds to the VCPU queries in a specific pod. Different dimensions in a dataset correspond to different VCPU types. A detailed description of the datasets can be found in Table 5. We tested METRO with the input length equal to 168 and prediction horizon [3, 6, 12, 24], for a testing duration of about 1 month. State-of-the-art MTGNN was chosen as a comparison method for its superiority.

Figure 8 shows the relative improvement of METRO over MTGNN w.r.t. MSE and CORR. We can see that METRO-attn significantly outperforms the baseline model in most cases, bringing up to 20% and 1300% relative improvement w.r.t RSE and CORR on dataset C1. Another interesting observation is that METRO tends to perform much better when the data has more variables. For example, the relative improvement on B1 and C1 is much more significant than on A6 and C4. This further verifies that effectively capturing correlations between variables can enhance prediction accuracy.

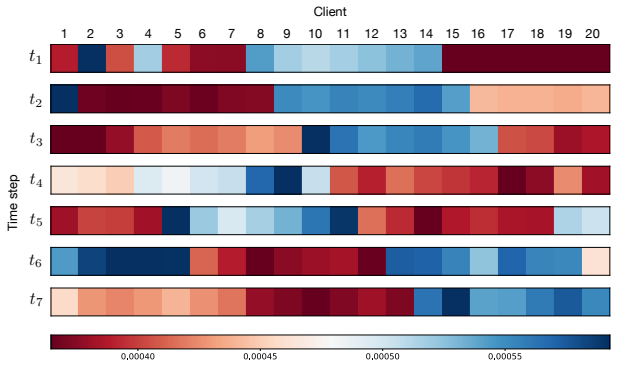
It can also be observed from Figure 8 that there are some negative improvements of MTEOR over MGTNN on C4. This phenomenon is mainly because C4 is a univariate time series dataset. Since a large portion of parameters of METRO-attn are designed for variable interaction, which is not the case for MTGNN, a simple dataset like C4 makes METRO-attn more vulnerable to overfit.

4.7 Visualizing METRO

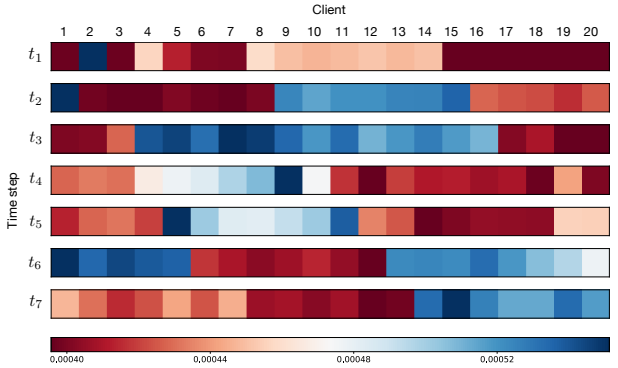
We further evaluate METRO by visualizing model parameters and prediction results in two case studies.

4.7.1 Temporal Graphs. Figure 9 shows the interpretability offered by METRO's temporal graphs using the Electricity dataset, predicting horizon of 12. We plot the attention weights used to model the adjacency matrix of single-scale graphs of one client (#32) over the other 20 (#1 - #20) across two attention heads and seven time steps.

We see that though each attention head operates on its own attention scale, they share a similar emphasis on certain clients. For example, clients #8 - #14 are of higher attention weights to the evaluated client on the first time step for both heads. This observation suggests that some clients are more related to a specific client and such information can be effectively captured by multiple attention heads.



(a) head1



(b) head2

Figure 9: Visualized single-scale attention across 2 heads and 7 time steps w.r.t. client #32 and clients #1 - #20 of the Electricity dataset, horizon of 12. It can be observed that the interdependency between variables (clients) evolves over time.

The proposed temporal graph aims to capture time-evolving variable correlations. For each attention head, we observe that the attention weights evolve over time steps. Taking the first head as an example, the most connected neighbors of the evaluated client transfer from clients #8 - #14 to #9 - #15 and then #10 - #16, on time steps 1 to 3. A similar trend can also be found on head 2. Therefore, we can conclude that time-evolving feature temporal graphs is well expressed by the model.

4.7.2 Model Predictions. Figure 10 displays sample predictions from the validation set of the Electricity dataset, horizon of 12. The two sub-figures (a) and (b) correspond to two different variables; the left column is the prediction from state-of-the-art method MTGNN [42] and the right column is from the proposed METRO-attn. We mark the ground-truth value with dark solid lines and prediction values with light dash lines. It can be observed that both methods can well forecast the general trend, while METRO-attn successfully captures detailed patterns, especially at peaks and valleys, which once more verifies that the improvement of prediction performance is indeed introduced by our proposed approach.

5 COMPLEXITY ANALYSIS

In this section, we analyze the time complexity of key components of METRO by taking the implementation **METRO-attn** as an example. The time complexity can be summarized in Table 6. We here

Table 6: Time complexity of key components of METRO, taking METRO-attn as an example.

Module	Time complexity
Temporal graph embedding unit	$O(NL^2)$
Single-scale graph update unit	$O(NL + N^2L^2)$
Cross-scale graph fusion kernel	$O(L + NM + N^2M^2)$

omit the parameter of batch size. There are some major notations use in the following analysis: D denotes the number of scales; i represents the i^{th}; K_i represents kernel size corresponding to the i^{th} scale; S_i denotes the stride length of the i^{th} scale and C denotes the number of output channel, a.k.a the number of kernels; N denotes the number of variables of the MTS; L denotes the length of the input MTS.

TGE. The time complexity of the temporal graph embedding (TGE) unit in METRO-attn is $O(N\sum_i^D(\frac{L-K_i}{S_i}+1)^2{K_i}^2C)$. Taking D, K_i, S_i and C as constants, the time complexity of the graph temporal graph learning unit becomes $O(NL^2)$.

SGU. In the single-scale graph update (SGU) unit, the time complexity can be calculated as $O(\sum_i^D Nk_i(\frac{L-K_i}{S_i}+1)C^2/h+N^2{k_i}^2(\frac{L-K_i}{S_i}+1)^2C+Nk_i(\frac{L-K_i}{S_i}+1)C^2)$, where k_i is the number of previous steps taken into considered and h denotes the number of heads. In advanced version of SGU-Contextual (SGU-C), where not only previous information but some future information is considered, the time complexity is $O(\sum_i^D N(k_i+v_i)(\frac{L-K_i}{S_i}+1)C^2/h+N^2(k_i+v_i)^2(\frac{L-K_i}{S_i}+1)^2C+N(k_i+v_i)(\frac{L-K_i}{S_i}+1)C^2)$, where v_i is the number of future steps considered. Treating k_i, v_i, h, D, K_i, S_i and C as constants, the time complexity of SGU can be written as $O(NL+N^2L^2)$, which indicates that attention computation and linear transforming of the multi-head outputs are the most time-consuming operations of the SGU unit.

CGF. The time complexity of the cross-scale graph fusion (CGF) kernel has two major parts. One is resulted from the time-step align operation, and the other one is resulted from the attention-based fusion. Taking scale s_i as the standard scale, let M_i denote the total number of sampled time steps from lower scales, mathematically, the time complexity of the CGF kernel can be descried as $O(\sum_1^i\frac{L-K_i}{S_i}+1+\sum_i^D NM_iC^2/h+N^2{M_i}^2C+NM_iC^2)$. Treating D, K_i, S_i and C as constants, above formula can be rewritten as $O(L+NM+N^2M^2)$.

6 RELATED WORK

Known as one of the most widely used models for linear univariate time series prediction, ARIMA [1, 40], which generalizes auto regression (AR), moving average (MA), and autoregressive moving average (ARMA) models, aims to extract autocorrelation between samples. Additionally, the Seasonal ARIMA (SARIMA) [41] model extends the ARIMA by adding a linear combination of seasonal past values and/or forecast errors, and BHT-ARIMA [32] generalizes the classical ARIMA to tensor form and incorporates it into Tucker decomposition to better capture intrinsic correlation among multiple time series. LSVR [38] and TRMF [48] extend the univariate autoregressive model to dynamic multivariate time series. Gaussian

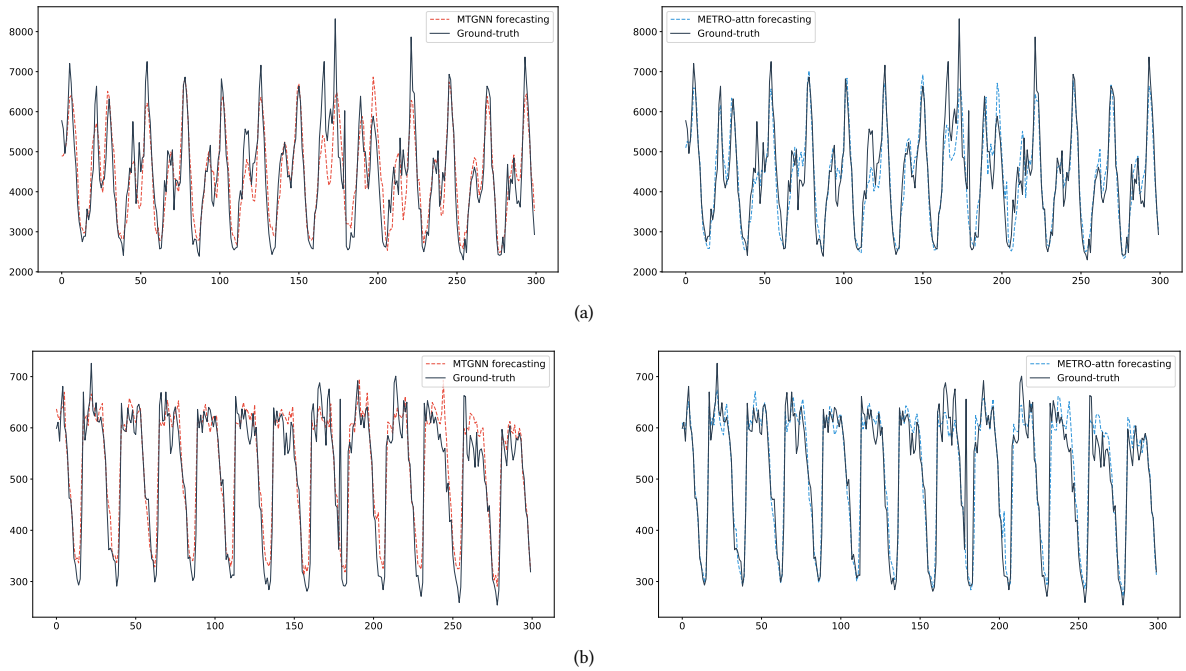


Figure 10: Prediction visualization on the Electricity dataset, horizon of 12. We plot 300 data points of the true time series (black) and the predicted ones (blue) by MTGNN (left column) and (red) by METRO-attn (right column) for two variables of the dataset. We can see that METRO's predictions fit ground-truth better, especially in peaks and valleys of the data.

processes (GP) [28, 35] approach benefits the prediction by modeling the data quantity properties that are inherited from normal distributions.

In spite of the easy accessibility of the above statistic models, they suffer from the strong approximation of linearity to complex real-world problems. Recently, it has been proven that deep-learning models are promising to capture non-linear interdependencies [12, 30]. LSTNet [21], TPA-LSTM [33] and R2N2 [8] utilize RNN to capture long-term temporal patterns. TLASM [45] employs the tensorized LSTM to model the temporal patterns of long-term trend sequences in a multi-task learning setting. The attention mechanism is employed in [5, 15, 18] to model potential relation in time series. [42] globally extracts the latent relations among variables and propagates information through GNN based approach. However, they all fail to deal with dynamic variable correlations and ignore potential relations between different temporal scales.

Rather than the typical setting of predicting values of a limited number of time-steps, i.e. 48 steps or fewer [21, 33, 42], an emerging line of work focuses on the problem of long sequence time-series forecasting (LSTF), where up to 720 steps can be predicted at a time [4, 52]. We here focus on the task of typical time series forecasting for it fits wider real-world application scenarios. To verify the effectiveness of METRO, we also include models proposed for LSTF for comparison and the experimental results verify the superiority of our proposed model.

Temporal/Dynamic graphs have been attracting increasing attention in many fields. In the machine learning community, according to the studied problem, previous works can be categorized into two groups: learning on Discrete Time Dynamic Graphs (DTDG) and learning on Continuous Time Dynamic Graphs (CTDG). Approaches for leaning on DTDG include aggregating graph snapshots and then applying static methods [17, 23, 31], combining snapshots with other learning-based models [43, 49, 50] and incorporating time series models, e.g. RNNs [10, 16]. Only recently have CTDG been addressed [29]. Sequential model based [24, 36, 37] and random walk-based [25, 26] approaches have been proposed to learn and update embeddings of nodes and edges. Some recent works have also study dynamic knowledge graphs [6, 9, 44]. However, the great majority of previous efforts on temporal graphs focus on representation learning that evaluates the tasks of link prediction or classification, which have very distinct challenges with time series prediction and thus cannot be adopted directly.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we propose METRO, a generic multi-scale temporal graph neural network for MTS forecasting. Our approach models MTS as a series of automatically learned dynamic graphs and at multiple temporal scales. Effective propagation strategies have been proposed to obtain the node embeddings where hidden temporal patterns of MTS are encoded. The superiority of the approach is verified by the state-of-the-art results achieved on four offline benchmark datasets. Detailed ablation studies also show the effectiveness of all key components. METRO has been successfully deployed onto the time series analytics platform of Huawei Cloud, with significant performance improvement demonstrated on a one-month production environment test. In the future, we plan to explore the automatic instance selection of the framework w.r.t. to specific tasks.

ACKNOWLEDGMENTS

This work is partially supported by Natural Science Foundation of China (No. 61972069, 61836007 and 61832017).

REFERENCES

[1] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

[2] Peter J Brockwell, Richard A Davis, and Matthew V Calder. 2002. *Introduction to time series and forecasting*. Vol. 2. Springer.

[3] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP'14*.

[4] Yue Cui, Jiandong Xie, and Kai Zheng. 2021. Historical Inertia: A Neglected but Powerful Baseline for Long Sequence Time-series Forecasting. In *CIKM'21*.

[5] Yue Cui, Chen Zhu, Guanyu Ye, Ziwei Wang, and Zheng Kai. 2021. Into the Unobservables: A Multi-range Encoder-decoder Framework for COVID-19 Prediction. In *CIKM'21*.

[6] Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha Talukdar. 2018. Hyte: Hyperplane-based temporally aware knowledge graph embedding. In *Proceedings of the 2018 conference on empirical methods in natural language processing*. 2001–2011.

[7] Roger Frigola, Fredrik Lindsten, Thomas B Schön, and Carl Edward Rasmussen. 2013. Bayesian Inference and Learning in Gaussian Process State-Space Models with Particle MCMC. In *NIPS'13*. 3156–3164.

[8] Hardik Goel, Igor Melnyk, and Arindam Banerjee. 2017. R2n2: Residual recurrent neural networks for multivariate time series forecasting. *arXiv preprint arXiv:1709.03159* (2017).

[9] Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. 2020. Diachronic embedding for temporal knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3988–3995.

[10] İsmail Güneş, Şule Gündüz-Öğüdücü, and Zehra Çataltepe. 2016. Link prediction using time series of neighborhood-based node similarity scores. *Data Mining and Knowledge Discovery* 30, 1 (2016), 147–180.

[11] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. (2019), 922–929.

[12] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. 2020. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting* (2020).

[13] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 14, 8 (2012), 2.

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

[15] Siteng Huang, Donglin Wang, Xuehan Wu, and Ao Tang. 2019. DSANet: Dual Self-Attention Network for Multivariate Time Series Forecasting. In *CIKM'19*. 2129–2132.

[16] Zan Huang and Dennis KJ Lin. 2009. The time-series link prediction problem with applications in communication surveillance. *INFORMS Journal on Computing* 21, 2 (2009), 286–303.

[17] Nahla Mohamed Ahmed Ibrahim and Ling Chen. 2015. Link prediction in dynamic social networks by integrating different types of information. *Applied Intelligence* 42, 4 (2015), 738–750.

[18] Dino Ienco and Roberto Interdonato. 2020. Deep Multivariate Time Series Embedding Clustering via Attentive-Gated Autoencoder. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 318–329.

[19] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR'15*. 1395–1402.

[20] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR'17*.

[21] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. 2018. Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. In *SIGIR'18*. 95–104.

[22] Maosen Li, Siheng Chen, Yangheng Zhao, Ya Zhang, Yanfeng Wang, and Qi Tian. 2020. Dynamic Multiscale Graph Neural Networks for 3D Skeleton Based Human Motion Prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 214–223.

[23] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58, 7 (2007), 1019–1031.

[24] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. 2020. Streaming graph neural networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 719–728.

[25] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*. 969–976.

[26] Giang H Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Dynamic network embeddings: From random walks to temporal random walks. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 1085–1092.

[27] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951), 400–407.

[28] Stephen Roberts, Michael Osborne, Mark Ebden, Steven Reece, Neale Gibson, and Suzanne Aigrain. 2013. Gaussian Processes for time-series modelling. *Philosophical Transactions Mathematical Physical & Engineering Sciences* (2013).

[29] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv preprint arXiv:2006.10637* (2020).

[30] Rajat Sen, Hsiang-Fu Yu, and Inderjit S Dhillon. 2019. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. In *NIPS'19*. 4837–4846.

[31] Umang Sharan and Jennifer Neville. 2008. Temporal-relational classifiers for prediction in evolving domains. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 540–549.

[32] Qiquan Shi, Jiaming Yin, Jiajun Cai, Andrzej Cichocki, Tatsuya Yokota, Lei Chen, Mingxuan Yuan, and Jia Zeng. 2020. Block Hankel Tensor ARIMA for Multiple Short Time Series Forecasting.. In *AAAI'20*. 5758–5766.

[33] Shun-Yao Shih, Fan-Keng Sun, and Hung-yi Lee. 2019. Temporal pattern attention for multivariate time series forecasting. In *Machine Learning*, Vol. 108. 1421–1441.

[34] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Charu C Aggarwal, Prasenjit Mitra, and Suhang Wang. 2020. Joint Modeling of Local and Global Temporal Dynamics for Multivariate Time Series Forecasting with Missing Values.. In *AAAI'20*. 5956–5963.

[35] Anh Tong and Jaesik Choi. 2019. Discovering latent covariance structures for multiple time series. In *ICML'19*. PMLR, 6285–6294.

[36] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. 2017. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In *international conference on machine learning*. PMLR, 3462–3471.

[37] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*.

[38] Vladimir Vapnik, Steven E Golowich, and Alex J Smola. 1997. Support vector method for function approximation, regression estimation and signal processing. In *NIPS'97*. 281–287.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *NIPS'17*.

[40] Andrew A Weiss. 1984. Systematic sampling and temporal aggregation in time series models. *Journal of Econometrics* 26, 3 (1984), 271–281.

[41] Billy M Williams and Lester A Hoel. 2003. Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: Theoretical basis and empirical results. *Journal of transportation engineering* 129, 6 (2003), 664–672.

[42] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. 2020. Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks. In *KDD'20*.

[43] Yu Xin, Zhi-Qiang Xie, and Jing Yang. 2016. An adaptive random walk sampling method on dynamic community detection. *Expert Systems with Applications* 58 (2016), 10–19.

[44] Chenjin Xu, Mojtaba Nayyeri, Fouad Alkhoury, Hamed Yazdi, and Jens Lehmann. 2020. Temporal knowledge graph completion based on time series gaussian embedding. In *International Semantic Web Conference*. Springer, 654–671.

[45] Dongkuan Xu, Wei Cheng, Bo Zong, Dongjin Song, Jingchao Ni, Wenchao Yu, Yanchi Liu, Haifeng Chen, and Xiang Zhang. 2020. Tensorized LSTM with Adaptive Shared Memory for Learning Trends in Multivariate Time Series.. In *AAAI'20*. 1395–1402.

[46] Haoyan Xu, Yida Huang, Ziheng Duan, Jie Feng, and Pengyu Song. 2020. Multivariate Time Series Forecasting Based on Causal Inference with Transfer Entropy and Graph Neural Network. *arXiv preprint arXiv:2005.01185* (2020).

[47] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Li. 2019. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *AAAI'19*, Vol. 33. 5668–5675.

[48] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. 2016. Temporal regularized matrix factorization for high-dimensional time series prediction. In *NIPS'16*. 847–855.

[49] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Haifeng Chen, and Wei Wang. 2017. Link Prediction with Spatial and Temporal Consistency in Dynamic Networks.. In *IJCAI*. 3343–3349.

[50] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. 2018. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2672–2681.

[51] G Peter Zhang. 2003. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing* 50 (2003), 159–175.

[52] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, online.