Pre-training Enhanced Spatial-temporal Graph Neural Network for Multivariate Time Series Forecasting

Zezhi Shao
Institute of Computing Technology,
Chinese Academy of Sciences
University of Chinese Academy of Sciences
shaozezhi19b@ict.ac.cn

Zhao Zhang
Institute of Computing Technology,
Chinese Academy of Sciences
zhangzhao2021@ict.ac.cn

Fei Wang*
Institute of Computing Technology,
Chinese Academy of Sciences
wangfei@ict.ac.cn

Yongjun Xu
Institute of Computing Technology,
Chinese Academy of Sciences
xyj@ict.ac.cn

ABSTRACT

Multivariate Time Series (MTS) forecasting plays a vital role in a wide range of applications. Recently, Spatial-Temporal Graph Neural Networks (STGNNs) have become increasingly popular MTS forecasting methods. STGNNs jointly model the spatial and temporal patterns of MTS through graph neural networks and sequential models, significantly improving the prediction accuracy. But limited by model complexity, most STGNNs only consider short-term historical MTS data, such as data over the past one hour. However, the patterns of time series and the dependencies between them (*i.e.,* the temporal and spatial patterns) need to be analyzed based on long-term historical MTS data. To address this issue, we propose a novel framework, in which STGNN is Enhanced by a scalable time series Pre-training model (STEP). Specifically, we design a pre-training model to efficiently learn temporal patterns from very long-term history time series (*e.g.,* the past two weeks) and generate segment-level representations. These representations provide contextual information for short-term time series input to STGNNs and facilitate modeling dependencies between time series. Experiments on three public real-world datasets demonstrate that our framework is capable of significantly enhancing downstream STGNNs, and our pre-training model aptly captures temporal patterns.

CCS CONCEPTS

• **Information systems** → **Data mining**.

KEYWORDS

multivariate time series forecasting, spatial-temporal graph neural network, pre-training model

ACM Reference Format:
Zezhi Shao, Zhao Zhang, Fei Wang, Yongjun Xu. 2022. Pre-training Enhanced Spatial-temporal Graph Neural Network for Multivariate Time Series Forecasting. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539396

*Corresponding author.

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539396

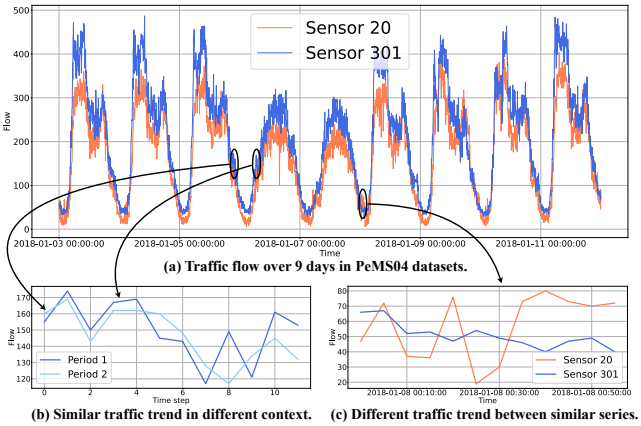


Figure 1: Examples of traffic flow multivariate time series data. (a) The two time series exhibit complex temporal patterns and strong spatial correlations. (b) Similar traffic trends within small windows in different contexts. (c) Different traffic trends within a small window between two similar series.

1 INTRODUCTION

Multivariate time series data is ubiquitous in our lives, from transportation and energy to economics. It contains time series from multiple interlinked variables. Predicting future trends based on historical observations is of great value in helping to make better decisions. Thus, multivariate time series forecasting has remained an enduring research topic in both academia and industry for decades.

Indeed, multivariate time series can be generally formalized as spatial-temporal graph data [36]. On the one hand, multivariate time series have complex temporal patterns, *e.g.,* multiple periodicities. On the other hand, different time series can affect other's evolutionary processes because of the underlying interdependencies between variables, which is non-Euclidean and is reasonably modeled by the graph structure. To illustrate, we take the traffic flow system as an example, where each sensor corresponds to a variable. Figure 1(a) depicts the traffic flow time series generated from two sensors deployed on the road network. Apparently, there

are two repeating temporal patterns, *i.e.,* daily and weekly periodicities. The morning/evening peaks occur every day, while the weekdays and weekends exhibit different patterns. Furthermore, the two time series share very similar trends because the selected sensors 20 and 301 are closely connected in the traffic network. Consequently, accurate time series forecasting depends not only on the pattern of its temporal dimension but also on its interlinked time series. Besides, it is worth noting that we made the above analysis on the basis of observing a sufficiently long time series.

To make accurate predictions, Spatial-Temporal Graph Neural Networks (STGNNs) have attracted increasing attention recently. STGNNs combine Graph Neural Networks (GNNs) [18] and sequential models. The former is used to deal with the dependencies between time series, and the latter is used to learn the temporal patterns. Benefitting from jointly modeling the spatial and temporal patterns, STGNNs have achieved state-of-the-art performance. In addition, an increasing number of recent works are further exploring the joint learning of graph structures and STGNNs since the dependency graph between time series, which is handcrafted by prior knowledge, is often biased and incorrect, even missing in many cases. In short, spatial-temporal graph neural networks have made significant progress for multivariate time series forecasting in many real-world applications. However, there is no free lunch. More powerful models require more complex structures. The computational complexity usually increases linearly or quadratically with the length of the input time series. Further considering the number of time series (*e.g.,* hundreds), it is not easy for STGNNs to scale to very long-term historical time series. In fact, most models use historical data in a small window to make predictions, *e.g.,* use the past twelve time steps (one hour) to predict the future twelve time steps [20, 29, 35, 36, 41]. The inability to explicitly learn from long-term information brings up some intuitive concerns.

Firstly, the STGNN model is blind to the context information beyond the window. Considering that time series are usually noisy, it may be difficult for the model to distinguish short-term time series in different contexts. For example, when observing data within two small windows of length twelve shown in Figure 1(b), we find that the two time series in different contexts are similar. Therefore, it is difficult for models to make accurate predictions about their different future trends based on limited historical data. Secondly, short-term information is unreliable for modeling the dependency graph, which is represented by the similarity (or correlation) between time series. As shown in Figure 1(c), the two time series are not similar when we observe data within the small window, neither in number nor in trend. On the contrary, long-term historical time series are beneficial for resisting noise, which facilitates obtaining more robust and accurate dependencies. Although long-term historical information is beneficial, as mentioned above, it is expensive for the STGNNs to scale to very long-term historical time series directly. Furthermore, the optimization of the model can also become problematic as the length of the input sequence increases.

To address these challenges, we propose a novel framework, in which STGNN is Enhanced by a scalable time series Pre-training model (STEP). The pre-training model aims to efficiently learn the temporal patterns from very long-term historical time series and generate segment-level representations, which contain rich contextual information that is beneficial to address the first challenge. In addition, the learned representations of these segments (*i.e.,* the short-term time series) are able to incorporate the information from the whole long historical time series to calculate the correlation between time series, thus solving the second challenge, the problem of missing the dependency graph. Specifically, we design an efficient unsupervised pre-training model for Time Series based on TransFormer blocks [33] (TSFormer), which is trained through the masked autoencoding strategy [13]. TSFormer efficiently captures information over very long-term historical data over weeks, and produces segment-level representations that correctly reflect complex patterns in time series. Second, we design a graph structure learner based on the representation of TSFormer, which learns discrete dependency graph and utilizes the kNN graph computed based on the representation of TSFormer as a regularization to guide the joint training of graph structure and STGNN. Notably, STEP is a general framework that can extend to almost arbitrary STGNNs. In summary, the main contributions are the following:

- We propose a novel framework for multivariate time series forecasting, where the STGNN is enhanced by a pre-training model. Specifically, the pre-training model generates segment-level representations that contain contextual information to improve the downstream models.
- We design an efficient unsupervised pre-training model for time series based on Transformer blocks and train it by the masked autoencoding strategy. Furthermore, we design a graph structure learner for learning the dependency graph.
- Experimental results on three real-world datasets show that our method can significantly enhance the performance of downstream STGNNs, and our pre-training model aptly captures temporal patterns.

2 PRELIMINARIES

We first define the concept of multivariate time series, the dependency graph. Then, we define the forecasting problem addressed.

Definition 1. ***Multivariate Time Series.*** *A multivariate time series has multiple time-dependent variable, such as observations from multiple sensors. It can be denoted as a tensor $\mathcal{X} \in \mathbb{R}^{T \times N \times C}$, where T is the number of time steps, N is the number of variables, e.g., the sensors, and C is the number of channels. We additionaly denote the data of time series i as $\mathrm{S}^i \in \mathbb{R}^{T \times C}$.*

Definition 2. ***Dependency Graph.*** *Each variable depends not only on its past values but also on other variables. Such dependencies are captured by a dependency graph $\mathcal{G} = (V, E)$, where V is the set of $|V| = N$ nodes, and each node corresponds to a variable, e.g., a sensor. E is the set of $|E| = M$ edges. The graph can also be denoted as an adjacent matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$.*

Definition 3. ***Multivariate Time Series Forecasting.*** *Given historical signals $\mathcal{X} \in \mathbb{R}^{T_h \times N \times C}$ from the past T_h time steps, multivariate time series forecasting aims to predict the values $\mathcal{Y} \in \mathbb{R}^{T_f \times N \times C}$ of the T_f nearest future time steps.*

3 MODEL ARCHITECTURE

As shown in Figure 2, STEP has two stages: the pre-training stage and the forecasting stage. In the pre-training stage, we design a masked autoencoding model for Time Series based on TransFormer

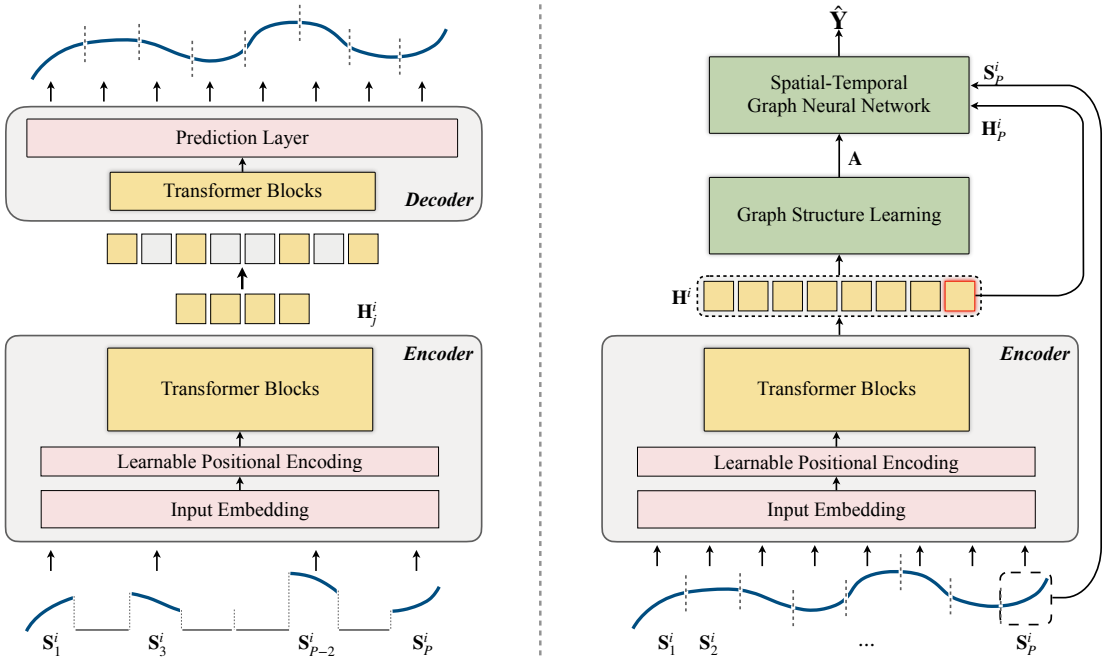


Figure 2: The overview of the proposed STEP framework. Left: the pre-training stage. We split very long-term time series into segments and feed them into TSFormer, which is trained via the masked autoencoding strategy. Right: the forecasting stage. We enhance the downstream STGNN based on the segment-level representations of the pre-trained TSFormer.

blocks (TSFormer) to efficiently learn temporal patterns. TSFormer is capable of learning from the very long-term sequence and gives segment-level representations that contain rich context information. In the forecasting stage, we use the pre-trained encoder to provide context information to enhance the downstream STGNN. Furthermore, based on the representations of the pre-training model, we further design a discrete and sparse graph learner to deal with the cases that the pre-defined graph is missing.

3.1 The Pre-training Stage

In this part, we aim at designing an efficient unsupervised pre-training model for time series. While the pre-training model has made significant progress in natural language processing [3, 8, 27], progress in time series lags behind them. First, we would like to discuss the difference between time series and natural language, which will motivate the design of TSFormer. We attempt to distinguish them from the following two perspectives:

(i) Time series information density is lower. As human-generated signals, each data point in natural language (*i.e.,* a word in a sentence) has rich semantics and is suitable as the data unit for model input. On the contrary, isolated data points in time series give less semantic information. Semantics only arise when we observe at least segment-level data, such as going up or down. On the other hand, language models are usually trained by predicting only a few missing words per sentence. However, masked values in time series can often be trivially predicted by simple interpolation [39], making the pre-training model only focuses on low-level information. To address this problem, a simple strategy that works well is to mask a very high portion of the model's input to encourage the learning of high-level semantics, motivated by recent development in computer vision [13]. It creates a challenging self-supervised task that forces the model to obtain holistic understanding of time series.

(ii) Time series require longer sequences to learn the temporal patterns. In natural languages, sequences of hundreds of lengths have contained rich semantic information. Thus, language pre-training models usually cut or pad the input sequence to hundreds [3, 8, 27]. However, although time series have relatively more straightforward semantics than natural languages, they require longer sequences to learn it. For example, traffic system records data every five seconds, and if we want to learn the weekly periodicity, we need at least consecutive 2016 time slices. Although sampling at a lower frequency is a possible solution, it inevitably loses information. Fortunately, although longer time series will increase the model complexity, we can alleviate it by stacking fewer Transformer blocks and fix model parameters during the forecasting stage to reduce computational and memory overhead.

Motivated by the above analyses and recent computer vision models [9], especially Masked AutoEncoder (MAE) [13], we propose a masked autoencoding model for time series based on Transformer blocks (*i.e.,* TSFormer). TSFormer reconstructs the original signal based on the given partial observed signals. We use an asymmetric design to largely reduce computation: the encoder operates on only partially visible signals, and the decoder uses a lightweight network on the full signals. The model is shown in Figure 2(left), and we will introduce each component in detail next.

Masking. We divide the input sequence $\mathbf{S}^i$ from node i into P non-overlapping patches of length L (Input sequences are obtained over the original time series through a sliding window of length $P * L$). The jth patch can be denoted as $\mathbf{S}^i_j \in \mathbb{R}^{LC}$, where C is the input

channel. We assume L is the commonly used length of input time series of STGNNs. We randomly mask a subset of patches with masking ratio r set to a high number of 75% to create a challenging self-supervised task. Here, we emphasize that the strategy of using patches as input units serves multiple purposes. Firstly, segments (*i.e.,* patches) are more appropriate for explicitly providing semantics than separate points. Secondly, it facilitates the use of downstream models, as downstream STGNNs take a single segment as input. Last but not least, it significantly reduces the length of sequences input to the encoder, and the high masking ratio r makes the encoder more efficient during the pre-training stage.

Encoder. Our encoder is a series of Transformer blocks [33] with an input embedding layer and a positional encoding layer. The encoder only operates on unmasked patches. As the semantics of time series are more straightforward than languages, we use four layers of Transformer blocks, far less than the depth of Transformer-based models in computer vision [9, 13] and natural languages [3, 8]. Specifically, the input embedding layer is a linear projection to transform the unmasked patches into latent space:

$$\mathbf{U}_j^i = \mathbf{W} \cdot \mathbf{S}_j^i + \mathbf{b}, \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{d \times (LC)}$ and $\mathbf{b} \in \mathbb{R}^d$ are learnable parameters, $\mathbf{U}_j^i \in \mathbb{R}^d$ are the model input vectors, and d is the hidden dimension. For masked patches, we use a shared learnable mask token to indicate the presence of a missing patch to be predicted. Next, the positional encoding layer is used to add sequential information. Notably, the positional encoding operates on all patches, although the mask tokens are not used in the encoder. Moreover, unlike the deterministic, sinusoidal embeddings used in MAE [13], we use learnable positional embeddings. On the one hand, in this work, learnable embeddings significantly outperform sinusoidal ones for all datasets. On the other hand, we observe that learned positional embeddings are crucial in learning time series' periodic features, which will be demonstrated in Section 4.3. Finally, we obtain the latent representations $\mathbf{H}_j^i \in \mathbf{R}^d$ through Transformer blocks for all unmasked patches j.

Decoder. The decoder is also a series of Transformer blocks that reconstruct the latent representations back to a lower semantic level, *i.e.,* numerical information. The decoder operates on the full set of patches, including the mask tokens. Unlike MAE [13], we no longer add positional embeddings here since all patches already have positional information added in the encoder. Notably, the decoder is only used during the pre-training stage to perform the sequence reconstruction task, and can be designed independently of the encoder. We use only a single layer of Transformer block for balancing efficiency and effectiveness. Finally, we apply Multi-Layer Perceptions (MLPs) to make predictions whose number of output dimensions equals the length of each patch. Specifically, given the latent representation $\mathbf{H}_j^i \in \mathbb{R}^d$ of patch j, the decoder gives the reconstructed sequence $\hat{\mathbf{S}}_j^i \in \mathbb{R}^{LC}$.

Reconstruction target. Our loss function compute mean absolute error between the original sequence $\mathbf{S}_j^i$ and reconstructed sequence $\hat{\mathbf{S}}_j^i$. Kindly note that we only compute loss over the masked patches, which is in line with other pre-training models [8, 13]. Moreover, all these operations are computed in parallel for all time series i.

In summary, TSFormer is efficient thanks to the high masking ratio and fewer Transformer blocks. TSFormer is capable of learning from the very long-term sequence (*e.g.,* weeks) and can be trained on a single GPU. The encoder generates representations for the input patches (segments). Furthermore, another noteworthy difference from MAE [13] is that we pay more attention to the representations of the patches. On the one hand, we can use the representations to verify periodic patterns in the data, which will be demonstrated in Section 4.3. More importantly, they can conveniently act as contextual information for short-term input of downstream STGNNs, which will be introduced in the next.

3.2 The Forecasting Stage

For a given time series i, TSFormer takes its historical signals $\mathbf{S}^i \in \mathbb{R}^{T_p \times C}$ of the past $T_p = L \times P$ time steps as input. We divide it into P non-overlapping patches of length L: $\mathbf{S}_1^i, \cdots, \mathbf{S}_P^i$, where $\mathbf{S}_j^i \in \mathbb{R}^{L \times C}$. The pre-trained TSFormer encoder generates representations $\mathbf{H}_j^i \in \mathbb{R}^d$ for each $\mathbf{S}_j^i$, where d is the dimension of hidden states. Considering that the computational complexity usually increases linearly or quadratically with the length of the input time series, STGNNs can only take the latest, *i.e.,* the last patch $\mathbf{S}_P^i \in \mathbb{R}^{L \times C}$ for each time series i as input. For example, the most typical setting is $L = 12$. In the forecasting stage, we aim at enhancing the STGNNs based on the representations of the pre-trained TSFormer encoder.

Graph structure learning. Many STGNNs [20, 36, 41] depend on a pre-defined graph to indicate the relationship between nodes (*i.e.,* time series). However, such a graph is not available or is incomplete in many cases. An intuitive idea is to train a matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, where $\mathbf{A}_{ij} \in [0, 1]$ indicates the dependency between time series i and j. However, since the learning of graph structure and STGNNs are coupled compactly, and there is no supervised loss information for graph structure learning [21], optimizing such a contiguous matrix usually leads to a complex bilevel optimization problem [10]. In addition, the dependency $\mathbf{A}_{ij}$ is usually measured by the similarity between time series, which is also a challenging task.

Fortunately, we can alleviate these problems based on the pre-trained TSFormer. Motivated by recent works [10, 17, 29], we aim to learn a discrete sparse graph, where $\mathbf{\Theta}_{ij}$ parameterizes the Bernoulli distribution from which the discrete dependency graph $\mathbf{A}$ is sampled. First, we introduce graph regularization to provide supervised information for graph optimization based on the representations of TSFormer. Specifically, we denote $\mathbf{H}^i = \mathbf{H}_1^i \parallel \mathbf{H}_2^i ... \mathbf{H}_{P-1}^i \parallel \mathbf{H}_P^i \in \mathbb{R}^{Pd}$ as the feature of time series i, where $\parallel$ means the concatenation operation. Then we calculate a kNN graph $\mathbf{A}^a$ among all the nodes. We can control the sparsity of the learned graph by setting different k. Benefiting from the ability of TSFormer, $\mathbf{A}^a$ can reflect the dependencies between nodes, which is helpful to guide the training of the graph structure. Then, we compute $\mathbf{\Theta}_{ij}$ as follows:

$$\begin{aligned} \mathbf{\Theta}_{ij} &= \mathrm{FC}(\mathrm{relu}(\mathrm{FC}(\mathbf{Z}^i \parallel \mathbf{Z}^j))) \\ \mathbf{Z}^i &= \mathrm{relu}(\mathrm{FC}(\mathbf{H}^i)) + \mathbf{G}^i, \end{aligned} \tag{2}$$

where $\mathbf{\Theta}_{ij} \in \mathbb{R}^2$ is the unnormalized probability. The first dimension indicates the probability of positive, and the second dimension indicates the probability of negative. $\mathbf{G}^i$ is the global feature of time series i, which is obtained by a convolutional network

$\mathbf{G}^i = \text{FC}(\text{vec}(\text{Conv}(\mathbf{S}^i_{train})))$, where $\mathbf{S}^i_{train} \in \mathbb{R}^{L_{train}}$ is the entire sequence i over training dataset, and L_{train} is the length of the training dataset. $\mathbf{S}^i_{train}$ is static for all samples during training, helping to make the training process more robust and accurate. The feature $\mathbf{H}^i$ is dynamic for different training samples to reflect the dynamics of dependency graphs [19]. As such, we use the cross-entropy between $\mathbf{\Theta}$ and the kNN graph A^a as graph structure regularization:

$$\mathcal{L}_{graph} = \sum_{ij} -\mathbf{A}^a_{ij} \log \mathbf{\Theta}^{'}_{ij} - (1 - \mathbf{A}^a_{ij}) \log(1 - \mathbf{\Theta}^{'}_{ij}), \quad (3)$$

where $\mathbf{\Theta}^{'}_{ij} = \text{softmax}(\mathbf{\Theta}_{ij}) \in \mathbb{R}$ is the normalized probability.

The last problem of discrete graph structure learning is that the sampling operation from $\mathbf{\Theta}$ to adjacent matrix $\mathbf{A}$ is not differentiable. Hence, we apply the Gumbel-Softmax reparametrization trick proposed by [15, 25]:

$$\mathbf{A}_{ij} = \text{softmax}((\mathbf{\Theta}_{ij} + \mathbf{g})/\tau), \quad (4)$$

where $\mathbf{g} \in \mathbb{R}^2$ is a vector of i.i.d. samples drawn from a Gumbel(0,1) distribution. τ is the softmax temperature parameter. The Gumbel-Softmax converges to one-hot samples (*i.e.,* discrete) when $\tau \to 0$.

Downstream spatial-temporal graph neural network. A normal downstream STGNN takes the last patch and the dependency graph as input, while the enhanced STGNN also considers the input patch's representation. Since the TSFormer has strong power at extracting very long-term dependencies, the representation $\mathbf{H}^i_P$ contains rich context information. STEP framework can extend to almost any STGNN, and we choose a representative method as our backend, the Graph WaveNet [36]. Graph WaveNet captures spatial-temporal dependencies efficiently and effectively by combining graph convolution with dilated casual convolution. It makes predictions based on its output latent hidden representations $\mathbf{H}_{gw} \in \mathbb{R}^{N \times d'}$ by a regression layer, which is a Multi-Layer Perception (MLP). For brevity, we omit its details, and interested readers can refer to the paper [36]. Denoting the representations $\mathbf{H}^i_P$ of TSFormer for all node i as $\mathbf{H}_P \in \mathbb{R}^{N \times d}$, we fuse the representations of Graph WaveNet and TSFormer by:

$$\mathbf{H}_{final} = \text{SP}(\mathbf{H}_P) + \mathbf{H}_{gw}, \quad (5)$$

where $\text{SP}(\cdot)$ is the semantic projector to transform the $\mathbf{H}^i_P$ to the semantic space of $\mathbf{H}_{gw}$. We implement it with a MLP. Finally, we make predictions by the regression layer: $\hat{\mathcal{Y}} \in \mathbb{R}^{T_f \times N \times C}$. Given the ground truth $\mathcal{Y} \in \mathbb{R}^{T_f \times N \times C}$, we use mean absolute error as the regression loss:

$$\mathcal{L}_{regression} = \mathcal{L}(\hat{\mathcal{Y}}, \mathcal{Y}) = \frac{1}{T_f NC} \sum_{j=1}^{T_f} \sum_{i=1}^{N} \sum_{k=1}^{C} |\hat{\mathcal{Y}}_{ijk} - \mathcal{Y}_{ijk}|, \quad (6)$$

where N is the number of nodes, T_f is the number of forecasting steps, and C is the dimensionality of the output. The downstream STGNN and the graph structure is trained in an end-to-end manner:

$$L = L_{regression} + \lambda L_{graph}. \quad (7)$$

We set the graph regularization term λ gradually decay during the training process to go beyond the kNN graph. Notably, the pre-trained TSFormer encoder is fixed in the forecasting stage to reduce computational and memory overhead.

Table 1: Statistics of datasets.

Dataset	# Samples	# Node	Sample Rate	Time Span
METR-LA	34727	207	5mins	4 months
PEMS-BAY	52116	325	5mins	6 months
PEMS04	16969	307	5mins	2 months

4 EXPERIMENTS

In this section, we present experiments on three real-world datasets to demonstrate the effectiveness of the proposed STEP and TSFormer. Furthermore, we conduct comprehensive experiments to evaluate the impact of important hyper-parameters and components. More experimental details, such as optimization settings and efficiency study, can be found in Appendix A, B, and C. It is notable that we conduct pre-training for each dataset since these datasets are heterogeneous in terms of length of time series, physical nature, and temporal patterns. Our code can be found in this repository[1].

4.1 Experimental Setup

Datasets. Following previous works [29, 35, 36], we conduct experiments on three commonly used multivariate time series datasets:

- **METR-LA** is a traffic speed dataset collected from loop-detectors located on the LA County road network [14]. It contains data of 207 selected sensors over a period of 4 months from Mar to Jun in 2012 [20]. The traffic information is recorded at the rate of every 5 minutes, and the total number of time slices is 34,272.
- **PEMS-BAY** is a traffic speed dataset collected from California Transportation Agencies (CalTrans) Performance Measurement System (PeMS) [5]. It contains data of 325 sensors in the Bay Area over a period of 6 months from Jan 1st 2017 to May 31th 2017 [20]. The traffic information is recorded at the rate of every 5 minutes, and the total number of time slices is 52,116.
- **PEMS04** is a traffic flow dataset also collected from CalTrans PeMS [5]. It contains data of 307 sensors in the Bay Area over a period of 2 months from Jan 1st 2018 to Feb 28th 2018 [11]. The traffic information is recorded at the rate of every 5 minutes, and the total number of time slices is 16,992.

The statistical information is summarized in Table 1. For a fair comparison, we follow the dataset division in previous works. For METR-LA and PEMS-BAY, we use about 70% of data for training, 20% of data for testing, and the remaining 10% for validation [20, 36]. For PEMS04, we use about 60% of data for training, 20% of data for testing, and the remaining 20% for validation [11, 12].

Baselines. We select a wealth of baselines that have official public code. Historical Average (HA), VAR [23], and SVR [30] are traditional methods. FC-LSTM [32], DCRNN [20], Graph WaveNet [36], ASTGCN [11], and STSGCN [31] are typical deep learning methods. GMAN [41], MTGNN [35], and GTS [29] are recent state-of-the-art works. More details of baselines can be found in Appendix A.1.

Metrics. We evaluate the performances of all baselines by three commonly used metrics in multivariate time series forecasting, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE).

[1] https://github.com/zezhishao/STEP

Implementation. We set patch size L to 12. We set the number of patches P to 168 for METR-LA and PEMS-BAY, and 336 for PEMS04, *i.e.,* we use historical information for a week for METR-LA and PEMS-BAY, and two weeks for PEMS04. We aim at forecasting the next 12 time steps. The masking ratio r is set to 75%. The hidden dimension of the latent representations of TSFormer d is set to 96. The TSFormer encoder uses 4 layer of Transformer blocks, and the decoder uses 1 layer. The number of attention heads in Transformer blocks is set to 4. The hyper-parameter of Graph WaveNet is set to default in their papers [36]. For the kNN graph $\mathbf{A}^a$, we set k to 10. We perform significance tests (t-test with p-value < 0.05) over all the experimental results.

4.2 Main Results

As shown in Table 2, our STEP framework consistently achieves the best performance in almost all horizons in all datasets, indicating the effectiveness of our framework. GTS and MTGNN jointly learn the graph structure among multiple time series and the spatial-temporal graph neural networks. GTS extends DCRNN by introducing a neighborhood graph as a regularization to improve graph quality and reformulates the problem as a unilevel optimization problem. MTGNN replaces the GNN and Gated TCN in Graph WaveNet with mix-hop propagation layer [1] and dilated inception layer, and proposes to learn latent adjacency matrix to seek further improvement. However, they can not consistently outperform other baselines. Kindly note that the results of GTS may have some gaps with the original paper because it calculates the evaluation metrics in a slightly different manner. Some details can be found in the appendix in the original paper [29] and similar issues in its official code repository[2]. We unify the evaluation process with other baselines, run GTS five times, and report its best performance. GMAN performs better in long-term prediction benefiting from the powerful ability of the attention mechanism in capturing long-term dependency. DCRNN and Graph WaveNet are two typical spatial-temporal graph neural networks. Even compared with many newer works such as ASTGCN and STSGCN, their performance is still very promising. This may be due to their refined and reasonable model architecture. FC-LSTM, a classic recurrent neural network, can not perform well since it only considers temporal features, ignoring the dependencies between time series. Other non-deep learning methods HA, VAR, and SVR perform worst since they have strong assumptions about the data, *e.g.,* stationary or linear. Thus, they can not capture the strong nonlinear and dynamic spatial and temporal correlations in real-world datasets.

In a nutshell, STEP provides stable performance gains for Graph WaveNet by fully exploiting representations extracted by TSFormer from very long-term historical time series. However, despite the significant performance improvements, it is difficult for us to intuitively understand what TSFormer has learned and how it can help STGNNs. In the next subsection, we will inspect the TSFormer and demonstrate the learned multiple periodicities temporal pattern.

4.3 Inspecting The TSFormer

In this subsection, we would like to intuitively explore what TSFormer has learned. We conduct experiments on the PEMS04 dataset.

[2]https://github.com/chaoshangcs/GTS/issues

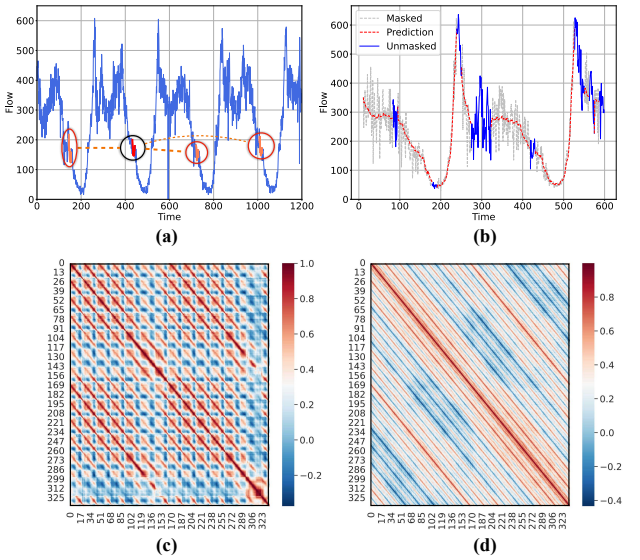


Figure 3: Inspecting the TSFormer. (a) Learned temporal periodicity. (b) Reconstruction. (c) Similarity of latent representations among different patches. (d) Similarity of positional embeddings among different patches.

Specifically, we randomly select a time series in the PEMS04 and then randomly choose a sample of the test dataset to analyze TSFormer. Note that each input sample in PEMS04 has 336 patches of length 12, which means it covers data of the past two weeks.

Learned temporal pattern. Firstly, we would like to explore whether TSFormer learned temporal patterns. We expect it to generate meaningful representations and be able to solve the problem in Figure 1(b). Therefore, we randomly select a patch, compute the cosine similarity with the representations of all the other patches, and select the most similar 3 patches. The result is shown in Figure 3(a), where the original patch is in the black circle, and the selected most similar patches are in the red circle. Apparently, TSFormer has a strong ability to identify similar patches. Furthermore, in order to get the bigger picture, we also calculate the pairwise similarity between all patches and get a 336×336 heat map, where element in i-th column and j-th row indicates the cosine similarity between patch i and patch j. The result shown in Figure 3(c) presents clear daily and weekly periodicities. For each patch, it is similar to the patch at the same time of a day, and the most similar patch usually falls on the same time of the week on the same day. The observation is in line with human intuition. The blue columns or rows mean that the sensor is down or has a large noise fluctuation at this moment, which makes it different from other patches. Since TSFormer has learned the correct relationship between patches, it is reasonable that it can significantly enhance the downstream STGNNs.

Reconstruction visualization. Additionally, we also visualized the results of the TSFormer reconstruction, which is shown in Figure 3(b), where the grey line presents masked patches and the red line demonstrates the reconstruction. The results show that TSFormer can effectively reconstruct masked patches based on a small number of unmasked patches (blue line).

Positional embeddings. Another important difference between TSFormer and MAE [13] and the original Transformer [33] is the

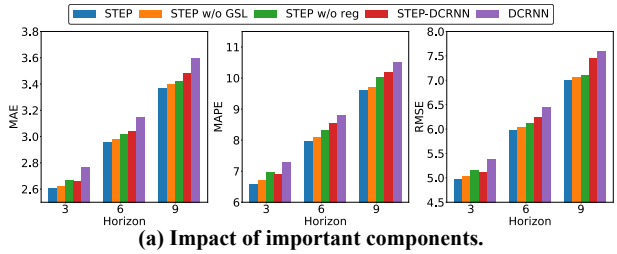
Table 2: Multivariate time series forecasting on the METR-LA, PEMS-BAY, and PEMS04 datasets. Numbers marked with * indicate that the improvement is statistically significant compared with the best baseline (t-test with p-value< 0.05).

Datasets	Methods	Horizon 3			Horizon 6			Horizon 12		
		MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
METR-LA	HA	4.79	10.00	11.70%	5.47	11.45	13.50%	6.99	13.89	17.54%
	VAR	4.42	7.80	13.00%	5.41	9.13	12.70%	6.52	10.11	15.80%
	SVR	3.39	8.45	9.30%	5.05	10.87	12.10%	6.72	13.76	16.70%
	FC-LSTM	3.44	6.30	9.60%	3.77	7.23	10.09%	4.37	8.69	14.00%
	DCRNN	2.77	5.38	7.30%	3.15	6.45	8.80%	3.60	7.60	10.50%
	STGCN	2.88	5.74	7.62%	3.47	7.24	9.57%	4.59	9.40	12.70%
	Graph WaveNet	2.69	5.15	6.90%	3.07	6.22	8.37%	3.53	7.37	10.01%
	ASTGCN	4.86	9.27	9.21%	5.43	10.61	10.13%	6.51	12.52	11.64%
	STSGCN	3.31	7.62	8.06%	4.13	9.77	10.29%	5.06	11.66	12.91%
	GMAN	2.80	5.55	7.41%	3.12	6.49	8.73%	3.44	7.35	10.07%
	MTGNN	2.69	5.18	6.88%	3.05	6.17	8.19%	3.49	7.23	9.87%
	GTS	2.67	5.27	7.21%	3.04	6.25	8.41%	3.46	7.31	9.98%
	STEP	**2.61***	**4.98***	**6.60%***	**2.96***	**5.97***	**7.96%***	**3.37***	**6.99***	**9.61%***
PEMS-BAY	HA	1.89	4.30	4.16%	2.50	5.82	5.62%	3.31	7.54	7.65%
	VAR	1.74	3.16	3.60%	2.32	4.25	5.00%	2.93	5.44	6.50%
	SVR	1.85	3.59	3.80%	2.48	5.18	5.50%	3.28	7.08	8.00%
	FC-LSTM	2.05	4.19	4.80%	2.20	4.55	5.20%	2.37	4.96	5.70%
	DCRNN	1.38	2.95	2.90%	1.74	3.97	3.90%	2.07	4.74	4.90%
	STGCN	1.36	2.96	2.90%	1.81	4.27	4.17%	2.49	5.69	5.79%
	Graph WaveNet	1.30	2.74	2.73%	1.63	3.70	3.67%	1.95	4.52	4.63%
	ASTGCN	1.52	3.13	3.22%	2.01	4.27	4.48%	2.61	5.42	6.00%
	STSGCN	1.44	3.01	3.04%	1.83	4.18	4.17%	2.26	5.21	5.40%
	GMAN	1.34	2.91	2.86%	1.63	3.76	3.68%	1.86	4.32	4.37%
	MTGNN	1.32	2.79	2.77%	1.65	3.74	3.69%	1.94	4.49	4.53%
	GTS	1.34	2.83	2.82%	1.66	3.78	3.77%	1.95	4.43	4.58%
	STEP	**1.26***	**2.73***	**2.59%***	**1.55***	**3.58***	**3.43%***	**1.79***	**4.20***	**4.18%***
PEMS04	HA	28.92	42.69	20.31%	33.73	49.37	24.01%	46.97	67.43	35.11%
	VAR	21.94	34.30	16.42%	23.72	36.58	18.02%	26.76	40.28	20.94%
	SVR	22.52	35.30	14.71%	27.63	42.23	18.29%	37.86	56.01	26.72%
	FC-LSTM	21.42	33.37	15.32%	25.83	39.10	20.35%	36.41	50.73	29.92%
	DCRNN	20.34	31.94	13.65%	23.21	36.15	15.70%	29.24	44.81	20.09%
	STGCN	19.35	30.76	12.81%	21.85	34.43	14.13%	26.97	41.11	16.84%
	Graph WaveNet	18.15	29.24	12.27%	19.12	30.62	13.28%	20.69	33.02	14.11%
	ASTGCN	20.15	31.43	14.03%	22.09	34.34	15.47%	26.03	40.02	19.17%
	STSGCN	19.41	30.69	12.82%	21.83	34.33	14.54%	26.27	40.11	14.71%
	GMAN	18.28	29.32	12.35%	18.75	30.77	12.96%	19.95	**30.21**	12.97%
	MTGNN	18.22	30.13	12.47%	19.27	32.21	13.09%	20.93	34.49	14.02%
	GTS	18.97	29.83	13.06%	19.29	30.85	13.92%	21.04	34.81	14.94%
	STEP	**17.34***	**28.44***	**11.57%***	**18.12***	**29.81***	**12.00%***	**19.27***	31.33	**12.78%***

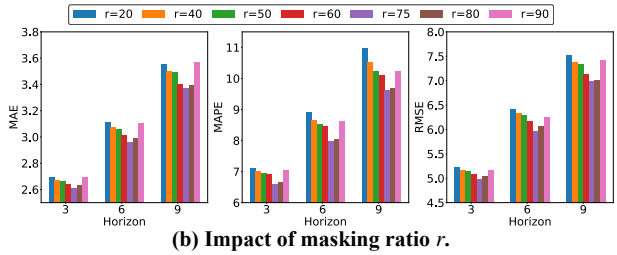
learnable positional embedding. Therefore, we would like to explore whether TSFormer has learned reasonable positional embeddings. We compute the cosine similarity between the positional embeddings of 336 patches and get a 336×336 heat map, shown in Figure 3(d). We find that the positional embedding of TSFormer better reflects the multi-periodicity in time series. This is because, unlike the representation of the encoder, which needs to depend on the input patches, the positional embeddings are completely free to optimize and are less affected by the noise of the input data. We conjecture that such positional embedding is the key factor for the success of TSFormer since we found that TSFormer could not get meaningful representations if we replace the learnable positional embeddings with the deterministic, sinusoidal ones.

4.4 Ablation Study

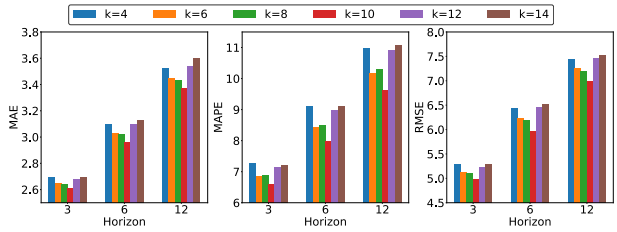
In this part, we conduct experiment to verify the impact of some key components. First, we set *STEP w/o GSL* to test the performance without the graph structure learning model. Second, we



(a) Impact of important components.



(b) Impact of masking ratio r.



(c) Impact of k of kNN graph.

Figure 4: Ablation study and hyper-parameter study.

set *STEP w/o reg* to replace the kNN graph computed by the representations of TSFormer with the kNN graph in GTS [29], which is computed based on the cosine similarity of raw time series $\mathbf{S}^i_{train}$, to test the superiority of the long sequence representations of TSFormer. Finally, we also test more downstream STGNNs to verify the generality of STEP. We choose DCRNN as another backend, *i.e., STEP-DCRNN*. Additionally, we also present the performance of DCRNN for comparison. The results are shown in Figure 4(a).

As can be seen from the figure, STEP outperforms *STEP w/o GSL*, which shows that our graph structure learning module consistently plays a positive role. Meanwhile, *STEP w/o GSL* still achieves satisfactory performance, demonstrating that segment-level representation plays a vital role. STEP also outperforms *STEP w/o reg*, showing that the long sequence representations of TSFormer is superior in improving the graph quality. In addition, as mentioned in Section 1, DCRNN represents a large class of STGNNs [21, 26, 29, 41] that are based on the seq2seq [32] architecture. We fuse the representation of TSFormer to the latent representations of the seq2seq encoder according to Eq.(5). We can see that STEP significantly enhances the performance of DCRNN, which verifies the generality of STEP.

4.5 Hyper-parameter Study

We conduct experiments to analyze the impacts of two hyper-parameters: the masking ratio r, the k of the kNN graph in graph structure learning. We present the results on METR-LA dataset.

The effect of r and k are shown in Figure 4(b) and Figure 4(c), respectively. We find there exist optimal values for both r and k. For masking ratio r, when r is small, masked values in time series can be predicted by simple average or interpolation. Thus it creates a trivial self-supervised learning task and can not get useful representations. When r is large, the model would lose too much information and fail to learn temporal patterns. For k of the kNN graph in graph structure learning, a small value of k would make the learned graph incomplete and lose dependency information, thus the performance is worse. A large value of k would introduce redundancies, which may hurt the information aggregation of graph neural networks, leading to unsatisfactory performance.

5 RELATED WORK

5.1 Spatial-Temporal Graph Neural Networks

The accuracy of multivariate time series forecasting has been largely improved by artificial intelligence [37], especially deep learning techniques. Among these techniques, Spatial-Temporal Graph Neural Networks (STGNNs) are the most promising methods, which combine Graph Neural Networks (GNNs) [7, 18] and sequential models [6, 32] to model the spatial and temporal dependency jointly. Graph WaveNet [36], MTGNN [35], STGCN [38], and StemGNN [4] combine graph convolutional networks and gated temporal convolutional networks with their variants. These methods are based on convolution operation, which facilitates parallel computation. DCRNN [20], ST-MetaNet [26], AGCRN [2], and TGCN [40] combine diffusion convolutional networks and recurrent neural networks [6, 32] with their variants. They follow the seq2seq [32] architecture to predict step by step. Moreover, attention mechanism is widely used in many methods, such as GMAN [41] and ASTGCN [11]. Although STGNNs have made significant progress, the complexity of STGNNs is high because it needs to deal with both temporal and spatial dependency at every step. Therefore, STGNNs can only take short-term historical time series as input, such as the past 1 hour (twelve time steps in many datasets).

More recently, an increasing number of works [10, 17, 29] have focused on joint learning of graph structures and graph neural networks to model the dependencies between nodes. LDS [10] models the edges as random variables whose parameters are treated as hyperparameters in a bilevel learning framework. The random variables parameterize the element-wise Bernoulli distribution from which the adjacency matrix $\mathbf{A}$ is sampled. GTS [29] introduces a neighborhood graph as a regularization that improves graph quality and reformulates the problem as a unilevel optimization problem. Notably, We follow the framework of GTS but enhance it by the pre-training model since TSFormer gives better latent representations of time series for calculating their correlations.

5.2 Pre-training Model

The pre-training model is used to learn a good representation from massive unlabeled data and then use these representations for other downstream tasks. Recent studies have demonstrated significant performance gains on many natural language processing tasks with the help of the representation extracted from pre-training models [28]. Prominent examples are the BERT [8] and GPT [3], which are based on the Transformer encoder and decoder, respectively. The Transformer architecture is more powerful and more efficient than LSTM architecture [27, 32] and has become the mainstream approach for designing pre-training models. More recently, Transformer for images has attracted increasing attention because

of its powerful performance. ViT [9] proposes to split an image into patches and provide the sequence of linear embeddings of these patches as an input to a Transformer, showing impressive performance. However, ViT needs supervised training, which requires massive labeled data. On the contrary, MAE [13] uses self-supervised learning based on the masked autoencoding strategy. MAE enables us to train large models efficiently and effectively and outperforms supervised pre-training. Although the pre-training model has made significant progress in natural language processing and computer vision, progress in time series lags behind them. In this paper, we propose a pre-training model (named TSFormer) for time series based on Transformer blocks and improve the performance of the downstream forecasting task.

6 CONCLUSION

In this paper, we propose a novel STEP framework for multivariate time series forecasting to address the inability of STGNNs to learn long-term information. The downstream STGNN is enhanced by a scalable time series pre-training model TSFormer. TSFormer is capable of efficiently learning the temporal pattern from very long-term historical time series and generating segment-level representations, which provide rich contextual information for short-term input of STGNNs and facilitate modeling dependencies between time series. Extensive experiments on three real-world datasets show the superiority of the STEP framework and the proposed TSFormer.

ACKNOWLEDGMENTS

This work is partially supported by NSFC No. 61902376 and No. 61902382. In addition, Zhao Zhang is supported by the China Postdoctoral Science Foundation under Grant No. 2021M703273.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *ICML*.
[2] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. 2020. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. In *NeurIPS*.
[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models are Few-Shot Learners. In *NeurIPS*.
[4] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. 2020. Spectral Temporal Graph Neural Network for Multivariate Time-series Forecasting. In *NeurIPS*.
[5] Chao Chen, Karl Petty, Alexander Skabardonis, Pravin Varaiya, and Zhanfeng Jia. 2001. Freeway performance measurement system: mining loop detector data. *Transportation Research Record* (2001).
[6] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *SSST@EMNLP*.
[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NeurIPS*.
[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*.
[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.
[10] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning Discrete Structures for Graph Neural Networks. In *ICML*.
[11] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. In *AAAI*.
[12] Shengnan Guo, Youfang Lin, Huaiyu Wan, Xiucheng Li, and Gao Cong. 2021. Learning dynamics and heterogeneity of spatial-temporal graph data for traffic forecasting. *TKDE* (2021).
[13] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2021. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377* (2021).
[14] Hosagrahar V Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M Patel, Raghu Ramakrishnan, and Cyrus Shahabi. 2014. Big data and its technical challenges. *Commun. ACM* (2014).
[15] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. In *ICLR*.
[16] Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.
[17] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. 2018. Neural relational inference for interacting systems. In *ICML*.
[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
[19] Fuxian Li, Jie Feng, Huan Yan, Guangyin Jin, Depeng Jin, and Yong Li. 2021. Dynamic Graph Convolutional Recurrent Network for Traffic Prediction: Benchmark and Solution. *CoRR* (2021). arXiv:2104.14917
[20] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR*.
[21] Haozhe Lin, Yushun Fan, Jia Zhang, and Bing Bai. 2021. REST: Reciprocal Framework for Spatiotemporal-coupled Predictions. In *TheWebConference*.
[22] Ilya Loshchilov and Frank Hutter. 2018. Decoupled Weight Decay Regularization. In *ICLR*.
[23] Zheng Lu, Chen Zhou, Jing Wu, Hao Jiang, and Songyue Cui. 2016. Integrating Granger Causality and Vector Auto-Regression for Traffic Prediction of Large-Scale WLANs. *KSII Trans. Internet Inf. Syst.* (2016).
[24] Helmut Lütkepohl. 2005. *New introduction to multiple time series analysis*. Springer Science & Business Media.
[25] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *ICLR*.
[26] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban traffic prediction from spatio-temporal data using deep meta learning. In *SIGKDD*. 1720–1730.
[27] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *NAACL*.
[28] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences* (2020).
[29] Chao Shang, Jie Chen, and Jinbo Bi. 2021. Discrete Graph Structure Learning for Forecasting Multiple Time Series. In *ICLR*.
[30] Alexander J. Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Stat. Comput.* (2004).
[31] Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. 2020. Spatial-Temporal Synchronous Graph Convolutional Networks: A New Framework for Spatial-Temporal Network Data Forecasting. In *AAAI*.
[32] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *NeurIPS*.
[33] 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 *NeurIPS*.
[34] Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu. 2020. Traffic Flow Prediction via Spatial Temporal Graph Neural Network. In *WWW*.
[35] 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 *SIGKDD*.
[36] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In *IJCAI*.
[37] Yongjun Xu, Xin Liu, Xin Cao, Changping Huang, Enke Liu, Sen Qian, Xingchen Liu, Yanjun Wu, Fengliang Dong, Cheng-Wei Qiu, et al. 2021. Artificial intelligence: A powerful paradigm for scientific research. *The Innovation* 2, 4 (2021).
[38] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *IJCAI*.
[39] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. 2021. A Transformer-based Framework for Multivariate Time Series Representation Learning. In *SIGKDD*.
[40] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2020. T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction. *IEEE TITS* (2020).
[41] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. GMAN: A Graph Multi-Attention Network for Traffic Prediction. In *AAAI*.

A MORE EXPERIMENTS DETAILS

A.1 Baseline Details

- **HA**: Historical Average model, which models time series as a periodic process and uses weighted averages from previous periods as predictions for future periods.
- **VAR**: Vector Auto-Regression [23, 24] assumes that the past time series is stationary and estimates the relationship between the time series and their lag value. [34]
- **SVR**: Support Vector Regression (SVR) uses linear support vector machine for classical time series regression task.
- **FC-LSTM** [32]: Long Short-Term Memory network with fully connected hidden units is a well-known network architecture that is powerful in capturing sequential dependency.
- **DCRNN** [20]: Diffusion Convolutional Recurrent Neural Network [20] replaces the fully connected layer in GRU [6] by diffusion convolutional layer to form a new Diffusion Convolutional Gated Recurrent Unit (DCGRU).
- **Graph WaveNet** [36]: Graph WaveNet stacks gated temporal convolutional layer and GCN layer by layer to jointly capture the spatial and temporal dependencies.
- **ASTGCN** [11]: ASTGCN combines the spatial-temporal attention mechanism to capture the dynamic spatial-temporal characteristics simultaneously.
- **STSGCN** [31]: STSGCN is proposed to effectively capture the localized spatial-temporal correlations and consider the heterogeneity in spatial-temporal data.
- **MTGNN** [35]: MTGNN extends Graph WaveNet through the mix-hop propagation layer in the spatial module, the dilated inception layer in the temporal module, and a more delicate graph learning layer.
- **GMAN** [41]: GMAN is an attention-based model which stacks spatial, temporal and transform attentions.
- **GTS** [29]: GTS learns a graph structure among multiple time series and forecasts them simultaneously with DCRNN.

A.2 Optimization Settings

Table 3: Pre-training setting.

config	value
optimizer	AdamW [22]
base learning rate	5.0e-4
weight decay	0
epsilon	1.0e-8
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$
learning rate schedule	MultiStepLR
milestones	50
gamma	0.5
gradient clip	5

Pre-training stage. The default setting is shown in Table 3. We use uniform distribution to initialize the positional embeddings, and we use truncated normal distribution with $\mu = 0$ and $\sigma = 0.02$ to initialize the mask token, similar to MAE [13]. We use PyTorch official implementation to implement the Transformer blocks. We use the linear scaling rule for learning rate and batch size: lr = base_lr × (batch_size/8) for all datasets in the pre-training stage.

Table 4: Forecasting setting.

config	value
optimizer	Adam [16]
learning rate	0.001/0.005/0.002 (PEMS-BAY/METR-LA/PEMS04)
batch size	64/64/32 (PEMS-BAY/METR-LA/PEMS04)
weight decay	1.0e-5
epsilon	1.0e-8
learning rate schedule	MultiStepLR
milestones	[1, 18, 36, 54, 72]
gamma	0.5
gradient clip	5
cl_num	3
warm_num	30

Forecasting stage. All the settings are shown in Table 4. Following many recent works, such as MTGNN [35] and GTS [29], we use the curriculum learning strategy for the forecasting task. The strategy gradually increases the prediction length of the model with the increase in iteration number. We increase the prediction length by one per cl_num epochs. Moreover, we additionally perform a warm-up of warm_num epochs to better initialize the model for curriculum learning. In addition, λ in Equation (7) decays by $\lambda = 1/(\lceil epoch/6 \rceil)$, where $\lceil \cdot \rceil$ means ceiling function and *epoch* is the epoch number.

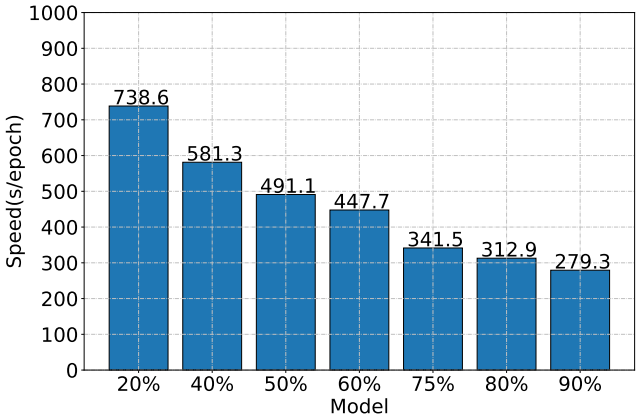


Figure 5: Training speed of different masking ratio r.

B EFFICIENCY

In this part, we compare the efficiency of STEP with other models and their own variants based on the METR-LA dataset. For a more intuitive and effective comparison, we compare the average training time required for each epoch. All the experiments are running on an Intel(R) Xeon(R) Gold 5217 CPU @ 3.00GHz, 128G RAM computing server, equipped with RTX 3090 graphics cards. First, we compare

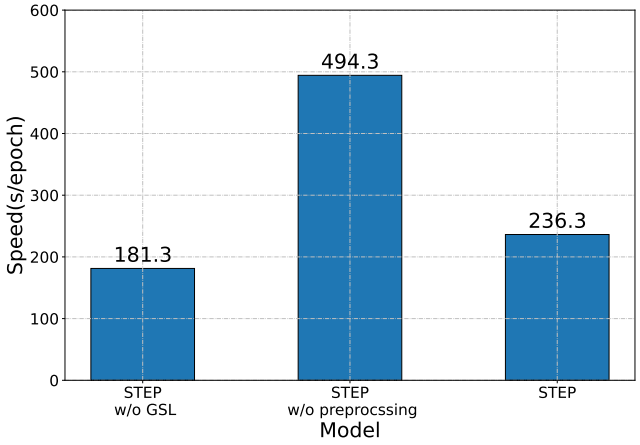


Figure 6: Training speed of different methods.

the efficiency of TSFormer in the pre-training stage under different masking ratios. The result is shown in Figure 5. As the masking ratio increases, the TSFormer will be more efficient. In summary, thanks to the high masking ratio and fewer Transformer blocks in the encoder and decoder, the TSFormer is lightweight and can be trained efficiently on a single NVIDIA 3090 GPU.

Second, we compare the efficiency of STEP framework in the forecasting stage with its variants. Recalling that the parameter of TSFormer is fixed during the forecasting stage, we can use TSFormer to provide off-the-shelf representations by preprocessing the whole dataset to reduce redundant calculations in the training process. We also test the efficiency of STEP without preprocessing, denoting the variant as *STEP w/o pre*. In addition, we test the efficiency of STEP without graph structure learning, *i.e., STEP w/o GSL*. The result is shown in Figure 6. We have the following findings: (i) the graph structure learning module accounts for about 55s per epoch on average. (ii) preprocessing does significantly reduce repetitive computations.

C VISUALIZATION

In order to further intuitively understand and evaluate our model, in this section, we give more visualizations. First, we provide more visualizations about reconstructions of the TSFormer on PEMS04 dataset like Figure 3(b). The results are shown in Figure 7. Note that due to space limitation, we only visualize time series in a small window rather than the whole input time series S^i. Surprisingly, we find that even given very limited information surrounding the unmasked patches, TSFormer reconstructs the masked patches accurately. These results again indicate that our model has a strong ability to learn rich temporal patterns from very long-term time series. Then, we visualize the prediction of our model and the groundtruth data based on METR-LA dataset. We randomly selected six time series and displayed their data from June 13th 2012 to June 16th 2012 (located the test dataset). The forecasting results on six randomly selected time series are shown in Figure 8. We can see that our model can accurately make predictions for different time series. Furthermore, we find that the model has the ability to resist noise. For example, in the right top figure, the traffic sensor apparently failed in the afternoon of June 13th, 2012. However, the model does not overfit the noise.

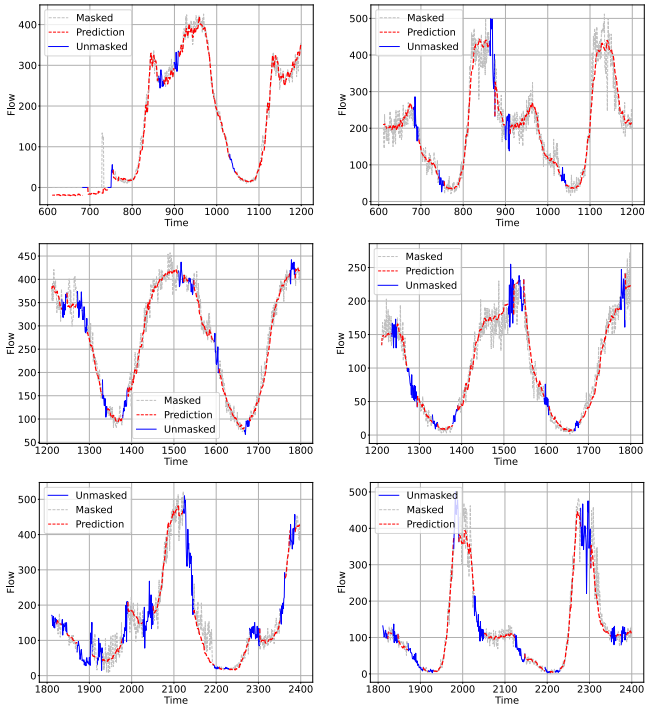


Figure 7: Reconstruction visualizations.

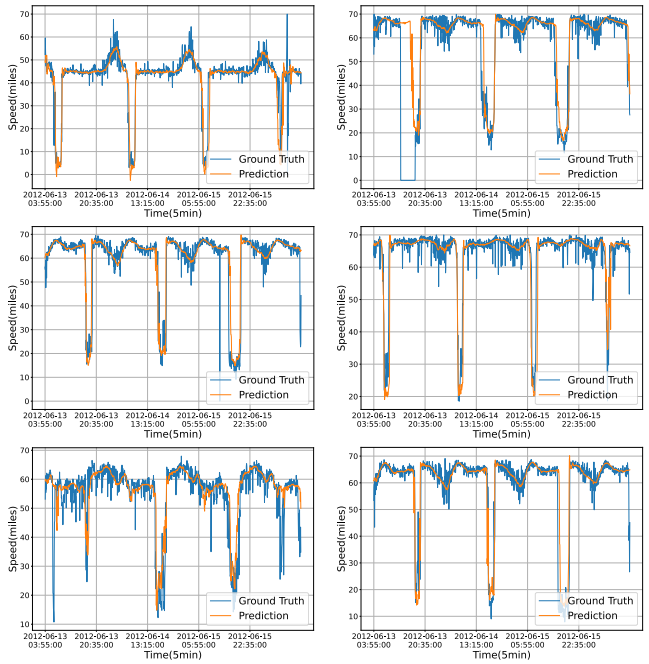


Figure 8: Forecasting visualizations.