SIMST: A GNN-FREE SPATIO-TEMPORAL LEARNING FRAMEWORK FOR TRAFFIC FORECASTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Traffic forecasting is a crucial and challenging problem in smart city efforts. Spatio-Temporal Graph Neural Networks (STGNNs) have demonstrated great promise and become the de facto solution in this field. While successful, they require the message passing scheme of GNNs to construct spatial dependencies between nodes, and thus inevitably inherit the notorious inefficiency of GNNs. Given these facts, in this paper, we propose a simple yet effective GNN-free spatio-temporal learning framework, entitled SimST. Specifically, our framework replaces GNNs with two feasible and efficient spatial context injectors, which provide proximity and position information, respectively. SimST is also compatible with various temporal encoding backbones and involves a tailored training strategy. We conduct extensive experiments on five popular traffic benchmarks to assess the capability of SimST in terms of effectiveness and efficiency. Experimental results show that such a simple baseline performs surprisingly well. Using much fewer parameters, SimST not only achieves comparable or better performance than more sophisticated state-of-the-art STGNNs, but also obtains substantial throughput improvements.

1 INTRODUCTION

Urban traffic forecasting has emerged as one of the most important components of Intelligent Transportation Systems (ITS). Given the historical traffic data collected from sensors over road networks, the task is to predict future traffic conditions (e.g., traffic speed, flow), which provides insights for improving urban planning and traffic management (Zheng et al., 2014). Spatio-Temporal Graph Neural Networks (STGNNs) have become the de facto most popular tool for traffic forecasting, in which sequential models such as Temporal Convolution Networks (TCN) or Recurrent Neural Networks (RNN) are applied for modeling temporal dependencies (Yu et al., 2018; Li et al., 2018; Pan et al., 2019; Wu et al., 2020), and Graph Neural Networks (GNNs) (Kipf & Welling, 2017; Defferrard et al., 2016) are utilized to capture spatial correlations among different locations, by first aggregating features from sensors' (nodes) neighbors, and then transforming the aggregated representations via a feed-forward network.

A fundamental problem for using GNNs in traffic modeling is how to establish a graph structure. After revisiting the STGNNs proposed in recent years, we find that the mainstream approach to build such structures is using either a predefined and sparse adjacency matrix constructed from the distances of the road network between sensors (Yu et al., 2018; Li et al., 2018; Song et al., 2020), or an adaptive adjacency matrix that is typically built by matrix multiplication of two node embedding tables, producing a fully-connected graph (Wu et al., 2019; Bai et al., 2020; Han et al., 2021). Due to the capacity of modeling latent spatial relations, adaptive matrix usually produces superior performance than the predefined one (Wu et al., 2019). Recently, more sophisticated enhancements to spatial learning modules have been proposed to improve predictive performance. For example, Fang et al. (2021); Choi et al. (2022) coupled GNNs with neural ordinary differential equations to produce continuous layers for modeling long-range spatio-temporal dependencies. However, standard GNNs are applied as the basic unit in their methods and they still fall into the above two categories: Fang et al. (2021) use a predefined matrix and Choi et al. (2022) apply an adaptive matrix.

While these approaches can successfully improve forecasting performance, they rely heavily on GNNs for performing the expensive message passing step and thus inevitably inherit or even ex-

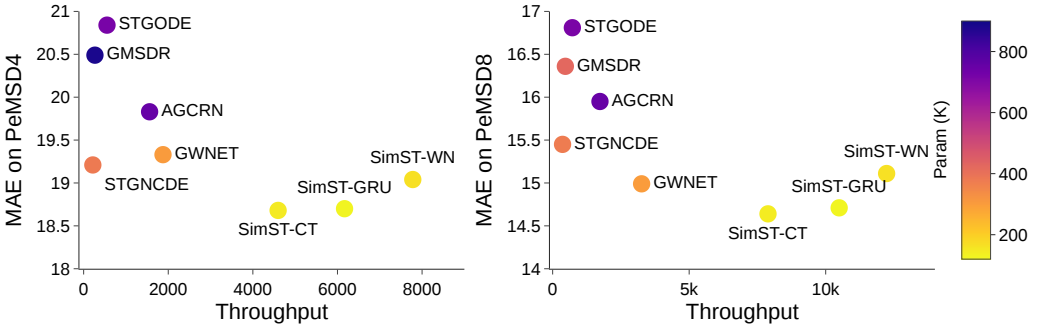

Figure 1: Comparisons between top-performing STGNNs and SimST variants, i.e., SimST-WaveNet (WN), SimST-GRU, and SimST-Causal Transformer (CT) on the popular PeMSD4/D8 datasets. Throughput means the number of samples the network can process in a second during inference.

acerbate GNNs' notorious inefficiencies (Chen et al., 2021a; Zhang et al., 2022). Specifically, the predefined matrix inherits the inefficiency of GNNs and has the complexity of $O(E)$, where E is the number of edges. The adaptive adjacency matrix that has preferable capability aggravates the issue, i.e., it uses a fully-connected graph structure to perform message passing, leading to $O(N^2)$ computational complexity during feature aggregation, where N denotes the number of sensors. Consequently, the bottleneck from GNNs may hinder the applications of STGNNs in large-scale sensor networks, such as metropolitan cities where traffic forecasting systems are urgently needed.

Although there are plenty of efforts in the graph domain to improve the efficiency of GNNs, such as via graph simplification approaches (Hamilton et al., 2017; Zeng et al., 2020), the related studies in STGNNs are still scarce to the best of our knowledge, as such simplification usually leads to information loss and performance degradation (Wu et al., 2020). In this study, we aim to investigate a more challenging problem – how to alleviate the inefficiency of STGNNs while preserving or even improving the learning capacity. Considering the information loss in graph simplification, we approach the problem from a complementary perspective, i.e., a model view to tackle this problem. In other words, we aim to change the mindset that we must use GNNs for spatial modeling, and deliver a solution that trims the explosive complexity but remains competitive in accuracy.

Present work The key challenge after removing GNNs is how to enable nodes to interact with each other to establish relationships between them, since now all nodes are processed individually by the model. In this paper, we propose **SimST**, a simple yet effective spatio-temporal learning framework for traffic forecasting. Specifically, we propose two feasible and efficient ways to inject spatial context into the model: (1) *Spatial position injector.* To allow the model to distinguish between different nodes, we replace GNNs by injecting the model with unique representations of nodes, which are obtained by a non-linear transformation of low-dimensional node embeddings. During end-to-end training, these node-specific embeddings implicitly capture latent spatial correlations in a data-driven manner and can be viewed as personalized features of each node. (2) *Spatial proximity injector.* Considering that spatial correlations are localized in traffic and closer neighbors usually have greater impact on the anchor node (Li et al., 2018), we incorporate neighbor information by explicitly concatenating the historical observations of each node's neighbors in the model input.

Moreover, SimST makes no assumption about the temporal encoding backbones and can be easily adapted to models such as GRU (Chung et al., 2014), WaveNet (Oord et al., 2016), and Transformer (Vaswani et al., 2017). We also devise a tailored training strategy for SimST to further improve performance. This strategy provides greater flexibility in choosing batch size and increases sample diversity during batch formation. In summary, our contributions lie in three aspects.

- We present SimST, a conceptually simple yet effective framework. It replaces GNNs with the proposed spatial injectors that have linear complexity w.r.t N during training and inference. It is also compatible with various temporal encoders and applies a customized training strategy.
- Extensive experiments demonstrate that SimST performs surprisingly well on traffic forecasting benchmarks, which is a baseline result absent in the literature. The results also show that message passing is not necessary for effective spatial modelling in traffic forecasting, which calls into question the prevailing practice in this field and opens up new research possibilities.

- As shown in Figure 1, SimST achieves strong empirical results in terms of both accuracy and throughput, while using much fewer parameters. For example, SimST-WN obtains comparable results to STGNCDE, but is 36× and 34× faster than it on PeMSD4 and PeMSD8, respectively.

2 PRELIMINARIES

We define a sensor network as a graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, where $\mathcal{V}$ is a set of nodes and $|\mathcal{V}| = N$, $\mathcal{E}$ is a set of edges and $|\mathcal{E}| = E$, and $\mathbf{A} \in \mathbb{R}^{N \times N}$ is a weighted adjacency matrix. Following Li et al. (2018), the entries w_{ij} in $\mathbf{A}$ represent the proximity between nodes and are calculated using a thresholded Gaussian kernel (Shuman et al., 2013):

$$w_{ij} = \begin{cases} \exp(-\frac{\mathrm{dist}(v_i, v_j)^2}{\sigma^2}), & \text{if } \mathrm{dist}(v_i, v_j) \leqslant r \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $\mathrm{dist}(v_i, v_j)$ denotes the road network distance between sensors, σ is the standard deviation of distances, and r is a threshold.

The graph G has a unique feature matrix $\mathbf{X}^t \in \mathbb{R}^{N \times F}$ at each time step t, where F is the feature dimensionality. The features consist of a target attribute (e.g., traffic speed or flow) and other auxiliary information, such as time of day (Wu et al., 2019). In traffic forecasting, we aim to learn a function f to predict the target attribute of the future T_f steps based on T_h historical frames:

$$\mathcal{G} : [\mathbf{X}^{(t-T_h):t}; G] \xrightarrow{f} \mathbf{Y}^{t:(t+T_f)} \tag{2}$$

where $\mathbf{X}^{(t-T_h):t} \in \mathbb{R}^{T_h \times N \times F}$ indicates the observations from the time step $t - T_h$ to t (excluding the right endpoint), and $\mathbf{Y}^{t:(t+T_f)} \in \mathbb{R}^{T_f \times N \times 1}$ denotes the T_f-step ahead predictions.

3 METHODOLOGY

As a typical spatio-temporal data mining task, traffic forecasting requires modelling both the spatial and temporal aspects. Figure 2 depicts the overall architecture of the proposed SimST framework, which consists of three major components. We commence by introducing three commonly used temporal encoders to establish the temporal correlations in Section 3.1. We then describe how spatial dependencies are captured by the two proposed spatial context injectors instead of using GNNs in Section 3.2. Finally, we illustrate a tailored training strategy for SimST in Section 3.3.

3.1 TEMPORAL ENCODING BACKBONES

There are several viable options for building the temporal dependencies of a node's history. We take three basic and popular backbones in this study. They are GRU (Chung et al., 2014) (RNN-based), WaveNet (Oord et al., 2016) (TCN-based), and Causal Transformer (Transformer-based).

We first apply an input MLP to map $\mathbf{X} \in \mathbb{R}^{T_h \times N \times F}$ to the input representation $\mathbf{H}_{in} \in \mathbb{R}^{T_h \times N \times D_m}$, where D_m is the model's hidden dimension size. Then, this representation is fed into the temporal encoder. For GRU and WaveNet, they compress the temporal dimension to 1, generating the output representations $\mathbf{H}_T \in \mathbb{R}^{N \times D_m}$, which can be seen as a temporal summary of each node.

In contrast, for Causal Transformers, the output representations $\mathbf{H}_T \in \mathbb{R}^{T_h \times N \times D_m}$ still have the same shape as $\mathbf{H}_{in}$. Here, we take the representation of the last time step as the temporal summary. Moreover, since the standard Transformer is aware of all input time steps during self-attention operations and the traffic status at the current step is in fact not conditioned on its future, we follow WaveNet to introduce causality into the attention mechanism. This ensures that the model cannot violate the temporal order of inputs; such causality can be easily implemented by masking the specific entries in the attention map.

3.2 SPATIAL CONTEXT INJECTORS

The inefficiency of GNNs has been extensively discussed within the graph domain (Chiang et al., 2019; Chen et al., 2021a; Zhang et al., 2022). Utilizing GNNs to explicitly set up spatial correlations via message passing, STGNNs also suffer from flagrant inefficiencies. In this work, we propose two simple alternatives to allow each node to exploit spatial information without message passing.

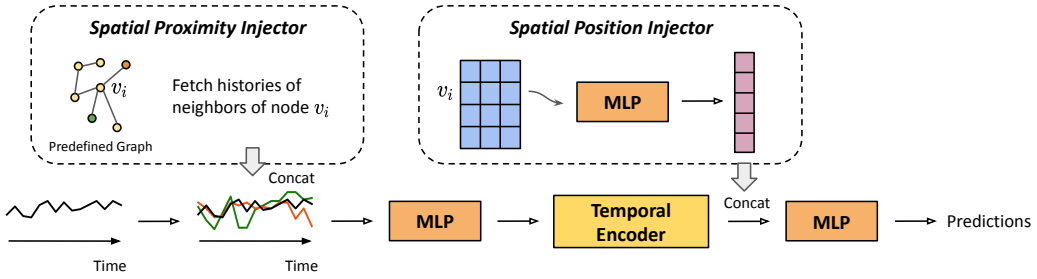

Figure 2: The framework of SimST from the perspective of a node v_i with T_h-step history. This differs from existing STGNNs that require the entire graph as input. The MLPs before and after the temporal encoder are the input MLP for embedding the input data, and the output MLP for producing predictions.

Spatial Position Injector Inspired by positional embeddings in Transformers, we randomly initialize a low-dimensional node embedding table $\mathbf{E} \in \mathbb{R}^{N \times D_n}$, where D_n is the node embedding size. This can be seen as the personalized features possessed by each node. Apart from this, all other trainable parameters in SimST are shared across nodes. We then apply a non-linear transformation to map $\mathbf{E}$ to high-dimensional representations $\mathbf{H}_S \in \mathbb{R}^{N \times D_m}$ and concatenate $\mathbf{H}_S$ with $\mathbf{H}_T$ to form the final spatio-temporal representations $\mathbf{H}_{ST} \in \mathbb{R}^{N \times 2D_m}$, which will be fed into the output MLP to produce the forecasting results. Formally, we have:

$$\mathbf{H}_S = ReLU(\mathbf{E}\mathbf{W} + \mathbf{b}), \quad \mathbf{H}_{ST} = \text{CONCAT}(\mathbf{H}_S, \mathbf{H}_T) \tag{3}$$

During end-to-end training, the embedding table learns the latent spatial relationships between nodes in a data-driven manner. For example, two distant nodes may be inherently correlated because they are both located around residential areas. Then, they may share similar traffic patterns and similar inputs. Such long-range dependencies can be implicitly learned by the model and reflected in their node embeddings' similarity. A further study of the learned embeddings is shown in Section 4.5.

Spatial Proximity Injector According to Tobler's First Law of Geography, near things are more related than distant things. We note that the spatial position injector does not consider the information from the physical neighbors of each node. To complement this, in this component we inject neighbor information by concatenating the histories of each node's neighbors in the model input.

Specifically, we utilize the predefined adjacency matrix $\mathbf{A}$ to obtain the neighbor information. We also consider the neighbors in the reversed direction, i.e., the entries in $\mathbf{A}^{\mathrm{T}}$, so that the model can perceive the impact from the upstream and downstream traffic directions. Since closer neighbors usually have greater correlation with the anchor node (Li et al., 2018), for each node v_i, we concatenate the historical observations from its top-k nearest neighbors with the original observations to highlight their influence. In addition, we also combine the average of the histories of all v_i's hop-one neighbors, allowing the anchor node to learn about its local context. We simplify the time period notation from $(t - T_h) : t$ to T_p here. Formally,

$$\mathbf{X}_{v_i}^{T_p} = \text{CONCAT}(\mathbf{X}_{v_i}^{T_p}, \mathbf{X}_{v_{u_1}}^{T_p}, ..., \mathbf{X}_{v_{u_k}}^{T_p}, \mathbf{X}_{v_{u'_1}}^{T_p}, ..., \mathbf{X}_{v_{u'_k}}^{T_p}, \mathbf{X}_{avg}^{T_p}, \mathbf{X}_{avg'}^{T_p}) \tag{4}$$

where $v_{u_1}, ..., v_{u_k}$ and avg denote v_i's top-k neighbors and average of hop-one neighbors in $\mathbf{A}$. The prime in the notation (e.g., $v_{u'_1}$) indicates neighbors from $\mathbf{A}^{\mathrm{T}}$. Note that this injector operates in the data preprocessing stage, so it has no influence on model training and inference.

Complexity Comparison We provide a complexity comparison between the spatial learning modules here. As GCN (Kipf & Welling, 2017) is the widely adopted form of GNNs in spatio-temporal models, we take it as a baseline. Suppose a L-layer GCN with fixed hidden features D_m are applied. For STGNNs applying the predefined adjacency matrix, the time complexity is $O(LED_m + LND_m^2)$. When applying the adaptive adjacency matrix that can boost performance, the complexity becomes $O(LN^2D_m + LND_m^2)$. In SimST, the complexity of position injector is $O(ND_nD_m)$, and the complexity of proximity injector is $O(NR)$, where R is the average degree of nodes. It is obvious that SimST has lower complexity and is capable of supporting traffic forecasting applications on large-scale sensor networks.

3.3 TRAINING STRATEGY FOR SIMST

When training an STGNN, the input data are usually a four-dimensional tensor $\mathbf{X}^{(t-T_h):t} \in \mathbb{R}^{B\times T_h\times N\times F}$, where B denotes the set batch size. That said, the entire graph is required to be incorporated into a batch, so that the nodes can interact with each other via message passing. However, we identify two main issues with this strategy. First, since SimST processes each node individually, the actual batch size B^* is equal to $B \times N$. For example, there are 307 nodes in the PeMSD4 dataset (Song et al., 2020), and usually B is set to 64, then $B^* = 19648$. Such a large number makes the gradient too exact and affects the effectiveness of stochastic gradient decent. Also, when N is large, it can easily lead to out-of-memory problems. Second, we argue that the sample diversity when forming mini-batches is not enough: for example, the traffic flow of all nodes from 8 a.m. to 9 a.m. on Sept. 28 will always appear in the same batch.

To alleviate the above issues, we simply combine the batch and node dimensions to form a large pool, and then sample batches in this pool during training. Thus, the input data to SimST are $\mathbf{X}^{(t-T_h):t} \in \mathbb{R}^{B^*\times T_h\times F}$ and $B^* \ll B \times N$. The benefits are two-fold. First, our solution provides flexibility to choose the actual batch size, which can be disproportionate to N, thus allowing for smaller values like 256. This property allows SimST to have low GPU memory costs and to easily scale to large graphs. Second, we provide more sample diversity and randomness when creating the batches.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Datasets We conduct experiments on commonly used traffic benchmarks. The details are presented in Table 1. All traffic readings are aggregated into 5-minute windows, resulting in 288 data points per day. Note that the values in traffic flow datasets are generally much larger than those in speed-related datasets. Following Li et al. (2018); Song et al. (2020); Choi et al. (2022), we use the 12-step historical data to predict the next 12 steps. The datasets are divided into three parts for training, validation, and test with a ratio of 6:2:2 on PeMSD4, PeMSD7, PeMSD8, and 7:1:2 on LA and BAY. Z-score normalization is applied to the input data for fast training. We build the predefined adjacency matrix of each dataset by using road network distances with the thresholded Gaussian kernel method (Shuman et al., 2013). The threshold r is set to 0 for PeMSD4, PeMSD7, PeMSD8, and 0.1 for LA and BAY. More information are provided in Appendix A.1

Table 1: Dataset statistics.

Dataset	#Node	#Edge	#Instance	Attribute
PeMSD4 (Song et al., 2020)	307	338	16,992	Flow
PeMSD7 (Song et al., 2020)	883	865	28,224	Flow
PeMSD8 (Song et al., 2020)	170	276	17,856	Flow
LA (Li et al., 2018)	207	1,515	34,272	Speed
BAY (Li et al., 2018)	325	2,369	52,116	Speed

Baselines We compare SimST with the following baselines. Historical Average (HA) (Pan et al., 2012) and Vector Autoregression (VAR) (Toda, 1991) are traditional methods. DCRNN (Li et al., 2018), STGCN (Yu et al., 2018) , ASTGCN (Guo et al., 2019), GWNET (Wu et al., 2019), STSGCN (Song et al., 2020), and AGCRN (Bai et al., 2020) are well-known STGNNs. We also incorporate methods that are published recently, namely, STGODE (Fang et al., 2021), STGNCDE (Choi et al., 2022), and GMSDR (Liu et al., 2022).

Implementation Details SimST is implemented with PyTorch 1.12. There are three variants of SimST based on the applied temporal encoders. We describe the specific configurations for different variants in Appendix A.2. The following settings are the same for all variants. We train our method via the Adam optimizer with an initial learning rate of 0.001 and a weight decay of 0.0001. We set the maximum epochs to 150 and use an early stop strategy with the patience of 20. The batch size is 1,024. The node dimension D_n is 20. The top-k nearest neighbors is set to 3 for LA and

BAY, and 0 for PeMSD4, PeMSD7, and PeMSD8 (due to limited available edges). More details of k are provided in Appendix A.4. For baselines, we run their codes based on the recommended configurations if their accuracy is not known for a dataset. If known, we use their officially reported accuracy. Experiments are repeated five times with different seeds on an NVIDIA RTX A6000 GPU.

Evaluation Metrics We adopt three common metrics in forecasting tasks to evaluate the model performance, including mean absolute error (MAE), root mean squared error (RMSE), and mean absolute percentage error (MAPE). For efficiency measurement, the commonly used floating points of operations (FLOPs) cannot reflect the "real" running speed of methods because it ignores the effects of parallelization. Hence, we use a more reasonable metric, throughput per second (TPS), to measure the efficiency in this study. In particular, TPS indicates the average number of samples the network can process in one second during inference.

Table 2: Test MAE, RMSE, and MAPE results averaged over all predicted time steps on PeMSD4, PeMSD7, and PeMSD8. We bold the best results and underline the second best results. * denotes the improvement of SimST over the best baseline is statistically significant at level 0.05.

Method	PeMSD4			PeMSD7			PeMSD8		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
HA (Pan et al., 2012)	38.03	59.24	27.88%	45.12	65.64	24.51%	34.86	59.24	27.88%
VAR (Toda, 1991)	24.54	38.61	17.24%	50.22	75.63	32.22%	19.19	29.81	13.10%
DCRNN (Li et al., 2018)	22.39	34.93	15.19%	23.41	36.66	9.98%	16.62	25.95	10.60%
STGCN (Yu et al., 2018)	21.59	33.83	15.49%	26.12	41.43	11.92%	17.41	26.72	11.78%
ASTGCN (Guo et al., 2019)	21.58	33.76	14.71%	25.77	39.41	11.67%	17.91	27.34	11.36%
GWNET (Wu et al., 2019)	19.33	30.73	13.18%	20.65	33.47	8.84%	14.99	<u>23.75</u>	9.75%
STSGCN (Song et al., 2020)	21.19	33.65	13.90%	24.26	39.03	10.21%	17.13	26.80	10.96%
AGCRN (Bai et al., 2020)	19.83	32.26	12.97%	20.69	34.19	8.86%	15.95	25.22	10.09%
STGODE (Fang et al., 2021)	20.84	32.82	13.77%	22.99	37.54	10.14%	16.81	25.97	10.62%
STGNCDE (Choi et al., 2022)	19.21	31.09	12.76%	20.53	33.84	8.80%	15.45	24.81	9.92%
GMSDR (Liu et al., 2022)	20.49	32.13	14.15%	22.27	34.94	9.86%	16.36	25.58	10.28%
SimST-WN	19.04	30.74	12.93%	19.88	32.94	8.39%	15.11	24.28	9.96%
SimST-GRU	<u>18.70</u>	<u>30.33</u>	**12.52%***	**19.42***	**32.61***	**8.09%***	<u>14.71</u>	23.96	<u>9.69%</u>
SimST-CT	**18.68***	**30.30***	<u>12.72%</u>	<u>19.70</u>	<u>32.73</u>	<u>8.25%</u>	**14.64***	**23.70**	**9.55%***

Table 3: Test MAE, RMSE, and MAPE results averaged over all predicted time steps on LA, BAY.

Method	LA			BAY		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE
HA (Pan et al., 2012)	4.16	7.80	13.00%	2.88	5.59	6.80%
VAR (Toda, 1991)	5.28	9.06	12.50%	2.24	3.96	4.83%
DCRNN (Li et al., 2018)	3.14	6.28	<u>8.65%</u>	1.63	3.65	3.70%
STGCN (Yu et al., 2018)	3.39	6.79	9.34%	1.88	4.30	4.28%
ASTGCN (Guo et al., 2019)	3.57	7.19	10.32%	1.86	4.07	4.27%
GWNET (Wu et al., 2019)	<u>3.06</u>	<u>6.10</u>	**8.38%**	1.58	3.54	3.59%
STSGCN (Song et al., 2020)	3.32	6.66	9.06%	1.79	3.91	4.06%
AGCRN (Bai et al., 2020)	3.19	6.41	8.84%	1.62	3.61	3.66%
STGODE (Fang et al., 2021)	4.73	7.60	11.71%	1.77	**3.33**	4.02%
STGNCDE (Choi et al., 2022)	3.58	6.84	9.91%	1.68	3.66	3.80%
GMSDR (Liu et al., 2022)	3.21	6.41	8.76%	1.69	3.80	3.74%
SimST-WN	3.10	6.21	8.88%	1.57	3.47	3.58%
SimST-GRU	3.08	6.16	8.84%	**1.55***	<u>3.40</u>	**3.50%***
SimST-CT	**3.04**	**6.08***	8.69%	<u>1.56</u>	3.45	<u>3.56%</u>

4.2 Performance Comparisons

In this subsection, we conduct model comparisons between SimST variants and state-of-the-art STGNNs on five real-world traffic benchmarks. The three variants of SimST are SimST-WaveNet (WN), SimST-GRU, and SimST-Causal Transformer (CT). As shown in Table 2 and 3, all variants of SimST achieve comparable or better performance than all competing baselines on the three evaluation metrics. This is a surprising discovery, since using GNNs to model spatial dependencies is

a common practice in traffic forecasting. The results also demonstrate that: (1) SimST is temporal-model-agnostic, and (2) it is viable and sufficient to use the proposed spatial context injectors to establish spatial correlations between nodes. We next compare only among SimST variants: note that for fairness we ensure that the variants have a similar number of parameters (around 150k). From the table, we observe that SimST-CT performs better than the other two variants in 3 of the 5 datasets, which can be attributed to Transformers' strong learning capability. At the same time, it is also surprising that integrating SimST with a simple GRU can achieve competitive performance, particularly on PeMSD7 and BAY. Additionally, we observe that: (1) STGNNs approaches surpass HA and VAR by a large margin due to their greater learning capacity. (2) The capability of methods that apply the adaptive matrix, such as GWNET and AGCRN, have been overlooked. They have achieved similar performance to the recently proposed approaches.

Table 4: Efficiency comparisons between five best-performing baselines and SimST variants. Param denotes the number of learnable parameters and TPS means throughput per second on the validation set. The magnitude of Param is Kilo (10^3). We bold the highest TPS values.

Method	PeMSD4		PeMSD7		PeMSD8		LA		BAY	
	Param	TPS	Param	TPS	Param	TPS	Param	TPS	Param	TPS
GWNET	303	1,876	314	522	300	3,258	301	2,825	303	1,738
AGCRN	749	1,559	755	445	747	1,733	748	1,604	749	1,466
STGODE	715	552	733	226	710	721	709	682	713	532
STGNCDE	377	217	388	76	374	362	375	251	377	204
GMSDR	880	267	2253	119	423	465	641	300	923	238
SimST-WN	167	**7,780**	179	**2,948**	164	**12,232**	165	**9,335**	168	**6,113**
SimST-GRU	130	6,167	142	2,268	127	10,494	129	7,766	131	5,440
SimST-CT	147	4,597	159	1,656	144	7,887	146	5,969	148	4,041

4.3 Efficiency Comparisons

In this part, we select the top-performing approaches and compare their efficiency with SimST variants. It can be seen from Table 4 that all SimST variants are significantly more efficient than baselines, thanks to their simple model architecture and linear time complexity w.r.t N. Specifically, comparing between TCN-based models, SimST-WN has around 45% fewer parameters than GWNET, but achieves 3.3× - 5.6× higher TPS. For RNN-based methods, SimST-GRU is 3.7× - 6.1× and 19× - 26× faster than AGCRN and GMSDR, respectively. STGODE and STGNCDE are the recently published methods, which are based on neural ordinary differential equations. Though achieving competitive performance, they suffer from non-ignorable inefficiency problem. SimST-WN has comparable performance to them, but is 11× - 17× and 30× - 39× faster. Among the variants, SimST-WN achieves the highest TPS due to WaveNet's superior parallelism capability. SimST-GRU also achieves good TPS results, which we attribute to an optimized implementation in the PyTorch library. See Appendix A.3 for more results on efficiency comparisons.

4.4 Ablation Studies

We have verified the superiority of SimST against STGNNs baselines. Next, we select SimST-CT and SimST-GRU as the representatives due to their preferable performance and conduct a series of ablation studies on one flow-based dataset PeMSD4 and one speed-related dataset LA.

Effects of spatial injectors. To study the effects of two spatial injectors, we consider the following three variants for comparisons: (1) **w/o Position**: we turn off the injection of spatial position representations. (2) **w/o Proximity**: for each node, we only input the histories of itself. (3) CT/GRU: we do not inject any spatial information. The results are shown in Figure 3. First, we find that removing position injector leads to a significant degradation on MAE for both SimST-CT and SimST-GRU on both datasets, revealing the great importance of distinguishing different nodes in the models. Second, the benefit of proximity injector is marginal on the PeMSD4 dataset. The reason is that the average degree of nodes in PeMSD4 is 1.1 and thus the available neighbor information is very limited. In contrast, the average degree of nodes in the LA dataset is 7.3, where the proximity injector can have a larger impact on performance. The results on the PeMSD4 dataset also suggest that on some real-world datasets, even without knowing the neighbor information, the model can achieve very competitive result by only perceiving the latent relationships provided by the position injector.

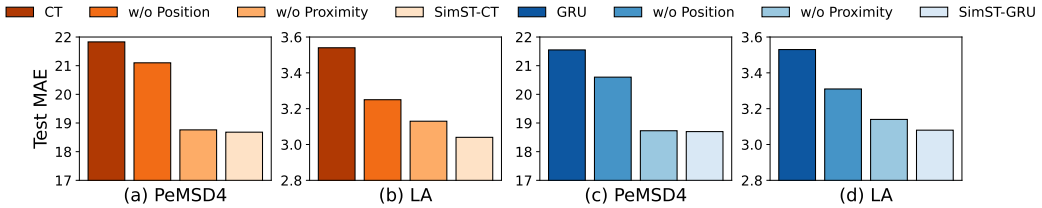

Figure 3: Effects of two spatial context injectors.

Effects of training strategy. We examine the influence of the batch size number in Figure 4 (a) and (b). Generally, both SimST-CT and SimST-GRU reach their best performance when setting B^* to 1,024 on the two datasets, revealing the necessity of applying small B^*. The influence of using a large batch size such as 19,648, is more significant on the PeMSD4 dataset. Besides, we notice that SimST-CT only occupies around 2GB of memory on GPU across all datasets. In contrast, the memory cost of STGNNs usually grows as N increases. For example, on the dataset with the highest N, PeMSD7 (883 nodes), and the lowest N, PeMSD8 (170 nodes), GWNET requires around 11GB and 3GB of memory, respectively. Furthermore, we investigate the effects of different batch forming methods in Table 5, where a SimST-CT model is trained by the strategies applied in STGNNs and SimST, respectively. It can be seen that under a similar actual batch size B^*, the strategy of SimST surpasses the counterpart on both datasets. This result verifies our speculation in Section 3.3 that offering more sample diversity during batch establishment can improve performance.

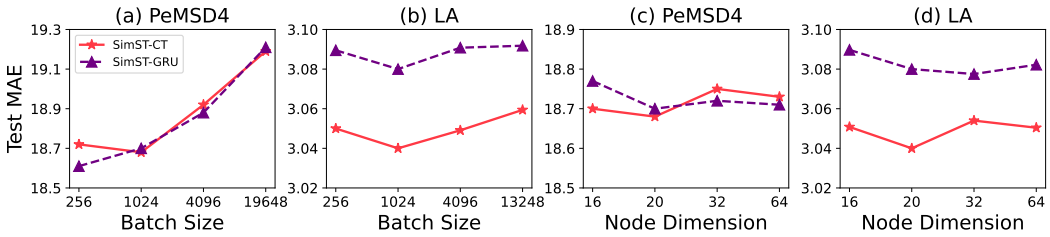

Figure 4: Effects of the actual batch size B^* and the node embedding dimension. In (a) and (b), the largest values in the x-axis are computed by N $\times$ 64 (the common batch size used in STGNNs (Wu et al., 2019; Bai et al., 2020; Choi et al., 2022)).

Table 5: Comparison of training the SimST-CT model by using the training strategies applied in STGNNs and SimST. B^* is calculated by $B \times N$. Dim: dimension.

Strategy	Input Dim	PeMSD4 (N=307)			LA (N=207)		
		B	B^*	MAE	B	B^*	MAE
STGNNs	4	4	1,228	18.95	4	828	3.07
SimST	3	1,024	1,024	**18.68**	1,024	1,024	**3.04**

Effects of node dimension size. We assess the effects of the node dimension size in Figure 4 (c) and (d). We observe that both SimST-CT and SimST-GRU are not sensitive to the changes of the dimension size on both datasets. The results also suggest that a small dimension such as 16, is sufficient to record the latent correlations between nodes.

4.5 Case Study

We further conduct a case study to investigate the effectiveness of the learned node embeddings. Figure 5 (a) shows the embedding visualization by t-SNE (Van der Maaten & Hinton, 2008). Due to page limits, we randomly select a region where node 43, 82, and 130 are located in. We also plot the nodes' raw traffic data in sub-figure (b) and (c). It can be seen that the three nodes are not only similar in the embedding space, but also share similar input traffic patterns. Moreover, by examining the adjacency matrix, we know that node 43's neighbors include node 81, 82, and 158. However in (a), two neighbors 81 and 158 are far from node 43. Also, we can see from (b) and (c) that the patterns of 81 and 158 have significant discrepancy to node 43. These results indicate that the

learned node embeddings can reflect the similarity in the input space. Moreover, given the fact that our node embeddings are not generated from input features, we conclude that the node embeddings have learned the inherent relations between nodes and SimST does work in a data-driven manner, i.e., nodes with similar patterns are likely to have similar node embeddings.

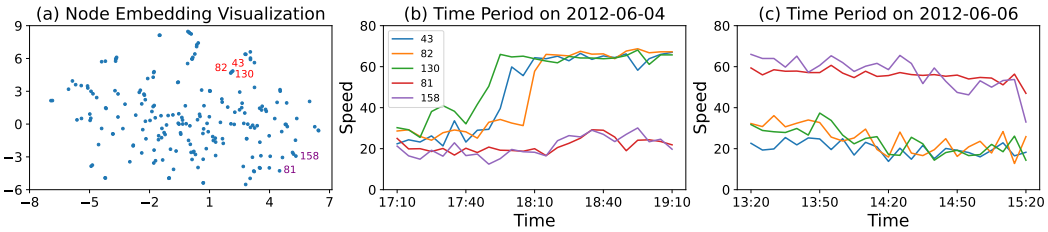

Figure 5: A case study on the test set of the LA dataset by using SimST-CT.

5 More Related Work

Traffic forecasting is a crucial application in smart city efforts. In recent years, Spatial-Temporal Graph Neural Networks (STGNNs) have become the most widely used tools for predicting traffic (Cao et al., 2020; Li & Zhu, 2021; Chen et al., 2021b; 2022). Generally, they integrate GNNs with either RNNs or TCNs to capture the spatial and temporal dependencies in traffic data. For example, DCRNN (Li et al., 2018) considered traffic flow as a diffusion process and combined a novel diffusion convolution with GRU. The following efforts (Pan et al., 2019; Bai et al., 2020; Fang et al., 2021; Liu et al., 2022) were also based on RNNs. To improve training speed and enjoy parallel computation, plenty of works (Wu et al., 2019; 2020; Li & Zhu, 2021; Han et al., 2021) replaced RNNs with dilated causal convolution operations.

A prerequisite for using GNNs effectively, however, is to have a high-quality adjacency matrix. In traffic forecasting, there are mainly two approaches, namely predefined adjacency matrix and adaptive adjacency matrix. Specifically, the former is usually a sparse matrix constructed from road network distances, and the latter is a dense matrix that records the pairwise relationships of nodes. For a comparison between the two matrices, researchers have pointed out that the predefined matrix is heuristic and cannot reflect the genuine dependencies between nodes, which negatively affects model performance (Wu et al., 2019; Bai et al., 2020). Indeed, we notice that the models applying the adaptive matrix achieve superior performance (Wu et al., 2019; Bai et al., 2020; Choi et al., 2022). Though successful, the adaptive matrix discards the sparsity of the graph and incurs quadratic computational cost, making it difficult for models using it to scale to large-scale traffic applications. In addtion, we note that there are also methods that use attention mechanism for spatial learning (Zheng et al., 2020). But they also suffer from quadratic time complexity. In this study, we propose SimST to alleviate this inefficiency issue, which removes the inefficient GNNs component and thus has linear time complexity.

6 Conclusion and Future Work

In this paper, we propose SimST, a simple, effective, and efficient spatio-temporal learning framework for traffic forecasting. It replaces GNNs with two proposed spatial injectors, involves a tailored training strategy, and is temporal-model-agnostic. Our study suggests that message passing is not the only effective way of modelling spatial relations in traffic forecasting; hence, we hope to spur the development of new model designs with both high efficiency and effectiveness in the community.

We also would like to point out several potential future directions. First, the node-specific representations in SimST are not varying over time. Thus, a possible direction for improving performance is to make it change dynamically with time. Second, though SimST can generalize to newly added sensors easily, it still needs to be fine-tuned over the new nodes and is not in a fully inductive manner. A future study may focus on developing a completely inductive spatio-temporal model. Third, our finding and analysis in this work are limited to traffic forecasting currently, a potential future work would be to expand SimST to other spatio-temporal applications such as air quality prediction.

REFERENCES

Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 17804–17815, 2020.

Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, et al. Spectral temporal graph neural network for multivariate time-series forecasting. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 17766–17778, 2020.

Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning*, pp. 1695–1706, 2021a.

Yuzhou Chen, Ignacio Segovia, and Yulia R Gel. Z-gcnets: time zigzags at graph convolutional networks for time series forecasting. In *International Conference on Machine Learning*, pp. 1684–1694, 2021b.

Yuzhou Chen, Ignacio Segovia-Dominguez, Baris Coskunuzer, and Yulia Gel. Tamp-s2gcnets: coupling time-aware multipersistence knowledge representation with spatio-supra graph convolutional networks for time-series forecasting. In *International Conference on Learning Representations*, 2022.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266, 2019.

Jeongwhan Choi, Hwangyong Choi, Jeehyun Hwang, and Noseong Park. Graph neural controlled differential equations for traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 6367–6374, 2022.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 3837–3845, 2016.

Zheng Fang, Qingqing Long, Guojie Song, and Kunqing Xie. Spatial-temporal graph ode networks for traffic flow forecasting. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 364–373, 2021.

Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 922–929, 2019.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

Liangzhe Han, Bowen Du, Leilei Sun, Yanjie Fu, Yisheng Lv, and Hui Xiong. Dynamic and multi-faceted spatio-temporal deep learning for traffic speed forecasting. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 547–555, 2021.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Mengzhang Li and Zhanxing Zhu. Spatial-temporal fusion graph neural networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4189–4196, 2021.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

Dachuan Liu, Jin Wang, Shuo Shang, and Peng Han. Msdr: Multi-step dependency relation networks for spatial temporal forecasting. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1042–1050, 2022.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *The 9th ISCA Speech Synthesis Workshop*, pp. 125, 2016.

Bei Pan, Ugur Demiryurek, and Cyrus Shahabi. Utilizing real-world transportation data for accurate traffic prediction. In *12th IEEE International Conference on Data Mining*, pp. 595–604, 2012.

Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. Urban traffic prediction from spatio-temporal data using deep meta learning. In *Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1720–1730, 2019.

David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, pp. 83–98, 2013.

Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 914–921, 2020.

Hiroyuki Toda. *Vector autoregression and causality*. Yale University, 1991.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of International Joint Conference on Artificial Intelligence*, pp. 1907–1913, 2019.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 753–763, 2020.

Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of International Joint Conference on Artificial Intelligence*, pp. 3634–3640, 2018.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. In *International Conference on Learning Representations*, 2022.

Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 1234–1241, 2020.

Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, pp. 1–55, 2014.

A APPENDIX

A.1 DATASETS

We conduct experiments on the traffic datasets of PeMSD4[1], PeMSD7[1], PeMSD8[1], LA[2] and BAY[2].

- PEMS-04 (Song et al., 2020): The dataset refers to the traffic flow data in San Francisco Bay Area. There are 307 sensors and the period of data ranges from Jan. 1 - Feb. 28, 2018.
- PEMS-07 (Song et al., 2020): This dataset involves traffic flow readings collected from 883 sensors, and the time range is from May 1 - Aug. 6, 2017.
- PEMS-08 (Song et al., 2020): The dataset contains traffic flow information collected from 170 sensors in the San Bernardino area from Jul. 1 - Aug. 31, 2016.
- LA (Li et al., 2018): The dataset records the traffic speed information collected from 207 loop detectors in the highway of Los Angeles County, ranging from Mar. 1 - Jun. 27, 2012.
- BAY (Li et al., 2018): This dataset contains traffic speed information from 325 sensors in the Bay Area. It has 6 months of data ranging from Jan. 1 - Jun. 30, 2017.

We use one-hour (12-step) historical data to predict the future one-hour (12-step) values. The datasets are split into three parts for training, validation, and testing with a ratio of 6:2:2 on PeMSD4, PeMSD7, and PeMSD8, and 7:1:2 on LA and BAY. The statistics of data partitions is in Table 6.

Table 6: Details of data partition.

Dataset	#Training	#Validation	#Testing
PeMSD4	10,172	3,375	3,376
PeMSD7	16,911	5,622	5,622
PeMSD8	10,690	3,548	3,549
LA	23,974	3,425	6,850
BAY	36,465	5,209	10,419

A.2 CONFIGURATIONS OF SIMST VARIANTS

Table 7: Detailed configurations of different variants of SimST. Dim: dimension.

Setting	SimST-WN	SimST-GRU	SimST-CT
Input MLP Dim	64	64	64
Temporal Encoder Hidden Dim	64	64	64
Kernel Size in WN	3	-	-
Skip Connection Dim in WN	64	-	-
Head Number in CT	-	-	2
Feed-Forward Network Dim in CT	-	-	128
Output MLP Dim	512	512	512
Layer Number	3	2	2
Dropout Ratio	0.1	0.1	0.1
Clip Norm	5	5	5

A.3 TRAINING AND INFERENCE TIME COMPARISONS

In Table 8, we provide the training and inference time per epoch of SimST and baselines, to enable straightforward efficiency comparisons. It can be seen that SimST variants have significantly shorter inference time. For the training efficiency, GWNET and AGCRN generally have shorter training time than SimST. The reason is that SimST applies a small actual batch size $B^* = 1,024$.

[1] https://github.com/Davidham3/STSGCN

[2] https://github.com/liyaguang/DCRNN

Table 8: Efficiency comparisons between five best-performing baselines and SimST variants. Train: training time (s) per epoch. Infer: inference time (s) per epoch.

Method	PeMSD4		PeMSD7		PeMSD8		LA		BAY	
	Train	Infer	Train	Infer	Train	Infer	Train	Infer	Train	Infer
GWNET	19.45	1.81	101.32	10.79	11.46	1.10	30.63	1.22	72.11	3.02
AGCRN	22.46	2.18	98.59	12.66	16.19	2.07	40.18	2.15	68.42	3.58
STGODE	69.46	6.11	272.90	24.88	67.56	4.92	151.20	4.99	264.27	9.75
STGNCDE	148.54	15.55	716.63	73.97	94.22	9.80	337.84	13.65	571.39	25.53
GMSDR	80.63	12.70	311.61	47.32	50.96	7.71	170.74	11.52	328.18	22.05
SimST-WN	42.90	0.43	205.25	1.91	24.47	0.29	72.12	0.37	177.49	0.85
SimST-GRU	23.76	0.55	112.39	2.48	13.64	0.34	39.75	0.45	94.08	0.96
SimST-CT	39.82	0.74	184.74	3.40	22.32	0.45	62.95	0.58	152.57	1.30

A.4 Impact of Top-k Neighbors

In Figure 6, we show the effects of k on the LA dataset within the range of $\{0, 1, 2, 3, 4\}$. For the PeMSD4 dataset, it is not necessary to do so because the average degree of nodes is only 1.1.

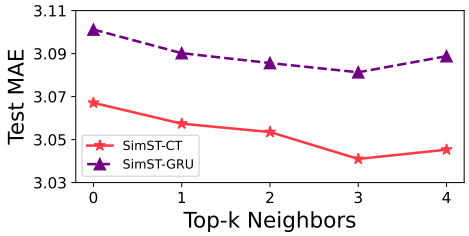

Figure 6: Effects of the top-k neighbors on the LA dataset.

A.5 Learning Curve Comparisons

In Figure 7, we compare the learning curves between some baselines and the variants of SimST. It can be seen that all SimST variants exhibit very fast convergence rates, dropping to a small validation MAE within a few epochs. In addition, we also observe that the standard deviation of SimST is generally smaller than those in baselines, indicating the additional stability provided by our approach.

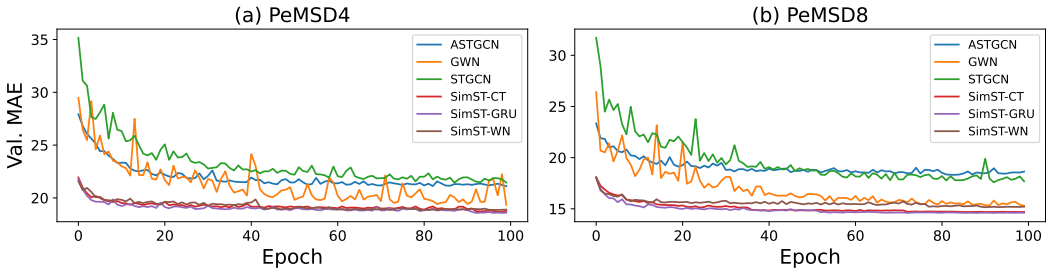

Figure 7: Learning curves of baselines and SimST variants on the validation set of PeMSD4/D8.

A.6 Visualization

To have a better sense of the capacity of our method, in this part, we randomly select two nodes and visualize the forecasting results of the SimST-CT model and the corresponding ground truths.

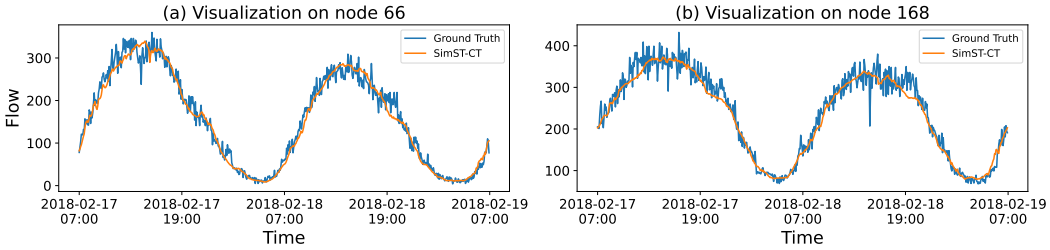

Figure 8: Visualization on the test set of PeMSD4.

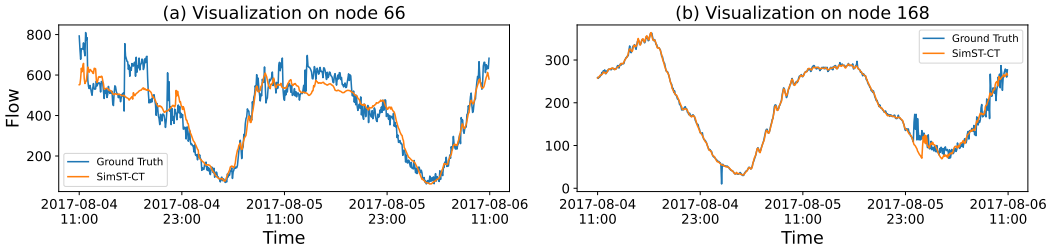

Figure 9: Visualization on the test set of PeMSD7.

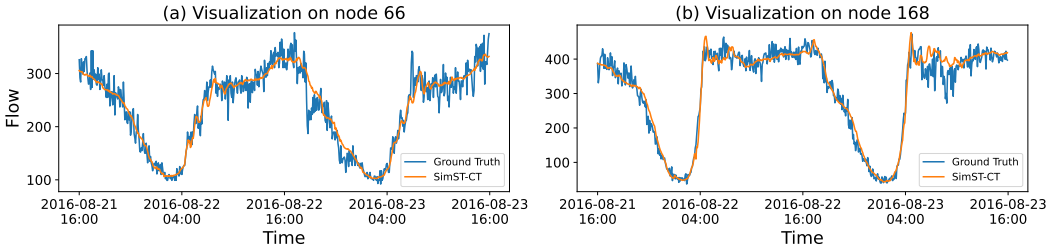

Figure 10: Visualization on the test set of PeMSD8.