Spatio-temporal graph convolutional network for stochastic traffic speed imputation

Cuza, Carlos Enrique Muniz; Ho, Nguyen; Zacharatou, Eleni Tzirita; Pedersen, Torben Bach; Yang, Bin

Published in:
30th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS 2022

DOI (link to publication from Publisher):
10.1145/3557915.3560948

Creative Commons License
CC BY 4.0

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Cuza, C. E. M., Ho, N., Zacharatou, E. T., Pedersen, T. B., & Yang, B. (2022). Spatio-temporal graph convolutional network for stochastic traffic speed imputation. In M. Renz, M. Sarwat, M. A. Nascimento, S. Shekhar, & X. Xie (Eds.), *30th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS 2022* (pp. 1-12). Article 14 Association for Computing Machinery. https://doi.org/10.1145/3557915.3560948

Spatio-Temporal Graph Convolutional Network for Stochastic Traffic Speed Imputation

Carlos Enrique Muniz Cuza
Aalborg University
cemc@cs.aau.dk

Nguyen Ho
Aalborg University
ntth@cs.aau.dk

Eleni Tzirita Zacharatou
IT University of Copenhagen
elza@itu.dk

Torben Bach Pedersen
Aalborg University
tbp@cs.aau.dk

Bin Yang
Aalborg University
byang@cs.aau.dk

ABSTRACT

The rapid increase of traffic data generated by different sensing systems opens many opportunities to improve transportation services. An important opportunity is to enable stochastic routing that computes the arrival time probabilities for each suggested route instead of only the expected travel time. However, traffic datasets typically have many missing values, which prevents the construction of stochastic speeds. To address this limitation, we propose the Stochastic Spatio-Temporal Graph Convolutional Network (SST-GCN) architecture that accurately imputes missing speed distributions in a road network. SST-GCN combines Temporal Convolutional Networks and Graph Convolutional Networks into a single framework to capture both spatial and temporal correlations between road segments and time intervals. Moreover, to cope with datasets with many missing values, we propose a novel self-adaptive context-aware diffusion process that regulates the propagated information around the network, avoiding the spread of false information. We extensively evaluate the effectiveness of SST-GCN on real-world datasets, showing that it achieves from 4.6% to 50% higher accuracy than state-of-the-art baselines using three different evaluation metrics. Furthermore, multiple ablation studies confirm our design choices and scalability to large road networks.

CCS CONCEPTS

• **Computing methodologies → Neural networks**.

KEYWORDS

data imputation, spatio-temporal, graph convolutional networks

ACM Reference Format:
Carlos Enrique Muniz Cuza, Nguyen Ho, Eleni Tzirita Zacharatou, Torben Bach Pedersen, and Bin Yang. 2022. Spatio-Temporal Graph Convolutional Network for Stochastic Traffic Speed Imputation. In *The 30th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '22), November 1–4, 2022, Seattle, WA, USA.* , 12 pages. https://doi.org/10.1145/3557915.3560948

SIGSPATIAL '22, November 1–4, 2022, Seattle, WA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9529-8/22/11.
https://doi.org/10.1145/3557915.3560948

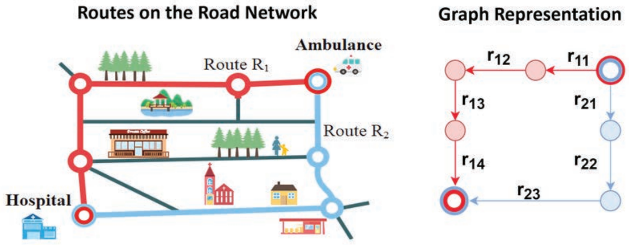


Figure 1: Example of shortest routes to the hospital.

Table 1: Arrival time and road segments speed distributions.

(a) Arrival time distributions

Arrival time (mins)	Probability R1	Probability R2
20	0.4	0.3
30	0.3	0.3
40	0.1	0.4
50	0.2	0.0
Expected Time	31	31

(b) Speed distributions of route R_1

Road segment	Speed range (m/s) 0-10	10-20	20-30	30-40
r_{11}	0.4	0.3	0.2	0.1
r_{12}	0.5	0.3	0.2	0.0
r_{13}	0.2	0.2	0.3	0.3
r_{14}	0.2	0.2	0.2	0.4

1 INTRODUCTION

Optimal routing algorithms can improve the efficiency of public transport, reduce costs, and prevent accidents in road networks. Current navigation systems generate a route between an origin and destination point based on the least expected travel time [17]. However, only considering the expected travel time is often unreliable, as it fails to capture the uncertainty of the travel duration and the traveler's delay tolerance [30]. The example in Figure 1 illustrates the situation where an ambulance needs the best route to the hospital. Here, R_1 and R_2 have the same expected arrival time, i.e., 31 minutes, but different arrival time distributions, as shown in Table 1a. Route R_1 offers the shortest travel duration with the highest probability, i.e., 20 mins with 0.4 chance, but there is also a 0.2 chance the travel duration will be the longest, i.e., 50 mins. In contrast, route R_2 offers a lower probability to be the fastest but guarantees the travel takes at most 40 mins. If the ambulance has to arrive within 40 minutes to guarantee the patient's life, route R_2 is the best option. Such a high-resolution routing service would enable the end-users to make better decisions.

High-resolution routing services often rely on time-dependent stochastic traffic data representing the speed distributions of vehicles in a road network [11, 25, 30]. Specifically, a speed distribution captures the probability that a vehicle travels at a certain speed on a road segment. Table 1b shows an example of the speed distributions for each road segment of R_1. Here, the speed distribution of r_{11} indicates that vehicles traveling in this segment are in the $[0 - 10)$ (m/s) speed range with 0.4 probability, in the $[10 - 20)$ (m/s) speed range with 0.3 probability, and so on. Given the speed distribution and distance of each road segment, we can derive the arrival time distribution of each route, as shown in Table 1a.

To construct traffic speed distributions for road segments, we can use vehicle tracking data obtained from GPS and loop detectors. However, data collected from these sensors are often incomplete due to technical reasons such as loop detectors malfunctioning or temporarily unavailable GPS data. This *data sparseness* problem [11] prevents the construction of time-dependent stochastic speeds for each road segment.

The present paper aims to alleviate this problem by designing a deep learning model to accurately estimate the missing speed distributions in a road network. However, achieving a highly accurate estimation model is a challenging task as traffic speeds in a road network exhibit complex spatial correlations between road segments and temporal correlations between time intervals. Specifically, *spatial correlations* occur since traffic conditions at nearby locations, either in consecutive or bidirectional road segments, impact each other. For instance, in Figure 1, the traffic flow in road segment r_{11} impacts the flow in r_{12}. In turn, traffic speeds between consecutive time intervals are *temporally correlated* as the vehicle speeds traveling on a road segment at time interval $t + 1$ are likely to be similar to the speeds at time interval t.

Several prior studies have attempted to estimate missing traffic speed values in a road network. However, they either fail to capture both spatial and temporal correlations, e.g., [11] only considers spatial correlations, or focus solely on estimating the average speed values rather than stochastic speeds [24, 29].

Contributions. This paper proposes a Stochastic Spatio-Temporal Graph Convolutional Network (SST-GCN) model that accurately estimates the missing speed distributions in a road network. Our key contributions are the following. (1) We design the SST-GCN framework that automatically captures both spatial and temporal correlations in a road network. Our framework integrates Graph Convolutional Networks (GCNs) to capture spatial correlations among road segments and Temporal Convolutional Networks (TCNs) to capture temporal correlations among time intervals in a graph-based road network model. (2) To cope with datasets with many missing values, we propose a novel self-adaptive context-aware GCN that prioritizes the propagation of the observed information in the graph while estimating the missing values. Furthermore, we show that our context-aware GCN is generic and thus, can be applied to any graph-based structure to learn context-aware spatial correlations efficiently. Experimental results show that our context-aware GCN improves the accuracy of SST-GCN by 3% on average. (3) We conduct an extensive experimental evaluation on real-world datasets showing that SST-GCN outperforms the baselines on estimation accuracy. In particular, SST-GCN has from 36% to 50% higher accuracy than the state-of-the-art stochastic speed estimation baseline [11]. We also compare against modified *deterministic* state-of-the-art baselines, achieving from 4.6% to 34%, and from 12% to 71% accuracy improvement over the modified deep learning and non deep learning baselines, respectively. Additionally, we perform a comprehensive study to evaluate the resilience, efficiency, and scalability of SST-GCN on different missing value patterns and graph sizes. The code is available at https://github.com/cmcuza/sst-gcn.

Table 2: Traffic data imputation and forecasting methods.

	Deterministic	Stochastic
Spatial and Temporal Correlations	ICLR'18 [16], IJCAI'19 [24] SIGKDD'16 [6], '19 [7], '20 [8] TR_C'20 [4], AAAI'20 [29]	**SST-GCN** (this paper)
Only Spatial Correlations	AAAI'11 [12], TKDE'13 [27]	ICDE'19 [11]

2 RELATED WORK

Learning from spatio-temporal traffic data has been extensively studied in the literature, where the main research focus has been on forecasting future traffic speeds and imputing missing speed values. Two main groups of techniques exist: deep learning [1, 7, 16, 24, 29] and matrix factorization [3, 4, 6, 19]. While matrix factorization approaches have been commonly used, they are often transductive, i.e., applied to specific datasets, thereby difficult to generalize and extend to other datasets. Recently, deep learning approaches, mostly relying on Graph Convolutional Networks (GCNs), have been widely used to solve different tasks in the traffic domain. The GCNs popularity is due to their efficiency in capturing complex graph structures and finding dependencies among nodes and edges in road networks. Consequently, deep learning approaches are not only inductive but also provide highly accurate models for traffic forecasting and imputation. Table 2 shows the most representative work in both tasks classified in two categories: *deterministic* and *stochastic. Deterministic* methods estimate the *average* speeds on road segments, while *stochastic* ones estimate the stochastic speeds.

The majority of the existing deep learning models are *deterministic*, i.e., they forecast *average* traffic speeds for future time intervals. The *deterministic* category is further divided into *spatial and temporal correlations* and *only spatial correlations.* The former includes methods that capture both spatial and temporal correlations in road networks. For example, MTGNN [8] and GraphWaveNet [24] combine GCNs and Temporal Convolution Networks (TCNs) into a single framework. ST-MetaNet [7] proposes three stacked RNNs within a sequence-to-sequence architecture. GMAN [29] uses attention mechanisms to capture spatial correlations between nodes and temporal correlations between time intervals. However, all these methods require available data for all road segments and are thus unsuitable for sparse data. The latter subcategory contains methods that only consider spatial correlations. In particular, [12, 27] apply regression-based loss functions to minimize the difference between traffic speed values of adjacent nodes. However, reducing the dissimilarity between adjacent nodes alone is not sufficient to capture complex spatial correlations among road segments.

Although there is no straightforward way to extend *deterministic* deep learning methods, they can be adapted to impute speed distributions instead of only average values. We select representative deterministic methods, i.e., MTGNN [8], GraphWaveNet [24], ST-MetaNet [7], and GMAN [29], and modify them to work in our stochastic setting. Specifically, we change their input layers such that they accept stochastic speed values as input features, and the output layers so that they produce a valid speed distribution. We also adapt the non-deep learning general imputation and forecasting methods MICE [22] and VAR [20] to work in the stochastic setting. However, as we show in Section 6, these simple adaptations are not sufficient for the stochastic speed imputation problem, as they fail to capture the dependencies among different speed ranges and ignore the missing information in the road network.

As Table 2 shows, only GCWC [11] performs stochastic speed estimation. GCWC uses GCNs to encode the road network's topology and estimate missing speed distributions. To cope with graphs with many missing values, it applies a graph pooling layer that reduces the graph dimensionality. Compared with modified *deterministic* baselines such as [6], GCWC shows that GCNs are better at capturing complex spatial correlations. However, GCWC does not consider temporal correlations, preventing the model from achieving high accuracy. In SST-GCN, we overcome this limitation by introducing a temporal component based on TCNs to learn temporal correlations between consecutive time intervals. Furthermore, we propose a novel diffusion process to handle road networks with many missing values, thereby providing a more accurate solution for stochastic speed estimation.

3 PRELIMINARIES

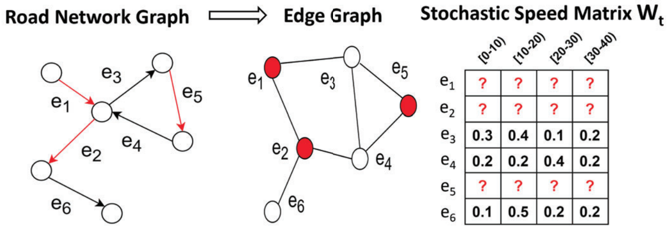
3.1 Stochastic Spatio-Temporal Traffic Data

Road Network (RN): A road network is a system of interconnected road segments. It is typically modeled as a directed graph $G_R = (V_R, E_R)$, where the edge set E_R represents road segments and the vertex set V_R represents road intersections. Each road segment (edge) has the vehicle's speed information. Since we aim to estimate the missing speed distributions on road segments, capturing the correlation between nearby road segments, especially those that share the same traffic flow is important. To achieve this, we transform the road network graph into an equivalent edge graph [11].

Edge Graph (EG): We obtain the *edge graph* $G = (V, E)$ by transforming the edges E_R of G_R into the vertices V of G, i.e., we transform each edge $e_i \in E_R$ into a vertex $v_i \in V$, so that $V = E_R$. We add an edge between vertices $v_i, v_j \in V$ if it is possible to travel from edge $e_i \in E_R$ to $e_j \in E_R$ (or vice versa) via a single vertex $v \in V_R$. The EG can be represented by the adjacency matrix $A \in \{0,1\}^{N \times N}$ where $N = |V|$ s.t. $A_{i,j} = A_{j,i} = 1$ if there is an edge between vertices v_i and v_j, and $A_{i,j} = 0$ otherwise.

Like in prior work [11], the edge graph is undirected, allowing to capture the correlation between two consecutive road segments regardless of the traffic flow direction. This is intuitive, as traffic flows of consecutive road segments, even if they are one-way streets, always have an impact on each other in both directions. Figure 2 shows a road network (left) and its equivalent edge graph (right). The EG has an edge between nodes e_1 and e_2 because we can travel from edge e_1 to e_2 in the RN via a single vertex. The red edges in the RN indicate road segments with missing speed distributions. They are correspondingly transformed into the red nodes in EG.

Notation convention: Hereafter, we use the notation v_i to denote a node in an EG in formal definitions and equations, but use e_i to denote a node in an EG when giving examples to be consistent with the example in Figure 2.



	[0-10)	[10-20)	[20-30)	[30-40)
e_1	?	?	?	?
e_2	?	?	?	?
e_3	0.3	0.4	0.1	0.2
e_4	0.2	0.2	0.4	0.2
e_5	?	?	?	?
e_6	0.1	0.5	0.2	0.2

Figure 2: A road network graph, its edge graph, and the SSM.

Speed Histogram: The speed distribution of a node in the EG is represented as a histogram of the speed values in the form $\{(b_i, p_i)\}$, where $b_i = [l_i, u_i)$ is the speed range, and p_i is the probability that vehicle speeds fall into that range. Since we fix the histogram's intervals, there is no need to explicitly store the speed ranges, which allows to simply store the probabilities in a vector. This vector is used as a graph signal on the nodes of the EG.

Stochastic Speed Matrix (SSM): Given an edge graph, a stochastic speed matrix is a matrix representation of all graph signals at a specific time interval t, denoted as $W_t \in \mathbb{R}^{N \times M}$ where N is the number of nodes, M is the number of speed ranges (or buckets) in the speed histogram, and t is the time interval during which the vehicle speeds are measured. Each row W_{t,v_i} stores the vector representation of the speed distribution on node v_i at time interval t. To construct W_t, we first divide the observation temporal granularity into equal-size time intervals. For example, we divide one day (i.e., 24 hours) into 96 time intervals of 15 minutes each. Then, for each time interval t and each node $v_i \in V$, we construct a speed histogram from vehicle speeds that pass through v_i during t. An example SSM is shown in Figure 2.

We initialize the missing value of node v_i at time interval t using the average stochastic speed, which is computed using all the observed speed values of v_i at t from other days in the training set. Furthermore, for each time interval t, we subdivide the nodes in V into two groups: V_o containing nodes with observed speed distributions, and V_m containing nodes with missing distributions. For example, in Figure 2 and at time interval t, we have $V_o = \{e_3, e_4, e_6\}$ and $V_m = \{e_1, e_2, e_5\}$. Next, we formulate the stochastic speed estimation problem that aims to reconstruct the missing distributions of the nodes in V_m using the information in V_o.

3.2 Stochastic Speed Estimation

Given a road network graph $G_R = (V_R, E_R)$, let $G = (V, E)$, W_t, and V_m be the corresponding edge graph, stochastic speed matrix, and sets of nodes with missing speed distributions, respectively. The stochastic speed estimation problem aims to estimate a new matrix $\hat{W}_t \in \mathbb{R}^{N \times M}$ such that the missing values in V_m at time interval t are replaced by stochastic values that are as close as possible to the ground truth in W_{G_t}. This is equivalent to learning an estimation function $g(\cdot)$ that minimizes the distance between the estimated SSM $\hat{W}_t$ and the ground truth SSM W_{G_t}, i.e.

$$\begin{gathered}\underset{\hat{W}_t}{\operatorname{argmin}}\ \mathbf{d}(\hat{W}_t, W_{G_t}) \text{ where:} \\ \hat{W}_t \leftarrow g([W_{t-T}, W_{t-T+1}, ..., W_{t-1}, W_t]; G)\end{gathered} \tag{1}$$

In Eq. (1), the estimation function $g(\cdot)$ takes as input the array of the SSMs $[W_{t-T}, ..., W_t]$ associated with T past time intervals from t and the edge graph G to estimate $\hat{W}_t$. To obtain an accurate estimation of the missing stochastic speed, $g(\cdot)$ should capture the temporal correlations of the speeds in consecutive time intervals and the spatial correlations of neighboring nodes in G.

4 STOCHASTIC SPATIO-TEMPORAL GRAPH CONVOLUTIONAL NETWORK

4.1 Architecture Overview

The Stochastic Spatio-Temporal Graph Convolutional Network can be represented as an *Encoder-Decoder* model, illustrated in Figure 3. The *Encoder* transforms the input matrix W_t into a high-dimensional tensor $\mathbb{L} \in \mathbb{R}^{N \times M \times D}$, where each probability value $W_{t,v_i,z}$ of the speed range z at node v_i and time interval t is transformed into a feature vector of size D. These feature vectors are used by the *Decoder* to estimate the missing speed distributions.

Input: The model receives 3 main inputs: (1) The list of *stochastic speed matrices* $\mathbf{W} = [W_{t-T}, ..., W_t] \in \mathbb{R}^{T \times N \times M}$, where each W_{t-i} represents the SSM at time interval $(t-i)$, $i \in [0, ..., T]$. (2) The *adjacency matrix* $A \in \{0,1\}^{N \times N}$ representing the topology of the edge graph. (3) The vector $c_t \in \{0,1\}^N$ that provides the *context information* in the edge graph: $c_t[i] = 1$ if the speed distribution of node v_i at time interval t is available, and $c_t[i] = 0$ otherwise. For instance, in the example of Figure 2, $c_t = [0,0,1,1,0,1]$ indicates that the speed distributions of nodes e_3, e_4 and e_6 are available, while those of e_1, e_2 and e_5 are missing.

Encoder: This component learns a high-dimensional representation of the SSM W_t where each probability $W_{t,v_i,z}$ is represented as a feature vector that captures spatio-temporal correlations. To capture the spatio-temporal correlations among nodes in the edge graph, the *Encoder* uses *convolutional filters* to learn how the speed distribution of node v_i at time interval t is influenced by the speed distribution of its neighboring nodes $\mathcal{N}(v_i)$, and by the speed distribution of node v_i itself at the past time intervals at $(t-1)$, $(t-2)$, ..., $(t-T)$. Intuitively, the probability a vehicle travels at a certain speed range at node v_i is *spatially correlated* with the probability that the vehicle travels at that same speed in the neighboring nodes. The same intuition applies for temporal correlation, where the vehicle's speed at time interval t is *temporally correlated* with its speed in the preceding time intervals. Overall, the *Encoder* consists of: (1) A *Linear Transformation* layer that projects the speed distributions into a high-dimensional space. (2) Multiple spatio-temporal blocks (*ST-Blocks*) that combine a Temporal Convolutional Network (TCN) and a Context-Aware Graph Convolutional Network (CGCN) to jointly learn temporal and spatial correlations.

Decoder: The last component of the architecture is the *Decoder*, which receives the tensor $\mathbb{L} \in \mathbb{R}^{N \times M \times D}$ as input, and generates the estimated SSM $\hat{W}_t$ as output where the missing speed distributions are replaced with newly estimated distributions. This is achieved by applying two *Feed Forward Neural Networks* and a *Softmax* layer to decode the feature vectors into valid probability distributions.

4.2 Linear Transformation

The *Linear Transformation* (LT) is the first layer in the *Encoder*. It projects the list of stochastic speed matrices $\mathbf{W} = [W_{t-T}, ..., W_t]$ into a high-dimensional space. Specifically, it transforms each probability value in the SSM W_t into a new representation $H^0_{t,v,z} = W_{t,v,z} \cdot \psi$, where t is the considered time interval, v is a node in V, z is the considered speed range, and $\psi \in \mathbb{R}^D$ is a shared vector of learnable parameters. Applying this transformation to each SSM in $\mathbf{W}$ outputs a new hidden representation $H^0 = [H^0_{t-T}, H^0_{t-T+1}, ..., H^0_t]$ that is called the stochastic feature vector. Stochastic feature vectors serve as the input for the *ST-Block* and can be understood as scaling up each speed range's probability into a high-dimensional space to differentiate each speed range better.

4.3 Spatio-Temporal Block

The *Spatio-Temporal Block* (ST-Block) applies Temporal Convolutional Networks and Context-Aware Graph Convolutional Networks to learn the temporal and spatial correlations in an edge graph, thereby enriching the feature representation H^0 obtained from the *Linear Transformation* layer. It does so by first splitting H^0 to obtain the feature vectors for each speed range z separately. Then, for each range z, it feeds the selected feature vectors to the respective TCNs and CGCNs layers, e.g., z_1,..., z_4 in Figure 3, and explicitly learns the feature representation for each speed range. Section 5 describes in detail the TCNs and CGCNs components.

To split the tensor H^0, a slicing operation is applied to obtain the feature vectors $H^0_{\because z} \in \mathbb{R}^{T \times N \times D}$, $z \in [1, ..., M]$. The feature vectors $H^0_{\because z}$ are then used as the initial input for the *ST-Block*. Through the splitting, the dimensionality of H^0 is also reduced which in turn helps optimize the convolutional filter learning. In addition, the *ST-Block* applies a residual connection that helps the network focus only on the missing values by learning the function $h(x) = g(x) + x$, where $+x$ brings back the original values and $g(x)$ learns how to estimate the missing values. Next, Dropout and Batch Normalization (BN) is applied to avoid overfitting. After applying multiple *ST-Blocks* in the Encoder, we obtain the tensor $\mathbb{L} \in \mathbb{R}^{N \times M \times D}$ with the spatio-temporal information encoded in the feature representation of each speed range.

4.4 Stochastic Speed Matrix Generation

The high-dimensional tensor $\mathbb{L} \in \mathbb{R}^{N \times M \times D}$ output by the *Encoder* is sent to the *Decoder* to generate the final SSM $\hat{W}_t$. Before estimating $\hat{W}_t$, we take another step to capture the dependencies between different speed ranges. This is because consecutive speed ranges in a speed histogram also exhibit a certain degree of correlation, i.e., the probabilities of two consecutive speed ranges can be highly positively correlated, while the probabilities of speed ranges that are further apart are typically negatively correlated. For example, a vehicle traveling between [10-20) (m/s) is likely to move to [20-30) (m/s), and thus, the probabilities of [10-20) and [20-30) speed ranges are more positively correlated. In contrast, the vehicle is unlikely to move from [10-20) to [30-40) (m/s), and thus, the probabilities of [10-20) and [30-40) speed ranges are more negatively correlated.

To capture this type of correlation, we transform the tensor $\mathbb{L}$ by concatenating the feature representation of each histogram bucket and obtain a combined feature representation for each node v_i as

$$\hat{\mathbb{L}}_{v_i} = [L_{v_i 1}, L_{v_i 2}, ..., L_{v_i M}] = \Big|\Big|_{z=1}^{M} \{\mathbb{L}_{v_i z}\} \tag{2}$$

where $L_{v_i z} \in \mathbb{R}^D$ represents the hidden representation learned for the z-th speed range on node v_i, and $||$ represents the concatenation

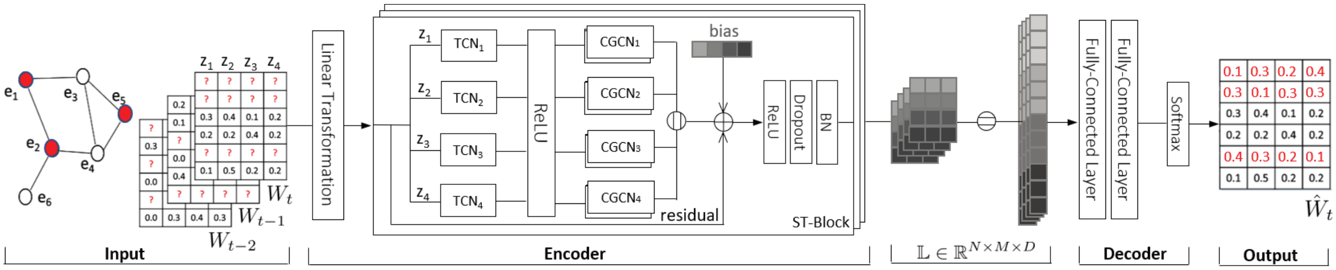


Figure 3: General architecture of our framework given an input example.

operator. Eq. (2) can be seen as the inverse of the previously described slicing operation. Once $\hat{\mathbb{L}} = [L_{v_1}, L_{v_2}, ..., L_{v_N}] \in \mathbb{R}^{N\times(M*D)}$ is obtained, we apply two Fully Connected Layers (FCs) that consider all speed ranges at once to finally obtain the intermediate matrix $Z \in \mathbb{R}^{N\times M}$ defined as $Z = \sigma(\hat{\mathbb{L}}W_{\text{fc1}} + b_{\text{fc1}})W_{\text{fc2}} + b_{\text{fc2}}$, where $W_{\text{fc1}} \in \mathbb{R}^{(M*D)\times(M*D/2)}$ and $W_{\text{fc2}} \in \mathbb{R}^{(M*D/2)\times M}$ are the parameter matrices and $[b_{\text{fc1}}, b_{\text{fc2}}]$ are the bias vectors.

Finally, to generate the estimated SSM $\hat{W}_t$, the *Softmax* function is applied to each row of the matrix Z to obtain the probability distribution for each node v_i at time interval t. The *Softmax* layer ensures that the obtained distribution is valid, i.e., each value $\hat{w}_{ij} \in \hat{W}$ is between $[0, 1]$, and $\sum_{j=1}^{M} \hat{w}_{ij} = 1$ with $i \in [1, N]$.

4.5 Loss Function

The loss function measures the distance between the estimated SSM $\hat{W}_t$ and the ground truth SSM W_{G_t} using the Kullback–Leibler (KL) divergence [15], which is defined as

$$\text{KL}(w||\hat{w}) = \sum_{j=1}^{M} \hat{w}_j \cdot \log(\frac{\hat{w}_j + \epsilon}{w_j + \epsilon}) \tag{3}$$

where w_j and $\hat{w}_j$ are the actual and estimated probabilities at the j-th histogram bucket. The small positive number ϵ in Eq. (3) prevents a division by zero. Finally, the general loss function is obtained by applying Eq. (3) to all graph nodes:

$$L(W_{G_t}, \hat{W}_t) = \sum_{i=1}^{N} I_i \cdot \text{KL}(w_{i\cdot}||\hat{w_{i\cdot}}) \tag{4}$$

where $I \in \{0, 1\}^N$ is an indicator vector with $I_i = 1$ if the i-th node is covered by the traffic data in the considered time interval, and $I_i = 0$ otherwise. This indicator vector I allows estimating the error only for nodes with available ground truth.

In summary, the main novelty of our SST-GCN architecture lies in two key features: (1) the explicit learning of spatio-temporal correlations for each speed range in an SSM, and (2) the use of the context vector c_t that distinguishes between nodes with observed speed distributions and nodes with missing ones. Both these features are combined into a single stochastic framework to estimate the missing speed distributions without using any annotated data.

5 SPATIO-TEMPORAL CONVOLUTIONAL NETWORK

Our ST-Block learns two different kinds of convolutional kernels to capture spatio-temporal correlations in the edge graph. Specifically, we use a modified Graph Convolutional Network to learn spatial correlations, and a Temporal Convolutional Network to learn temporal correlations between edges. We describe them below.

5.1 Temporal Convolutional Network

The Temporal Convolutional Network applies Dilated Convolutional Networks (DCNs) [28] to learn temporal correlations between the current time interval t and its preceding T time intervals. To capture this temporal dependency, DCNs compute the dot product (i.e., the convolution operation) between the feature vectors of Q different time intervals, and the parameters vector ϕ of size Q (ϕ is also called a convolutional filter and is updated during training). Compared to 1D-CNNs, DCNs can capture temporal dependencies of more preceding time intervals using fewer hidden layers thanks to the dilation factor l. Formally, given a feature representation $H^{v}_{\cdot\cdot z}$ of the speed range z (or the z-th histogram’s bucket), the temporal convolution operation is defined as

$$H^{v}_{\cdot vz} *_l \phi^{v} = \sum_{i=0}^{Q} H^{v}_{\cdot vz}(t - l \cdot i) \otimes \phi^{v}(i) \tag{5}$$

where l is the dilation factor, and $H^{v}_{\cdot vz}(t - l \cdot i)$ is the feature vector of the speed range z at node v at the preceding $(t - l \cdot i)$-th time interval. Besides, $\phi^{v} \in \mathbb{R}^{Q\times D}$ is the learnable convolutional filter that is independent of the size of the edge graph and different for each speed range z. Eq. (5) is computed independently (i.e., in parallel) for each node v avoiding memory overflow and enabling efficient training as N increases. Finally, $*_l$ and $\otimes$ denote the dilated convolution operation and the dot product, respectively.

Similar to traditional convolutional neural networks, we apply multiple convolutional filters $\Phi^{v} = \{\phi^{v}_1, \phi^{v}_2, ..., \phi^{v}_D\}$ to generate the output of each TCN as

$$H^{v+1}_{\cdot\cdot z} = ReLU(\bigg|\bigg|_{j=1}^{D} H^{v}_{\cdot\cdot z} *_l \phi^{v}_j) \tag{6}$$

where $H^{v+1}_{\cdot\cdot z} \in \mathbb{R}^{T^{v+1}\times N\times D}$ is the new feature representation of the speed range z for all nodes and time intervals obtained at the $(v+1)$-th *ST-Block*, $\phi^{v}_j \in \Phi^{v}$, and $||$ denotes the concatenation of the results obtained from applying D convolutional filters. Compared with TCNs used in other methods [24], we learn a different set of convolutional kernel parameters for each histogram bucket.

5.2 Context-Aware GCN

We propose a novel Context-Aware Graph Convolutional Network (CGCN) to learn the spatial correlations between nodes in an edge graph, considering the context around them. Typically, a standard Graph Convolutional Network (GCN) finds localized patterns around a given node v_i by aggregating information from its neighboring nodes $\mathbf{v} = \mathcal{N}(v_i)$. However, using a standard GCN poses a challenge in road networks with many nodes with missing information. Aggregating information from such nodes could propagate nonsensical information around the network, resulting in an incorrect estimation. Replacing the missing information with the average values, as we did in the initial phase (Section 3), is still insufficient, as those do not represent the true state of the road network.

As an example, consider node e_3 of the edge graph in Figure 2 which has a speed distribution constructed from the available historical data. The neighboring nodes of e_3 are $\mathcal{N}(e_3) = \{e_1, e_4, e_5\}$, in which e_1 and e_5 have missing information and thus, their speed distributions are initialized with the average stochastic values. Using a standard GCN, the features of e_3 are combined with the aggregated information of neighboring nodes, resulting in the loss of the true speed distribution of node e_3. Specifically, during the training, the new vector representation of e_3 obtained from the aggregation will converge to the average speed distribution, which is the naive estimation initially assigned to unobserved neighboring nodes.

Context-Aware Adjacency Matrix: To address this problem, we propose a novel aggregation function that allows only the observed node's information to be propagated around the network. We achieve this by using a context vector $c_t \in \{0, 1\}^N$ to distinguish between observed and unobserved nodes: $c_t[i] = 1$ if the node v_i is observed and its speed distribution is available, and $c_t[i] = 0$ otherwise. Given the context vector c_t, we modify the topology of the edge graph through its adjacency matrix to ensure that only the observed nodes can propagate information to their neighbors. This new adjacency matrix is called the *context-aware adjacency matrix* A_c, and is defined as

$$A_c = D_c A + I \tag{7}$$

where D_c is the diagonal matrix obtained from c_t, A is the adjacency matrix of the original edge graph, and I is the identity matrix.

Context-Aware Aggregation: Using the context-aware adjacency matrix A_c, we can define an aggregation function that only combines information from the observed nodes as

$$H^{\nu+1}_{\cdot vz} = \frac{1}{d^{in}_v}(H^{\nu}_{\cdot vz} + \sum_{u \in \mathcal{N}_{A_c}(v)} H^{\nu}_{\cdot uz}) = \hat{P}^{\mathbf{T}}(v) H^{\nu}_{\cdot\cdot z} \tag{8}$$

where d^{in}_v is the in-degree of node v (i.e., the number of in-going edges of v), $\mathcal{N}_{A_c}(v)$ is the set of neighboring nodes of v in A_c, and $\hat{P}^{\mathbf{T}}(v)$ is the transpose of $\hat{P}(v)$, returning only the row associated with node v in $\hat{P}^{\mathbf{T}}$. In Eq. 8, the product of the row vector $\hat{P}^{\mathbf{T}}(v) \in R^{1\times N}$ and $H^{\nu}_{t\cdot z} \in R^{N\times D}$ is applied for each time interval t in parallel. Moreover, the transition matrix $\hat{P} \in \mathbb{R}^{N\times N}$ is computed as

$$\hat{P} = \frac{1}{\text{colsum}(A_c)} A_c \tag{9}$$

where colsum(A_c) performs the sum through the columns in A_c to compute the in-degree of the node as a normalizing factor. Intuitively, given a node v in an edge graph, the Eq. (8) updates the feature representation of v by computing the normalized linear combination of the feature vectors from observed neighboring nodes $\sum_{u\in\mathcal{N}_{A_c}(v)} H_{\cdot uz}$, and of the central node itself $H_{\cdot vz}$. This simple, but effective, change in the diffusion process filters out unobserved information when estimating the missing values.

Self-Adaptive Context-Aware Diffusion: To capture the spatial correlation among nodes at k-hop distance, we can apply the aggregation function in Eq. (8) successively k times. However, this can result in a situation where the observed information does not reach distant nodes because unobserved nodes in the transition matrix are *absorbing* information from their neighbors without propagating it. Thus, a pair of nodes at k-hop distance cannot share information if there are unobserved nodes in the path connecting them. We address this problem by making the context vector c_t and the respective context-aware adjacency matrix A_c self-adaptive, i.e., they automatically update after each aggregation. Specifically, we keep track of nodes that receive information in each aggregation and update the context vector c_t. The new context vector c'_t will have the value 1 for all observed nodes, and also the unobserved nodes which have just received information after the aggregation. Once the new vector c'_t is obtained, we apply Eq. (7) to update A_c, and use the updated A_c to obtain the new transition matrix $\hat{P}'$. This process is repeated for every aggregation. Let $\hat{P}_k$ be the transition matrix obtained after k aggregations. We define $\hat{P}_k$ recursively as

$$\hat{P}_k = \hat{P}_{k-1}\hat{P}' \tag{10}$$

where $\hat{P}_{k-1}$ is the transition matrix obtained after $(k-1)$ aggregations, and $\hat{P}'$ is the newly updated transition matrix with the new context vector c'_t. From $\hat{P}_k$, we can observe how information is aggregated among nodes within k-hop distance. For instance, applying the second aggregation on node e_2 renders:

$$H_{\cdot e_2 z} = \hat{P}^{\mathbf{T}}_2(e_2) H_{\cdot\cdot z} \approx [\frac{1}{8}, \frac{2}{25}, \frac{1}{4}, \frac{1}{5}, 0.0, \frac{1}{3}] \otimes H_{\cdot\cdot z} \tag{11}$$

Here, the vector $[\frac{1}{8}, \frac{2}{25}, \frac{1}{4}, \frac{1}{5}, 0.0, \frac{1}{3}]$ shows the weights of nodes within a 2-hop distance when transmitting their information to e_2. Note that e_2 is receiving information from all nodes, except from node e_5. This is because e_5 is an unobserved node, and does not propagate information in the first aggregation. Thus, the information from e_5 will only reach e_2 after the third aggregation. In contrast, the unobserved node e_1 already sends information to e_2 in the second aggregation since they are direct neighbors. Moreover, the weight of e_1 is $\frac{1}{8}$, less than the weights $\frac{1}{4}$, $\frac{1}{5}$, and $\frac{1}{3}$ of observed ones e_3, e_4, and e_6 respectively. This indicates that the context-aware diffusion process puts more weight on observed nodes, i.e., making them more *important* than unobserved ones. We further analyze this behavior in section 5.2. Finally, given an upper bound K, we define the K-hop distance convolution operation as

$$H^{\nu}_{\cdot\cdot z} = ReLU(\sum_{k=1}^{K} \hat{P}^{\mathbf{T}}_k H^{\nu}_{\cdot\cdot z} \varphi^{\nu}_k) \tag{12}$$

where $\varphi^{\nu}_k \in \mathbb{R}^{D\times D}$ is the learnable convolution filter to capture spatial correlation among k-hop distance nodes. The operation updates the feature representation in the ν-th ST-Block.

Context-Aware GCN Generalization: As discussed in Eq. (11), the context-aware diffusion process puts more weight on observed nodes than unobserved ones, e.g., e_4 has higher weight than e_1. Without the context vector, a standard aggregation function will assign a higher weight on e_1 than on e_4 since it is closer to e_2. In general, without considering the context information, one can expect that within a k-hop distance, node weights only depend on the number of possible paths through which they can reach the central node after k steps. However, this behavior is not desirable for estimating missing values, since it ignores the validity of the information held by the neighboring nodes. Instead, it is better to consider both the number of paths and the information validity of the neighboring nodes when assigning weights to them. Below, we show that the context-aware diffusion process can be applied to any graph-based structure to obtain better weight assignments.

Consider an edge graph $G = (V, E)$, and a GCN diffusion process $\mathcal{P}$ applied on G. Let $c_t \in \{0, 1\}^N$ be the context vector of G, representing the observable state of each node $v \in V$: $c_t[v] = 1$ if v is observed, and $c_t[v] = 0$ otherwise. We consider two scenarios: (1) the diffusion process $\mathcal{P}$ applies the context vector c_t during its aggregation, denoted as $\mathcal{P}_C$, and (2) $\mathcal{P}$ does not use the context vector c_t but only a standard aggregation function, denoted as $\mathcal{P}_{\overline{C}}$.

Lemma 1. *Within a K-hop distance in G, let $\hat{P}_K$ and P_K be the transition matrix obtained after K aggregations from the context-aware diffusion process $\mathcal{P}_C$ and the standard diffusion process $\mathcal{P}_{\overline{C}}$, respectively. Then, for any node $u \in \mathcal{N}_K(v)$ where $\mathcal{N}_K(v)$ represents the set of neighboring nodes of v within the K-hop distance, we have:*

$$\hat{\boldsymbol{P}}_{\boldsymbol{K}}[\boldsymbol{u};\boldsymbol{v}] \geq \boldsymbol{P}_{\boldsymbol{K}}[\boldsymbol{u};\boldsymbol{v}],\ \forall u \in \mathcal{N}_K(v) \ s.t.\ c_t[u] = 1 \tag{13}$$

where $\hat{P}_K[u;v]$ ($P_K[u;v]$) returns the weight associated with the observed neighboring node u of the central node v, representing the fraction of information node u is transmitting to v using $\mathcal{P}_C$ ($\mathcal{P}_{\overline{C}}$).

Lemma 1 states that the context-aware diffusion process places more importance (i.e., higher weights) on observed nodes than the standard diffusion process (see proof in Appendix A.1). The experiments in Section 6 show that this better captures the spatial correlations between nodes, improving the accuracy of SST-GCN by 3% on average.

6 EXPERIMENTAL EVALUATION

6.1 Experimental Setup

Datasets: We use two real-world traffic datasets: Highway Tollgates Network [14], and City Road Network [10]. The *Highway Tollgates Network* (HT) dataset records vehicle speeds using loop detectors deployed on 24 roads of a highway in China. During preprocessing, we partition each day into 96 15-min intervals. The dataset has a speed range of $[0 - 40)$ (m/s), which we divide into 4 equal-width buckets, i.e., $[0-10)$, $[10-20)$, $[20-30)$, $[30-40)$, following [11]. We set the number of buckets $M = 4$ after trying different values as it generates histograms where each bucket has a non-zero probability. For $M = 8$, we obtain histograms with zero probability in many buckets. We set the number of preceding time intervals $T = 3$, i.e., we look back 45 minutes.

The *City Road Network* (CR) dataset contains 3.01 billion GPS records produced by taxis in Chengdu, China during 5 representative time horizons: 3:00–5:00, 8:00–10:00, 12:00–14:00, 17:00–19:00, and 21:00–23:00, which involve rush, normal, and night hours. The data granularity is 2 mins, so we divide a day into 72 20-min intervals. Similar to HT, we set $M = 4$ and $T = 3$, i.e, we look back 1 hour. Note that the time horizons limit the number of past time intervals we can consider. The CR graph has 1902 nodes and 5943 directed links, so we extract two subgraphs using the Quasi-Cliques [21] dense subgraph extractor. The extractor uses a parameter α to control the degree of density in the subgraph. Modifying α, we obtain one subgraph with 173 nodes ($\alpha = 0.02$) and one with $1,026$ nodes ($\alpha = 0.004$). We note that the largest road network used in all baselines contains at most 1024 nodes. Therefore, we chose $\alpha = 0.004$ to obtain a similar graph size, while $\alpha = 0.02$ gives us a similar graph size as the one used in our closest baseline [11]. Table 3 summarizes the main characteristics of the edge graphs.

Table 3: Main characteristics of the edge graphs.

Dataset	#Samples	Time Interval	Nodes	Edges	Average Degree
HT	3093	15	24	24	2.0
CR_{173}	675	20	173	904	5.2
CR_{1026}	675	20	1026	6558	5.4

Ground Truth and Input Data: We construct the ground truth matrix W_{G_t} from the available traffic data. Specifically, we construct the speed histogram for nodes having at least 5 speed records and compute the historical average speed distribution for the remaining nodes. To simulate nodes with missing information, we randomly select a subset of nodes from the edge graph and remove their true speed distributions. This allows us to evaluate the accuracy of SST-GCN by comparing the estimated values with the ground truth. We control the amount of missing-information nodes using a ratio $\rho = |V_m|/N$, where $|V_m|$ is the number of nodes with missing speed distributions and N is the total number of nodes in the edge graph. We conduct experiments with $\rho \in \{0.5, 0.6, 0.7, 0.8\}$. When $\rho = 0.5$ ($\rho = 0.8$), 50% (20%) of the nodes with ground truth are used to estimate the other 50% (80%) with missing information.

6.2 Baselines and Training Configurations

We compare our approach with the following baselines: (1) *Graph Multi-Attention Network* **(GMAN)** [29] uses an encoder-decoder model with multi-head transformer-based attention to forecast the *average* speed. (2) *Temporal Convolutional Network* **(TCN)** uses TCNs [23] to encode the temporal correlations, and two Feed Forwards Networks to forecast the *average* speed. (3) *Graph MetaNet* (**ST-MetaNet**)[7] applies three stacked RNNs combined with an attention mechanism and meta-knowledge to forecast the *average* speed. (4) *Graph Wavenet* (**GWaveNet**) [24] combines TCNs and GCNs to learn spatio-temporal correlations in a road network and forecasts the *average* speed. (5) *Multivariate Time Series Graph Neural Network* (**MTGNN**) [8] combines TCNs and GCNs to forecast future values of the target time series. We adapt the above *deterministic* baselines to estimate the stochastic speeds as discussed in Section 2. (6) *Graph Convolutional Weight Completion* (**GCWC**) [11] uses GCNs to capture spatial dependencies between the stochastic speed values and estimate the missing *speed distributions*. (7) **SST-GCN$_{\mathbf{mv}}$** is a modification of our SST-GCN that estimates the *mean* and *variance* of the speed distribution. To compare it with the other baselines, we estimate the probability for each histogram bucket from the obtained mean and variance using the Cumulative Density Function of the Gaussian distribution. (8) *Vector Autoregression* **(VAR)** [20] is a non-deep learning method that extends the univariate auto-regressive model to multivariate time series. (9) **MICE** [22] is a widely used non-deep learning method that imputes incomplete multivariate data by chained equations. We use MICE and VAR to estimate the missing values for each histogram bucket independently. For example, if we have 4 buckets, we use 4 different VAR/MICE models to estimate a value for each bucket. Finally, we normalize the values to sum one.

Training: We use *5-fold cross-validation* in all experiments: every run uses 4 folds for training and validating, and 1 fold for testing. We use the average results in the test set for comparison between the baselines and our method. We found that the best values of hyper-parameter K, number of kernels, learning rate and dropout for our method do not change much per dataset. In particular, $K = 2$ for HT and CR_{173}, while $K = 3$ for CR_{1026}. The number of kernels is 32 for all the datasets, and the dropout is either 0.05 or 0.1 for HT or CRs datasets. The best learning rate is 0.003 for HT and CRs. Appendix A.3 provides more detail about the hyper-parameters.

Hardware Configuration: The experiments are performed on a machine with an Intel Xeon CPU @2.50GHz and a Tesla V100-SXM3 32Gb GPU running Ubuntu 16.04.

6.3 Evaluation Metrics

To evaluate the accuracy of SST-GCN, we measure the distance between the ground truth SSM W_{G_t} and the estimated SSM $\hat{W}_t$. Since we estimate probability distributions, standard regression metrics such as MAE and MAPE are not applicable. Thus, we use three well-known distance metrics for probability distributions: Kullback-Leibler Divergence (KLD) [15], Jensen-Shannon Divergence (JSD) [9], and Earth Mover's Distance (EMD) [18]. A lower distance indicates a more accurate estimation. The KLD metric is defined in Eq. (3), whereas the details of the JSD and EMD metrics are provided in Appendix A.2.

Since KLD and EMD are unbounded metrics, i.e., their values range from 0 to ∞, we perform a normalization using the average distribution HA built from historical data to provide an easier interpretation. Here, we interpret $\mathrm{HA_j}$ as the worst estimation of the speed distribution for the j-th road segment [11]. We normalize all three metrics using HA as

$$\mathrm{D}_f = \frac{\sum_{i=1}^{\Omega} \sum_{j=1}^{N} I_{ij} \mathrm{f}(w_{j\cdot}^i || \hat{w}_{j\cdot}^i)}{\sum_{i=1}^{\Omega} \sum_{j=1}^{N} I_{ij} \mathrm{f}(w_{j\cdot}^i || \mathrm{HA}_{j\cdot})} \tag{14}$$

where f(.) is the distance metric, $w_{j\cdot}^i$ the ground truth distribution, $\hat{w}_{j\cdot}^i$ the estimated distribution, $\mathrm{HA}_{j\cdot}$ the average distribution, Ω the total number of time intervals, N the number of nodes in the edge graph, and I_{ij} is an indicator matrix. $\mathrm{D}_f < 1.0$ indicates that the estimated distribution is closer to the ground truth distribution than the average distribution, and $\mathrm{D}_f > 1.0$ indicates the opposite. Thus, a smaller D_f value indicates a more accurate estimation.

6.4 Accuracy Evaluation

Table 4 shows the accuracy comparison of **SST-GCN** with the baselines, using the normalized KLD, JSD, and EMD metrics (boldface denotes the best results). As seen, **SST-GCN** has the *highest accuracy* among all baselines and settings. Compared to GCWC, the only existing stochastic speed estimation baseline, **SST-GCN** improves the accuracy by **49.5%**, **50.22%** and **36.73%** on average for KLD, JSD, and EMD, respectively. This confirms our hypothesis that both spatial and temporal correlations are crucial factors for an accurate stochastic model, and by capturing both, we gain an advantage over GCWC that only considers spatial correlations. The other deep learning baselines also outperform GCWC, except for GMAN in HT. Considering that HT has only 24 nodes with an average degree of 2, we believe that the attention mechanism is not as effective as in CR where the average degree is 5. Other factors, such as a high missing information rate, could negatively affect the attention mechanism, making GMAN underperform compared to the rest of the baselines. Among all baselines originally designed to forecast average speeds, GWaveNet, after being modified to forecast stochastic values, achieves the closest accuracy to our **SST-GCN**. **SST-GCN** improves the accuracy by 4.6%, 7.7% and 5.5% on average over GWaveNet for KLD, JSD and EMD, respectively. In some cases, SST-GCN outperforms GWaveNet by a very high margin. For example, for HT, when $\rho = 0.7$ and 0.8, SST-GCN is up to 14.9% and 13% better than GWaveNet using the JSD metric. We note that although MTGNN outperforms GWavenet in forecasting tasks [8], this is not the case in our experiments. This is because MTGNN relies on a self-learning adjacency matrix, which does not perform as well in the presence of many nodes with missing values.

Compared to the mean-variance baseline SST-GCN$_{\mathrm{mv}}$, our solution has from 18% to 42% higher accuracy. This confirms previous studies [5, 26] showing that in practice, the traffic speed does not follow a Gaussian distribution. Therefore, to correctly model the real traffic behavior, it is necessary to use more precise representations, such as histograms, that can model arbitrary distributions. Compared to the rest of the baselines, **SST-GCN** has significantly higher accuracy with an average improvement from **11%** to **35%**. Note that GWaveNet, MTGNN, and ST-MetaNet have better results than TCN, since they can capture spatial correlations. Surprisingly, MICE performs very well in HT dataset, especially when using the EMD metric. However, our **SST-GCN** is still better than MICE when using JSD and KLD in HT and it is always much better in the CR datasets. These results suggest that MICE can only be applied in small road networks like HT. Overall, **SST-GCN** improves the accuracy over MICE by **45.6%**, **43.62%** and **12.38%** on average for KLD, JSD, and EMD, respectively. Finally, VAR has the worst performance of all.

Statistical significance test: To determine the statistical significance of our accuracy results, we conduct a *paired sample t-test* [2]. For all accuracy measures KLD, JSD, and EMD, we achieve a p-value less than 0.01, rejecting the null hypothesis that **SST-GCN** and the baselines have the same average performance. Thus, the accuracy improvement of SST-GCN is statistically significant. We attribute this improvement to the context-aware diffusion process and the explicit learning of spatio-temporal correlations for each speed range. In Sections 6.5 and 6.6 we evaluate further this hypothesis.

6.5 Context-Aware Diffusion Process Evaluation

To measure the impact of the context-aware diffusion process, we evaluate the accuracy of **SST-GCN** without the context information. We name this version SST-GCN$_{\mathrm{NC}}$ (NC stands for No Context). Furthermore, we modify GWaveNet (the best performing baseline) by incorporating the context-aware diffusion process, named GWaveNet$_{\mathrm{C}}$. Table 5 shows the results of these four methods for

Table 4: Accuracy evaluation of SST-GCN and the baselines. Boldface denotes the best results.

Metrics	Methods	*HT*				CR_{173}				CR_{1026}			
		0.5	0.6	0.7	0.8	0.5	0.6	0.7	0.8	0.5	0.6	0.7	0.8
D_{KLD}	VAR	0.8616	0.8562	0.8574	0.8593	0.8097	0.8470	0.8897	0.8994	1.1517	1.1318	1.1105	1.1224
	MICE	0.2135	0.3402	0.4785	0.6990	0.3948	0.4892	0.6080	0.7458	0.4267	0.5268	0.6276	0.7306
	GMAN	0.4161	0.4193	0.4936	0.6059	0.2742	0.3134	0.3539	0.3911	0.2563	0.3043	0.3447	0.3860
	GCWC	0.3948	0.4488	0.4964	0.5777	0.5183	0.5483	0.5788	0.6079	0.6123	0.6513	0.6607	0.6762
	TCN	0.2569	0.3559	0.4305	0.5325	0.2393	0.2797	0.3203	0.3559	0.2497	0.2954	0.3375	0.3750
	ST-MetaNet	0.2177	0.2909	0.3738	0.4993	0.2320	0.2798	0.3201	0.357	0.2339	0.2902	0.3254	0.3688
	MTGNN	0.2195	0.3709	0.4798	0.5896	0.2278	0.2679	0.3081	0.3450	0.2415	0.2807	0.3242	0.3666
	GWaveNet	0.1748	0.2799	0.3556	0.4768	0.2194	0.2612	0.3038	0.3410	0.2266	0.2692	0.3141	0.3576
	SST-GCN_{mv}	0.5705	0.6294	0.6752	0.7548	0.3240	0.3679	0.4038	0.4526	0.3574	0.413	0.464	0.5014
	SST-GCN	**0.1662**	**0.2619**	**0.3319**	**0.4458**	**0.2081**	**0.2494**	**0.2938**	**0.3358**	**0.2119**	**0.2589**	**0.3041**	**0.3491**
D_{JSD}	VAR	0.8789	0.8737	0.8737	0.8737	0.8537	0.8873	0.9251	0.9504	1.2232	1.2082	1.1965	1.1931
	MICE	0.2237	0.3500	0.4963	0.7184	0.4710	0.5728	0.6981	0.8562	0.4875	0.5987	0.7048	0.8043
	GMAN	0.5605	0.6289	0.7079	0.8974	0.3507	0.3987	0.4601	0.5130	0.3018	0.3411	0.3888	0.4398
	GCWC	0.5158	0.5763	0.6316	0.7105	0.5963	0.6308	0.6653	0.6871	0.7074	0.7132	0.7291	0.7441
	TCN	0.3368	0.4395	0.5421	0.6921	0.2742	0.3255	0.3726	0.4214	0.2834	0.3194	0.3662	0.4130
	ST-MetaNet	0.3105	0.4026	0.5026	0.6474	0.2750	0.3499	0.3877	0.4340	0.2651	0.3161	0.3570	0.4156
	MTGNN	0.2579	0.4316	0.5632	0.6921	0.2582	0.3078	0.3608	0.4087	0.2550	0.2968	0.3512	0.4013
	GWaveNet	0.2184	0.3579	0.4579	0.6158	0.2540	0.3045	0.3625	0.4138	0.2453	0.2943	0.3448	0.3938
	SST-GCN_{mv}	0.5868	0.6579	0.7158	0.8237	0.3650	0.4163	0.4693	0.5265	0.3896	0.4390	0.4916	0.5435
	SST-GCN	**0.1974**	**0.3053**	**0.3895**	**0.5342**	**0.2422**	**0.2902**	**0.3474**	**0.3970**	**0.2408**	**0.2776**	**0.3319**	**0.3829**
D_{EMD}	VAR	1.0457	1.0481	1.0555	1.0841	1.6112	1.6109	1.6246	1.6378	2.2529	2.2280	2.2118	2.2196
	MICE	**0.2402**	**0.3803**	**0.5133**	0.7292	0.4168	0.5114	0.6235	0.7479	0.4165	0.5101	0.6045	0.7018
	GMAN	0.7038	0.7472	0.8267	0.9298	0.4865	0.5243	0.5648	0.5959	0.4508	0.4832	0.5231	0.5622
	GCWC	0.7274	0.7740	0.8068	0.8655	0.6482	0.6689	0.6912	0.7146	0.7286	0.7358	0.7438	0.7530
	TCN	0.5065	0.6442	0.7135	0.8323	0.4519	0.5000	0.5434	0.5792	0.4386	0.4779	0.5209	0.5530
	ST-MetaNet	0.4792	0.5841	0.6844	0.8073	0.4360	0.4819	0.5128	0.5459	0.4061	0.4612	0.4985	0.5445
	MTGNN	0.3304	0.5018	0.6197	0.7671	0.3983	0.4475	0.4958	0.5333	0.3985	0.4427	0.4950	0.5400
	GWaveNet	0.3651	0.5342	0.6423	0.7717	0.4016	0.4486	0.4947	0.5351	**0.3475**	0.4073	0.4640	0.5196
	SST-GCN_{mv}	0.6063	0.6497	0.6872	0.7528	0.4491	0.4901	0.5254	0.5704	0.4686	0.5061	0.5499	0.5875
	SST-GCN	0.3364	0.4681	0.5591	**0.7130**	**0.3668**	**0.4235**	**0.4786**	**0.5245**	0.3570	**0.4014**	**0.4602**	**0.5101**

Table 5: Effect of Context-Aware Diffusion Process.

Datasets / ρ		**SST-GCN**	SST-GCN_{NC}	GWaveNet	GWaveNet_{C}
HT	0.5	**0.1974**	0.2000	0.2184	0.2105
	0.6	**0.3053**	0.3184	0.3579	0.3395
	0.7	**0.3895**	0.4105	0.4579	0.4421
	0.8	**0.5342**	0.5658	0.6158	0.5947
CR_{173}	0.5	**0.2422**	0.2498	0.2540	0.2473
	0.6	**0.2902**	0.2986	0.3045	0.3003
	0.7	**0.3474**	0.3549	0.3625	0.3532
	0.8	**0.3970**	0.4071	0.4138	0.4003
CR_{1026}	0.5	**0.2408**	0.2542	0.2453	0.2458
	0.6	**0.2776**	0.2910	0.2943	0.2885
	0.7	**0.3319**	0.3420	0.3448	0.3403
	0.8	**0.3829**	0.3972	0.3938	0.3905

the normalized JSD. The results for KLD and EMD have similar behavior, thus we omit them due to space limitations.

Table 5 shows that the context-aware diffusion process improves the accuracy of both **SST-GCN** and GWaveNet by approximately 3% and 2.6% on average. Moreover, the context-aware diffusion process has a higher impact on the estimation accuracy as ρ increases. For example, on HT, the accuracy of **SST-GCN** is improved by 1.3% over SST-GCN_{NC} when $\rho = 0.5$, and by 5.5% when $\rho = 0.8$. Based on this analysis, we conclude that our proposed context-aware diffusion process better captures the spatial correlations between the road segments when the graph contains many nodes with missing information. Finally, although the context-aware diffusion process improves the accuracy of GWaveNet, it is still not better than our **SST-GCN**. This is because GWaveNet cannot capture the spatio-temporal correlations among the speed ranges.

Table 6: Effect of considering separate speed ranges.

Datasets / Methods / $\rho =$		0.5	0.6	0.7	0.8
HT	**SST-GCN**	**0.1974**	**0.3053**	**0.3895**	**0.5342**
	SST-GCN_{NH}	0.2105	0.3263	0.4132	0.5658
CR_{173}	**SST-GCN**	**0.2422**	**0.2902**	**0.3474**	**0.3970**
	SST-GCN_{NH}	0.2498	0.2986	0.3583	0.4146
CR_{1026}	SST-GCN	**0.2408**	**0.2776**	**0.3319**	**0.3829**
	SST-GCN_{NH}	0.2458	0.2901	0.3353	0.3888

6.6 Speed Range Feature Vectors Evaluation

To better understand the effect of learning different convolutional kernels for each speed range, we implement a new model, SST-GCN_{NH} (NH stands for non-histogram). SST-GCN_{NH} does not consider the features of each histogram bucket separately but combines them into a single representation. Table 6 shows the results for the normalized JSD. As can be seen, **SST-GCN** improves the accuracy by 4% over SST-GCN_{NH}, indicating the effectiveness of our design decision to learn convolutional kernels for each speed range separately in the ST-Block. The results for the KLD and EMD measures also show a similar improvement.

6.7 Model Efficiency Analysis

Table 7 compares the space and time efficiency of **SST-GCN** against the deep learning baselines on the biggest CR_{1026} graph. In terms of both time and space efficiency, **SST-GCN** is well-positioned w.r.t the baselines. The inference and training time is faster than ST-MetaNet and GMAN, and only slightly slower than the remaining baselines. Having the lowest number of parameters, TCN has the lowest training and inference times. However, **SST-GCN** is from 16% to

Table 7: Efficiency analysis on CR$_{1026}$ dataset.

Methods	Parameters ($\times 10^3$)	Inference time (s/b)	Training time (s/b)
GCWC	~ 20000	~ 0.004	~ 0.01
GMAN	~ 380	~ 0.08	~ 0.21
TCN	~ 2.8	~ 0.001	~ 0.01
ST-MetaNet	~ 18	~ 0.10	~ 0.30
MTGNN	~ 135	~ 0.01	~ 0.13
GWaveNet	~ 192	~ 0.01	~ 0.05
SST-GCN	~ 45	~ 0.03	~ 0.14

18% more accurate than TCN. In contrast, both GWaveNet and GCWC have from 4 to 400 times more parameters, which increases their space complexity. This is because GCWC's decoder component utilizes a fully connected layer to reconstruct the stochastic speed matrix for all nodes in the graph. Therefore, as the number of nodes increases, the number of parameters in this last layer also increases. GWavenet uses 256 kernels in the skip connection as suggested in [24], making the number of parameters also higher than SST-GCN. Nevertheless, in both cases, the layer with the high number of parameters requires only a single matrix multiplication that is computed very fast when the data fits in the GPU memory. Finally, although ST-MetaNet has the second fewest parameters, its inference and training times are much higher due to the use of three stacked RNNs which cannot be parallelized.

6.8 Other Experiments

We conduct further experiments to evaluate two aspects of our **SST-GCN**: 1) resilience to spatially localized missing values and 2) scalability to bigger graphs. Due to space limitations, we include the detailed description and results for these experiments in Appendices A.5 and A.6. In the first experiment, we generate a new dataset where the nodes with missing values are clustered and compare **SST-GCN** against GWaveNet. The results show that the accuracy of **SST-GCN** gets only slightly affected and is still better than GWaveNet. In the second experiment, we show that estimating the missing stochastic speed in a large road network (3064 nodes) simply requires partitioning the graph into smaller subgraphs and applying **SST-GCN** on those subgraphs independently. Finally, we also study in more detail the intermediate representation learned by our *Encoder* in Appendix A.7. Overall, the experiments show that our *Encoder* can differentiate the speed ranges and capture the similarities between adjacent nodes in the edge graph.

7 CONCLUSION AND FUTURE WORK

This paper introduces a novel deep learning architecture, SST-GCN, to impute missing speed distributions in a road network. Unlike prior work, SST-GCN efficiently captures both spatial and temporal correlations and uses a novel context-aware diffusion process that effectively prioritizes the observed nodes in the graph while estimating the missing values. This context-aware diffusion process is generic, and thus can be applied to any graph-based model for efficient learning. Extensive experiments show that SST-GCN outperforms the state-of-the-art on estimation accuracy from 4.6% up to 50% while being competitive in terms of time and space complexity. Furthermore, multiple ablation studies confirm our design choices and scalability to large road networks. In future work, we plan to extend SST-GCN to simultaneously estimate the missing distributions of current time intervals and forecast the distributions of future time intervals, paving the path toward even more efficient routing services.

ACKNOWLEDGMENTS

This paper was supported in part by the MORE project funded by the EU Horizon 2020 program under grant agreement no. 957345.

REFERENCES

[1] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. 2020. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. *NeurIPS* (2020).

[2] Michael George Bulmer. 1979. *Principles of statistics.* Courier Corporation.

[3] Xinyu Chen, Zhaocheng He, and Jiawei Wang. 2018. Spatial-temporal traffic speed patterns discovery and incomplete data recovery via SVD-combined tensor decomposition. *Transportation Research Part C* (2018).

[4] Xinyu Chen, Jinming Yang, and Lijun Sun. 2020. A nonconvex low-rank tensor completion model for spatiotemporal traffic data imputation. *Transportation Research Part C* (2020).

[5] Jian Dai, Bin Yang, Chenjuan Guo, Christian Søndergaard Jensen, and Jilin Hu. 2016. Path cost distribution estimation using trajectory data. *VLDB J.* (2016).

[6] Dingxiong Deng et. al. 2016. Latent Space Model for Road Networks to Predict Time-Varying Traffic. In *SIGKDD.*

[7] Zheyi Pan et. al. 2019. Urban Traffic Prediction from Spatio-Temporal Data Using Deep Meta Learning. In *SIGKDD.*

[8] Zonghan Wu et. al. 2020. Connecting the Dots: Multivariate Time Series Forecasting with Graph Neural Networks. In *SIGKDD.*

[9] Bent Fuglede and Flemming Topsøe. 2004. Jensen-Shannon divergence and Hilbert space embedding. In *ISIT.*

[10] Feng Guo, Dongqing Zhang, Yucheng Dong, and Zhaoxia Guo. 2019. Urban link travel speed dataset from a megacity road network. *Scientific Data* (2019).

[11] Jilin Hu, Chenjuan Guo, Bin Yang, and Christian S Jensen. 2019. Stochastic weight completion for road networks using graph convolutional networks. In *ICDE.*

[12] Tsuyoshi Idé and Masashi Sugiyama. 2011. Trajectory regression on road networks. In *AAAI.*

[13] Ian T Jolliffe. 1986. Principal components in regression analysis. In *Principal component analysis.* Springer.

[14] Nathaniel J. Kendall, Alex Risner, and Harry saxophone Allen. 2017. Highway Tollgates Traffic Flow Prediction. In *SIGKDD.*

[15] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* (1951).

[16] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR.*

[17] Simon Aagaard Pedersen, Bin Yang, and Christian S Jensen. 2020. Fast stochastic routing under time-varying uncertainty. *VLDB J.* (2020).

[18] Ofir Pele and Michael Werman. 2009. Fast and robust earth mover's distances. In *ICCV.*

[19] Matthew Roughan, Yin Zhang, Walter Willinger, and Lili Qiu. 2012. Spatio-Temporal Compressive Sensing and Internet Traffic Matrices. *IEEE/ACM Transactions on Networking* (2012).

[20] James H Stock and Mark W Watson. 2001. Vector autoregressions. *JEP* (2001).

[21] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. 2013. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *SIGKDD.*

[22] Stef Van Buuren and Karin Groothuis-Oudshoorn. 2011. mice: Multivariate imputation by chained equations in R. *Journal of statistical software* (2011).

[23] Aäron Van Den Oord et. al. 2016. WaveNet: A generative model for raw audio. *SSW* (2016).

[24] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In *IJCAI.*

[25] Bin Yang, Jian Dai, Chenjuan Guo, Christian S. Jensen, and Jilin Hu. 2018. PACE: a PAth-CEntric paradigm for stochastic path finding. *VLDB J.* (2018).

[26] Bin Yang, Chenjuan Guo, Christian S. Jensen, Manohar Kaul, and Shuo Shang. 2014. Stochastic skyline route planning under time-varying uncertainty. In *ICDE.*

[27] Bin Yang, Manohar Kaul, and Christian S Jensen. 2013. Using incomplete information for complete weight annotation of road networks. *TKDE* (2013).

[28] Fisher Yu and Vladlen Koltun. 2016. Multi-Scale Context Aggregation by Dilated Convolutions. In *ICLR.*

[29] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. GMAN: A Graph Multi-Attention Network for Traffic Prediction. In *AAAI.*

[30] Wanzheng Zheng, Pranay Thangeda, Yagiz Savas, and Melkior Ornik. 2021. Optimal Routing in Stochastic Networks with Reliability Guarantees. In *24th IEEE International ITS.*

A APPENDIX

A.1 Context-Aware GCN Generalization

In this section, we provide formal proof of Lemma 1.

Proof. We prove Eq. (13) using induction in k.

Base case: when $k = 1$, let $m \geq 0$ be the number of unobserved nodes within 1-hop distance. From Eq. (9), we obtain the transition matrix $\hat{P}_1$ for $\mathcal{P}_C$, and the transition matrix P_1 for $\mathcal{P}_{\overline{C}}$ as

$$\hat{P}_1 = \frac{1}{\text{colsum}(A_c)} A_c;\ P_1 = \frac{1}{\text{colsum}(A)} A \tag{15}$$

From Eq. (15), we obtain the weights of the observed neighboring node u of v as

$$\hat{P}_1[u;v] = \frac{1}{d_v^{in} - m} A_c[u;v] \geq P_1[u;v] = \frac{1}{d_v^{in}} A[u;v] \tag{16}$$

where $d_v^{in} > 0$ is the in-degree of node v and $c_t[u] = 1$. Thus, Eq. (13) hold for $k = 1$.

Induction hypothesis: assume that Eq. (13) holds for some positive k. Thus, we have :

$$\hat{P}_k[u;v] \geq P_k[u;v],\ \forall u \in \mathcal{N}_k(v) \wedge c_t[u] = 1 \tag{17}$$

Induction step: we prove that Eq. (13) also holds for $k + 1$. Let $m' \geq 0$ be the number of nodes that are still unobserved after k aggregations. From Eq. (10), we obtain the transition matrices for $\mathcal{P}_C$ and $\mathcal{P}_{\overline{C}}$ after $k + 1$ aggregations as

$$\hat{P}_{k+1} = \hat{P}_k \hat{P}';\ P_{k+1} = P_k P' \tag{18}$$

The weight of the observed neighboring node u of v is therefore computed as

$$\begin{aligned} \hat{P}_{k+1}[u;v] &= \sum_{i \in \mathcal{N}_k(v)} \hat{P}_k[u;i] \hat{P}'[i;v] \\ P_{k+1}[u;v] &= \sum_{i \in \mathcal{N}_k(v)} P_k[u;i] P'[i;v] \end{aligned} \tag{19}$$

where

$$\hat{P}'[i;v] = \frac{1}{d_v^{in} - m'} \geq P[i;v] = \frac{1}{d_v^{in}} \tag{20}$$

Since we have $\hat{P}_k[u;i] \geq P_k[u;i]$ by the induction hypothesis, we obtain Eq. (13) from Eqs. (19), (20) QED. □

A.2 JSD and EMD Metrics

The *Jensen-Shannon Divergence* (JSD) is computed as

$$\text{JSD}(w, \hat{w}) = \frac{\text{KL}(w||\bar{w}) + \text{KL}(\hat{w}||\bar{w})}{2} \tag{21}$$

where $\bar{w} = 0.5 \times (w + \hat{w})$ and KL is the textitKullback-Leibler Divergence (KLD), shown in Eq. (3). The JSD measure is symmetric, bounded, and satisfies the triangle inequality property.

The *Earth Mover's Distance* (EMD) measures the distance between two probability distributions $(w, \hat{w})$ as the minimum cost to transform $\hat{w}$ into w. The cost is defined as the product of two factors: the difference between the probability values of histogram's buckets, e.g., $\hat{w}_i - w_j$, and the measuring distance between the buckets themselves, e.g., $|i - j| + 1$. Formally, EMD is defined as

$$\text{EMD}(w, \hat{w}) = \frac{\sum_{i=1}^{M} \sum_{j=1}^{M} F_{ij} C_{ij}}{\sum_{i=1}^{M} \sum_{j=1}^{M} F_{ij}} \tag{22}$$

where F_{ij} is the optimal flow that minimizes the transformation cost from $\hat{w}_i$ to w_j, and C_{ij} is the pairwise distance between bucket i and bucket j.

A.3 Hyper-Parameters Searching

All deep learning models are trained using the mini-batch Adam optimizer, with batch size 32, the learning rate (lr) is searched from {0.001, 0.002, ..., 0.01}, the dropout (dp) ratio is tuned in {0.05, 0.1, 0.15, 0.2, ..., 0.5}, and the number of diffusion processes (K) per layer is searched from {2, 3, 4, 8}. For MTGNN, we set the number of neighbors parameter $k = 20$ for the CR-derived datasets as suggested in [8], and $k = 5$ for *HT* since it has only 24 nodes. For GMAN, we search for the best results using several attention heads, i.e., {2, 3, 4}, and set the node embedding size to 64. The number of kernels for graph convolution and size of the hidden layer of RNN is searched from {8, 16, 32}. The regularization term is fixed to 0.001 for SST-GCN, while for the baselines we use the value suggested by the authors. In SST-GCN, the number of *ST-Blocks* is set to 2, and we maintain the same number of kernels for each layer. Grid search is used for all methods.

A.4 Edge Graph Visualization

Figure 4 shows the *HT* and CR_{173} edge graphs. The CR_{1026}'s edge graph is too big for visualization but, in general, it follows the same topology as CR_{173}. Note that the topology of the two graphs is completely different. *HT* is a dispersed graph with few connections between nodes, while *CR* has many connections, i.e., a higher average degree. This clearly has an impact on all baselines, as well as our **SST-GCN**. Note that the results deteriorate faster in *HT* than in the *CR* datasets as the number of missing values increases. Intuitively, since *CR* datasets have more connections, observed information can reach the rest of the nodes faster than in *HT*.

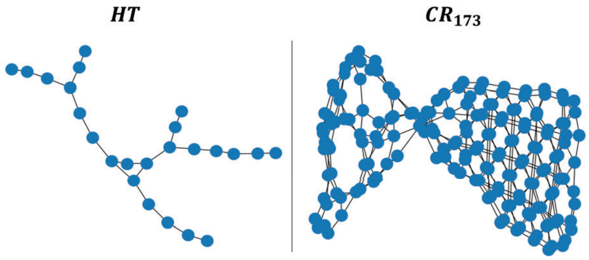


Figure 4: Visualization of *HT* and CR_{173} edge graphs.

A.5 Spatially Localized Missing Values

In this section, we evaluate the resilience of **SST-GCN** by artificially generating missing values that are concentrated in the same region. We note that in all other experiments, the missing values are randomly distributed in the road network. To generate such concentrated missing regions, we randomly choose a starting node v and perform a Breadth-First Search starting from v, labeling every node reached on the way as missing. We repeat this process to create multiple missing clusters in the tested road network, using the missing values rate ρ to control the cluster size. In this experiment, we use the CR$_{1026}$ graph and the average cluster size is 58 nodes.

Table 8 shows the results of **SST-GCN** compared to the best baseline GWaveNet for all metrics. We can see that the accuracies of **SST-GCN** and GWaveNet are only slightly affected. Specifically, SST-GCN experiences only from 0.98% to 3.8% (2.7% on average) accuracy reduction for all the metrics. Intuitively, this is attributed to the highly connected road networks (cf. Table 3) that allow the observed information to reach many missing nodes in just a few hops. In comparison, GWavenet experiences from 0.95% to 4.2% (2.2% on average) accuracy reduction for all the metrics. Additionally, our **SST-GCN** still obtains better results overall with an average accuracy improvement of 2% over GWaveNet. Finally, we would like to emphasize that, in general, the random missing node scenario is more realistic as GPS or sensor malfunctioning is highly unlikely to follow a specific pattern like in this experiment. Nevertheless, with this set of experiments, we show that even if such a pattern occurs, our method can still achieve higher accuracy than the closest baseline.

Table 8: Impact of spatially localized missing values.

Metric	Method/ρ =	0.5	0.6	0.7	0.8
D_{KLD}	**SST-GCN**	**0.2253**	**0.2677**	**0.3138**	**0.3545**
	GWaveNet	0.2311	0.2743	0.3201	0.3605
D_{JSD}	**SST-GCN**	0.2508	**0.2960**	**0.3428**	**0.3913**
	GWaveNet	**0.2478**	0.2971	0.3484	0.3938
D_{EMD}	**SST-GCN**	**0.3547**	**0.4109**	**0.4641**	**0.5176**
	GWaveNet	0.3685	0.4262	0.4848	0.5336

A.6 Scalability by Partitioning

As discussed in Section 5.2, to estimate the missing value of a node v, **SST-GCN** only needs to propagate information from the neighboring nodes that are within its K-hop distance. Therefore, applying **SST-GCN** to a very big road network simply requires partitioning the graph into smaller subgraphs and applying the method on those subgraphs independently. The scalability of **SST-GCN** is thus guaranteed if the estimation results obtained from the subgraphs are similar to the estimation results obtained from the full big graph.

Table 9: Accuracy over partitions.

Datasets/ρ =	0.5	0.6	0.7	0.8
CR_{1522}	0.2387	0.2847	0.3324	0.3811
CR_{1542}	0.2604	0.3175	0.3728	0.4256
CR_{3064}	0.2511	0.3010	0.3508	0.4007

We validate this approach by comparing the results of **SST-GCN** on a bigger graph that consists of 3064 nodes extracted from the CR dataset (CR_{3064}) with the results on two disjoint subgraphs of CR_{3064}, CR_{1522} and CR_{1542}. The results are shown in Table 9 using the JSD metric. We can see that **SST-GCN** obtains similar accuracy when training and testing on the full graph CR_{3064}, and when training and testing on the two subgraphs separately. This shows that using this partitioning strategy, **SST-GCN** can be applied to very big road networks and can still obtain high accuracy while maintaining the training and inference times that grow linearly to the number of partitions. We also believe that more advanced partitioning strategies such as those that consider overlapping between different partitions can further improve the results.

A.7 Intermediate Representation Visualization

To gain more intuition about the learned intermediate representation $\mathbb{L} \in \mathbb{R}^{N \times M \times D}$ of **SST-GCN**, we use Principal Component Analysis (PCA) [13] to analyze the feature vectors associated with each speed range and node in the graph. Figure 5 visualizes the feature vectors of two adjacent nodes using the first two principal components. Here, different colors represent different histogram buckets. First, we observe that the histogram buckets of two adjacent nodes have similar feature representations. This confirms our intuition that the probability of driving at a certain speed range is similar in adjacent road segments. However, this does not apply to all speed ranges, as there are other factors, such as traffic lights, that also affect the vehicle's speed. For example, the features of the bucket [20-30) are slightly different. Second, we observe that the feature vectors of different buckets form relatively independent clusters, and the features of consecutive speed ranges (adjacent buckets) are closer to each other than the features of non-consecutive speed ranges. This helps explain the better performance of **SST-GCN** over the baselines, as **SST-GCN** explicitly learns the feature representation of each bucket separately, and thus can capture distinct features that are representative for each speed range.

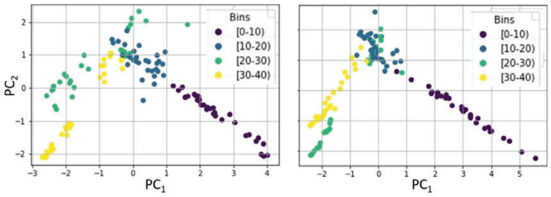


Figure 5: Feature vectors of two adjacent nodes.