Nuhuo: An Effective Estimation Model for Traffic Speed Histogram Imputation on A Road Network

Haitao Yuan
Nanyang Technological University
Singapore
haitao.yuan@ntu.edu.sg

Gao Cong
Nanyang Technological University
Singapore
gaocong@ntu.edu.sg

Guoliang Li
Tsinghua University
Beijing, China
liguoliang@tsinghua.edu.cn

ABSTRACT

Traffic speed histograms show the distribution of traffic speeds over a certain period. Traffic speed might not be recorded continuously, leading to missing histograms for some links on a road network. However, accurate imputation of missing histograms is a critical yet challenging task. This paper introduces a novel framework to address four previously unexplored dimensions crucial for precise traffic speed histogram estimation: regionality, proximity, sparsity, and volatility. First, to address the challenge of regionality and proximity, we employ a global partition graph that captures both regional and proximal correlations within the road network. Next, in response to the challenge of sparsity, the framework features a disentangled feature encoding pipeline, comprising a global encoder and a localized spatio-temporal encoder. This design allows for the effective handling of entangled spatio-temporal dimensions, thereby mitigating the issues related to input sparsity. In particular, the framework leverages graph neural networks and recurrent neural networks to capture spatial and temporal correlations. In addition, to encompass the complexities of spatio-temporal correlations both on global and local scales, we employ a two-layer fusion module with an attention-based mechanism for representation integration. Lastly, to mitigate the challenge of volatility due to missing values, we incorporate a self-supervised learning task using an auto-encoder framework, enhancing the stability and robustness of the encoding models. Extensive evaluations on two real-world datasets confirm that our method significantly outperforms state-of-the-art solutions in terms of both accuracy and robustness.

PVLDB Reference Format:
Haitao Yuan, Gao Cong, and Guoliang Li. Nuhuo: An Effective Estimation Model for Traffic Speed Histogram Imputation on A Road Network. PVLDB, 17(7): 1605 - 1617, 2024.
doi:10.14778/3654621.3654628

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/yuanhaitao/Nuhuo.git.

1 INTRODUCTION

The acquisition of traffic speed data across road networks serves as a foundational preprocessing step for a myriad of transportation applications. These range from travel time estimation [34, 35] and optimal route planning [6, 23, 24, 33] to the identification of traffic anomalies [21]. In certain real-world contexts, it becomes crucial to examine the distribution of traffic speeds, often represented in histogram formats [14, 20]. However, the collection of comprehensive traffic speed data often faces challenges due to the absence of loop detectors or vehicle-generated data, resulting in numerous missing values. To mitigate this, existing literature [10, 18, 20, 30] proposes effective methodologies for imputing missing speed histograms or general spatio-temporal data. Specifically, these methodologies concentrate on learning spatio-temporal representations for link features by harnessing spatial and temporal correlations. This is typically achieved through the employment of technologies such as Graph Neural Networks or Attention Mechanisms, which are adept at encapsulating the constraints of the entire road network. Additionally, temporal convolutional networks and diffusion networks are often utilized to effectively capture temporal dependencies. Despite these advancements, there remain critical gaps that have yet to be addressed by current methodologies.

(1) **Regionality and proximity: spatially multi-view correlations.** Spatial correlations in traffic speed data can be broadly categorized into two distinct types: proximity-based and regionality-based correlations [34, 36]. Proximity-based correlations illustrate the direct impact of one road link's traffic condition on neighboring links. For instance, an incident on a link might initiate a cascading effect, influencing the conditions on adjacent links, and thus altering their traffic speed patterns. On the other hand, regionality-based correlations stem from distinct travel behaviors tied to specific regions. An example of this can be seen in business districts, where high vehicle concentration leads to decreased speeds during rush hours. It's crucial to note that existing methods struggle to accurately capture both types of correlations simultaneously.

(2) **Sparsity: ineffectiveness of entangled spatio-temporal representations:** A significant challenge arises from the frequent absence of data on some links for some time intervals, leading to sparse training data. Existing techniques [14, 20] often directly produce entangled spatio-temporal representations due to their reliance on more extensive training data, which can be inefficient. For instance, consider spatial and temporal dimensions with respective sizes of m and n. In a simplistic scenario, a minimum of $m + n$ samples would suffice to span these dimensions. Yet, when these dimensions are entangled, the minimum sample requirement for effective representation increases to $m \times n$, a number significantly higher than the sum of the dimensions considered separately.

(3) **Volatility: missing values in inputs lead to unstable encodings.** Existing learning-based methods generate representations

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 17, No. 7 ISSN 2150-8097.
doi:10.14778/3654621.3654628

from input features, but often struggle with raw traffic data containing missing values. Existing techniques either overlook these missing values [30] or use default values like historical averages [14, 20] and linear interpolation [18], without constraining the resulting representations. This leads to volatility in feature-based encodings. For example, consider labeled data $[1, 2, 3, 4]$. If the first two items are missing, the encoding becomes $f_\theta([?, ?, 3, 4])$, whereas it would be $f_\theta([1, 2, ?, ?])$ if the last two items are missing. Despite the differences in input, a robust encoder should recognize the similarity between these two encodings as they originate from the same labeled data. However, there is a notable lack of constraint in the encoding generation process, which fails to address this volatility issue adequately.

To comprehensively address the aforementioned challenges, we introduce a novel framework, denoted as **Nuhuo**, designed for the accurate estimation of missing traffic speed histograms. Initially, **to address the first challenge**, we segment the entire road network into non-overlapping partitions and construct a global partition graph to encapsulate the network's structure. This approach enables us to capture regionality-based correlations by integrating partition-specific encodings into the graph. Concurrently, it allows for the assessment of proximity-based correlations among individual links within each partition. Next, **in response to the second challenge**, we introduce a disentangled feature encoding pipeline comprising a global encoder and a localized spatio-temporal encoder. Specifically, the global encoder is employed to independently generate spatial and temporal representations at the partition level, utilizing global features such as the partition graph and time intervals. Subsequently, additional features pertinent to each partition are processed by the local spatio-temporal encoder, yielding spatial and temporal representations at the link level. To further capture spatio-temporal correlations both on global and local scales, the final spatio-temporal representation is synthesized through a two-layer fusion framework, which initially fuses global and local representations from both spatial and temporal perspectives, followed by a module that integrates these spatial and temporal representations. At last, to enhance the robustness of our encoding models **in response to the third challenge**, we incorporate a self-supervised learning task. Specifically, we utilize an auto-encoder framework to generate spatio-temporal representations for complete ground-truth data, which then serve as supervisory labels for the training of our encoding modules. In summary, we develop a self-supervised task specifically designed to impose explicit constraints on the encoding model's learning process. This strategic intervention is aimed at significantly reducing the volatility typically associated with varying input data, thereby enhancing the stability and reliability of the model's output.

In summary, the main contributions are listed as follows.

- We identify and address four critical yet previously unexplored dimensions essential for accurate traffic speed histogram imputation: regionality, proximity, sparsity, and volatility. We then develop a fine-grained pipeline that leverages features pertinent to these dimensions. (Section 2)
- We design a comprehensive neural network model, **Nuhuo**, that can fully exploit different spatio-temporal features to achieve an accurate traffic speed histogram estimation. (Section 3)

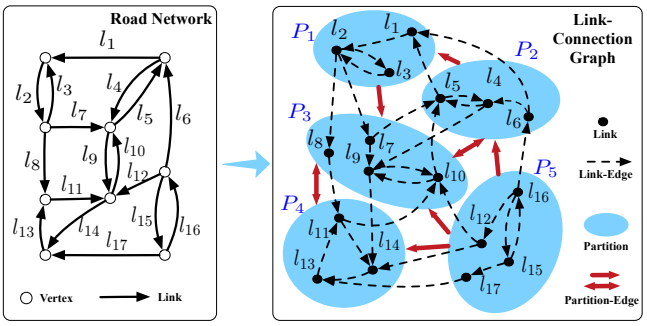

Figure 1: Road Network & Link-Connection Graph

- We employ a fusion module to encompass the complexities of spatio-temporal correlations both on global and local scales. This module leverages an attention-based mechanism to amalgamate diverse representations. To enhance stability, we incorporate an auto-encoder module into the learning process. (Section 4 and 5)
- We conduct a comprehensive evaluation on two real-world datasets. Our method demonstrates significant improvements over existing state-of-the-art solutions in terms of both accuracy and robustness. (Section 6)

2 PRELIMINARY

We first introduce some key concepts and then formalize the speed histogram imputation problem on a road network.

2.1 Basic Concepts

Road Network & Link-Connection Graph(LCG). As shown in the left part of Fig. 1, a road network is modeled as a directed graph $\langle V, L\rangle$, where V is a vertex set and L is a link set. Each link $l_i \in L$ represents a road segment linking two vertices. In this paper, we focus on spatio-temporal data of links, so we convert the road network into a link-connection graph (**LCG**) $G = \langle L, E\rangle$, where each node $l \in L$ in the **LCG** corresponds to an edge (link) in the original road network. Therefore, the number of nodes in the **LCG** is equal to the number of edges in the road network. Edges in the **LCG** are established based on the road network's topology; specifically, if two links are adjacent in the road network, this adjacency is represented as an edge in the **LCG**.

LCG Partitions. We further utilize some graph partition approaches to divide the whole graph into disjoint partitions. The reason for partitioning G is two-fold. On the one hand, capturing the relationship between every two links in the whole graph is inefficient, especially when the number of links is too large. For example, if there are N links in the whole graph, the adjacent matrix would be in the shape of $N \times N$. In contrast, if we keep the partition balance and let the number of links in each partition be M ($M \ll N$), there would be $\lceil \frac{N}{M} \rceil$ partitions, and each partition corresponds to an adjacent matrix with the shape of $M \times M$, so the full size would be $\lceil \frac{N}{M} \rceil M \times M \approx N \times M \ll N \times N$. On the other hand, according to the first law of geography [19], near things are more related than distant things, so it's unnecessary to capture fine-grained correlations for distant links. As shown in the right part of Fig. 1, we use the link-edge to model the relationship between near links in a

partition while using the partition-edge to model the relationship between distant links. For simplicity, we demote the k-partition as $P_k = \langle L_k, E_k \rangle$, where $L_k \subset L, E_k \subset E$, and each $e = \langle l_i, l_j \rangle$ in E_k means both l_i and l_j belong to L_k. Hence, we reformulate the link-connection graph with $G =< \mathcal{P}, \mathcal{E} >$, where $\mathcal{P} = \{P_1, P_2, \cdots\}$ denotes the set of partitions, and $\mathcal{E}$ denotes the set of edges between partitions. Specifically, if $l_i \in P_{k_1}$ and $l_j \in P_{k_2} (k_1 \neq k_2)$, there would be an edge $\langle P_{k_1}, P_{k_2} \rangle \in \mathcal{E}$.

Remark. In this paper, we use Metis [12] as the graph partition method, which is chosen for its ability to minimize the impact of partitioning on the overall graph structure, effectively balancing the trade-offs between regionality and proximity. In particular, its efficacy in road network partitioning is well-documented in previous studies [27, 32]. In addition, due to the uneven nature of graph partitioning with the method Metis, the number of links in different partitions inherently suggests varying granularities.

Traffic Speed Histogram. Given a time interval t (e.g.,8:00-8:10AM) and a link l in the **LCG**, we consider the speed of all vehicles passing through the link during the time interval. Following the prior work [20], we represent the speed distribution at the time interval t as a histogram $hist_l^t$. In particular, considering the link's speed range(e.g., $0 \sim 60km/h$), we split it into m disjoint parts (denoted as $slot_1, \cdots, slot_k$) to approximate the speed distribution $hist_l^t = [\alpha_1, \cdots, \alpha_m]$, where $0 \leq \alpha_i \leq 1$ and $\sum_{i=1}^{m} \alpha_i = 1$. For example, given five vehicles' speed values (i.e., 5, 2, 12, 15, 20) and the two parts $slot_1 = [0, 10)$ and $slot_2 = [10, 20]$, the corresponding histogram would be $hist = [\frac{2}{5}, \frac{3}{5}]$. Specifically, considering all links L in the **LCG** at the interval t, we represent their speed distributions with a matrix $hist_L^t = [hist_{l_1}^t, \cdots, hist_{l_{|L|}}^t] \in R^{|L| \times m}$. Furthermore, the traffic speeds at continuous time intervals may have correlations, so we denote the speed histograms at $T = [t_1, \cdots, t_{|T|}]$ time intervals as the following tensor $hist_L^T = [[hist_{l_1}^{t_1}, \cdots], \cdots, [\cdots, hist_{l_{|T|}}^{t_{|T|}}]] \in R^{|L| \times |T| \times m}$.

2.2 Problem Formulation

Traffic Speed Histogram Imputation on A Road Network. In the real-world scenario, it's impractical to collect all vehicles' information to compute the traffic speed histogram for each link on the road network, and there even exist some sparse links without any speed value. Although we can leverage some statistical methods (e.g., the average value of historical speeds) to impute traffic speed histograms, it requires an effective method to generate a more accurate speed histogram for each link in the road network. To distinguish observed and missing speed distributions, we use the matrix $M_L^T \in R^{|L| \times |T|}$ to label histograms in $hist_L^T$, where $M_L^T[l][t] = 0$ means the speed distribution of l at the time interval t is missing, otherwise $M_L^T[l][t] = 1$. Formally, we define the speed histogram imputation as an estimation problem as follows.

Definition 1 (Speed Histogram Imputation). *Given the* **LCG** $G = \langle L, E \rangle$, *a time interval sequence* $T = [t_1, \cdots, t_{|T|}]$, *the raw speed distributions* $hist_L^T$ *and the corresponding label matrix* M_L^T, *the goal is to estimate a new tensor* $\hat{hist}_L^T$ *such that each histogram* $\hat{hist}_L^T[l][t]$ *is as close as possible to the ground truth* $\bar{hist}_L^T[l][t]$. *In other words, we aim to learn a model* $g_\theta(\cdot)$ *as follows:*

$$\begin{aligned} &argmin_\theta \, d(\hat{hist}_L^T, \bar{hist}_L^T) \\ &s.t. \; \hat{hist}_L^T = g_\theta(G, T, hist_L^T, M_L^T) \end{aligned} \tag{1}$$

where θ *represents the model's parameter and* $d(\cdot, \cdot)$ *is a metric evaluating the alignment between estimates and ground truth.*

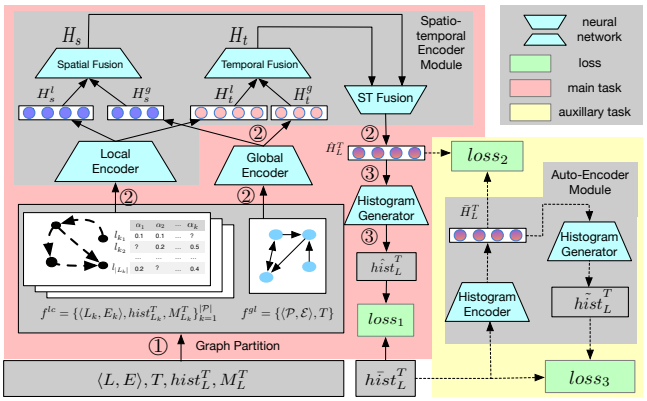

Figure 2: The Architecture of Our Proposed Model Nuhuo

3 MODEL OVERVIEW

In this section, we introduce a fine-grained pipeline to illustrate the process of deducing the estimated speed histograms from the given input and then elaborate on the design of our model. In particular, the model includes two learning tasks: the main imputation task and an auxiliary task, where the main task is to generate stochastic speed histograms following the outlined pipeline. Meanwhile, the auxiliary task is specifically designed to augment the learning of spatio-temporal hidden representations, thereby enriching the model's overall predictive capability.

3.1 A Pipeline of Speed Histogram Imputation

As shown in Fig. 2, the speed histogram imputation pipeline includes three steps as follows.

step ①: The first step belongs to a pre-processing procedure, which aims to extract local and global features from the given input (i.e., $\{\langle L, E \rangle, T, hist_L^T, M_L^T\}$). To improve the scalability, we utilize a graph partition algorithm, such as Metis [12] (Notably, the partition strategy is orthogonal to our proposed method), to divide the whole graph **LCG** into disjoint parts, as well as the data $hist_L^T$ and M_L^T, and hence generate the local feature $f_k^{lc} = \{\langle L_k, E_k \rangle, hist_{L_K}^T, M_{L_k}^T\}$ for each partition. For simplicity, we denote the set $\{f_k^{lc}\}_{k=1}^{|\mathcal{P}|}$ as f^{lc}. Moreover, to keep the spatial correlations among different partitions, we denote $\langle \mathcal{P}, \mathcal{E} \rangle$ as the global spatial feature. Similarly, we consider the feature T as the global temporal feature, which is shared by all partitions. In summary, we denote all global features as $f^{gl} = \{\langle \mathcal{P}, \mathcal{E} \rangle, T\}$.

step ②: The second step aims to encode both global and local spatio-temporal features into hidden representations. At first, to capture the spatio-temporal correlation among different partitions, we leverage the module *Global Encoder* to learn the hidden representations of global features. Next, we feed them and local features

into the module *Spatio-temporal Encoder* to further generate spatio-temporal representations of all links at different time intervals for each partition. For the sake of simplicity, we denote them as the set $H_L^T = \{H_{L_k}^T\}_{k=1}^{|\mathcal{P}|}$, where $H_{L_k}^T \in R^{|L_k| \times |T| \times d}$ denotes all links' representations in the k-th partitions and d is the representation's dimension.

Discussion. The intuitive method of generating H_L^T is to merge global and local representations which are generated by two independent modules respectively. However, it cannot capture the correlation between global and local features. According to Bayesian inference, training a regression model $y = g_\theta(x)$ equals to finding the optimal parameter $\theta^* = \arg\min_\theta -log(P(\theta|x, y))$. In our setting, we have $x = \langle f^{gl}, f^{lc} \rangle$, $\theta = \langle \theta_1, \theta_2 \rangle$ and $y = H_L^T$. According to the chain rule in probability theory, we have

$$\begin{aligned} log(P(\theta|x,y)) &= log(P(\theta_1, \theta_2 | f^{gl}, f^{lc}, H_L^T)) \\ &= logP(\theta_1 | f^{gl}, f^{lc}, H_L^T) + logP(\theta_2 | \theta_1, f^{gl}, f^{lc}, H_L^T) \\ &= logP(\theta_1 | f^{gl}, z) + logP(\theta_2 | z, f^{lc}, H_L^T). \end{aligned}$$

where $z = g_{\theta_1}(f^{gl})$ denotes the representation of f^{gl}. Subsequently, the whole model $y = g_\theta(x)$ can be converted into two sub-modules $z = g_{\theta_1}(f^{gl})$ and $H_L^T = g_{\theta_2}(z, f^{lc})$, where θ_1 and θ_2 respectively denote parameters in *Global Encoder* and *Spatio-temporal Encoder*.

step ③: The third step aims to deduce the speed histogram from the spatio-temporal representation for each link at each time interval. In particular, we first use the module *Histogram Generator* to transform hidden representations into estimated histograms $\hat{hist}_L^T$.

3.2 Model Design

Main task. We leverage different neural networks to implement three modules (i.e., *Global Encoder*, *Spatio-temporal Encoder* and *Histgoram Generator* respectively) as depicted in Section 3.1. In particular, *Global Encoder* aims to extract the spatial and temporal backgrounds/contexts from the given global features. Therefore, we design different neural networks in *Global Encoder* to generate two hidden representations H_s^g and H_t^g to denote spatially and temporally global context. As for *Spatio-temporal Encoder*, we need to generate spatio-temporal representation for local features based on the global context. To achieve this goal, we first propose an encoding network *Local Encoder* to generate the local-spatial representation H_s^l and the local-temporal representation H_t^l, and then fuse them with the global representations H_s^g and H_t^l. Specifically, we propose a two-layer fusion mechanism to deduce the final spatio-temporal representation H_L^T. In the first layer, we aim to fuse global and local representations from spatial and temporal perspectives respectively. Therefore, we design the module *Spatial Fusion* to fuse the local-spatial representation H_s^l and the global-spatial representation H_s^g into the spatial fused representation H_s. Similarly, we design the module *Temporal Fusion* to generate the temporal fused representation H_t by fusing local and global temporal representations H_t^l and H_t^g. Subsequently, in the second layer, we design the module *ST Fusion* to capture the correlation between spatial and temporal dimensions by fusing H_t^l and H_t^g into the final spatio-temporal representation $\hat{H}_L^T$. At last, we implement the module *Histogram Generator* with neural networks and deduce the imputation results $\hat{hist}_L^T = \{\hat{hist}_{L_k}^T\}_{k=1}^{|\mathcal{P}|}$ for all partitions. Notably, we denote the ground truth as $\bar{hist}_L^T$, so we can leverage some functions to measure the loss $loss_1$ between $\hat{hist}_L^T$ and $\bar{hist}_L^T$, which can help train the model by the back-propagation operation.

Auxiliary task. To enhance the model's effectiveness, we design an auxiliary branch to assist the learning of the spatio-temporal representation $\hat{H}_L^T$. On the one hand, $\hat{H}_L^T$ can be regarded as a low-dimensional representation of the estimated result $\hat{hist}_L^T$. On the other hand, the main goal of our problem is to make the estimated result $\hat{hist}_L^T$ close to the ground truth $\bar{hist}_L^T$. Therefore, we first design the encoder module *Histogram Encoder* to convert the ground truth $\bar{hist}_L^T$ into another low-dimensional representation $\bar{H}_L^T$, and then leverage some distance functions to constrain the loss $loss_2$ between $\hat{H}_L^T$ and $\bar{H}_L^T$ when training the whole model. In addition, to better train the encoder *Histogram Encoder*, we borrow the training structure of auto-encoder [28], which applies a generator module to covert the encoded low-dimensional representation into raw data and learn both the encoder and the generator in the self-supervised mode. Specifically, we set the generator to be the same as the generator module *Histogram Generator* in the main branch. In other words, these two generator modules share neural network weights. Subsequently, we can get new estimated results $\tilde{hist}_L^T = \{\tilde{hist}_{L_k}^T\}_{k=1}^{|\mathcal{P}|}$ for all partitions and hence can compute the loss $loss_3$ between $\tilde{hist}_L^T$ and the ground truth $\bar{hist}_L^T$ with the similar function as $loss_1$.

To sum up, the main components in our model consist of three encoder modules (i.e., *Global Encoder*, *Local Encoder* and *Histogram Encoder*), three fusion modules (i.e., *Spatial Fusion*, *Temporal Fusion* and *ST Fusion*), and a generator module (i.e., *Histogram Generator*). Notably, all modules are first trained with labeled samples in the training phase, and then we can leverage trained modules in the main branch to infer estimated results for testing samples. Next, we will elaborate on each module's detail to effectively incorporate both global-local and spatio-temporal characteristics.

4 MAIN BRANCH

As described in Section 3, the main branch in **Nuhuo**, aiming to generate speed histograms from given local and global features, includes a global encoder module, a spatio-temporal encoder module, and a histogram generator module, where the spatio-temporal encoder module consists of a local encoder and three fusion sub-modules. In particular, we first explain the design of the global encoder in Section 4.1, and then elaborate on the spatio-temporal encoder's details in Section 4.2, which can effectively incorporate spatio-temporal correlation among global and local features. At last, we focus on the procedure of speed histogram generation from encoded results in Sectoin 4.3.

4.1 Global Encoder

In the global encoding module, we take as input the global features $G = \langle \mathcal{P}, \mathcal{E} \rangle$ and T, where G represents all partitions of the link-connection graph, and T means the time intervals. The intuitive method is to encode G and T into spatial and temporal representations respectively. However, inter-regional traffic conditions between different partitions in G are dynamic according to the

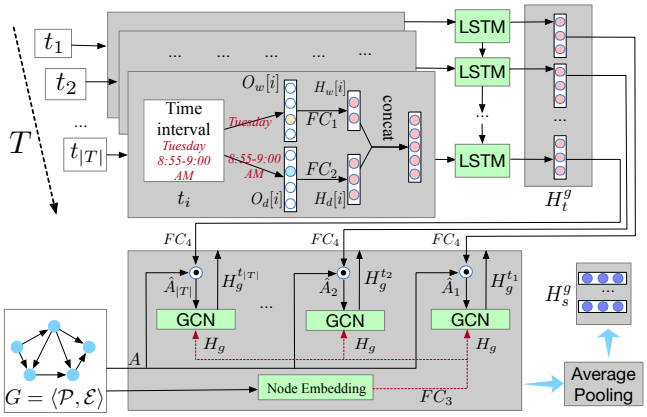

Figure 3: The Framework of Global Encoder

change of time intervals T. For example, the traffic demands differ between daytime and nighttime, and the weather varies at different time intervals. These variations can influence individuals' choices of transportation modes, consequently resulting in variations in the traffic congestion levels. In addition, from a perspective of probability and statistics, the joint distribution $p(G, T)$ can decomposed as $p(T) \cdot p(G|T)$, where $p(T)$ models the prior distribution of the global temporal feature T and $p(G|T)$ models the conditional distribution of G given T. Therefore, we first design a global temporal encoder to encode T into H_t^g, based on which we then design a global spatial encoder to encode G into H_s^g.

Encoding T. As mentioned before, T is composed of time intervals $[t_1, t_2, \cdot, t_{|T|}]$. Therefore, we first design the time interval embedding module to encode each time interval t_i. Due to the continuous nature of time intervals, it is unrealistic to embed every time interval. Thanks to the weekly periodicity, we only need to consider time intervals of a week. In particular, we can regard the time interval as a combination of "day-in-week" (1 to 7 for Monday–Sunday) and "time-in-day" (1 to $\frac{24\times60}{\Delta t}$ if the size of a time interval is Δt minutes). As shown in Figure 3, the time interval (Tuesday 8:55-9:00Am) corresponds to 2 and $\frac{9\times60}{\delta t}$ for "day-in-week" and "timein-day", respectively. Next, we use one-hot codes to represent them, denoted as $O_w \in \mathbb{R}^7$ and $O_d \in \mathbb{R}^{\frac{24\times60}{\delta t}}$. Later, we use two fully connected neural networks (FC_1 and FC_2) to embed O_w and O_d into dense vectors $H_w \in \mathbb{R}^{d_w}$ and $H_d \in \mathbb{R}^{d_d}$, which are then concatenated to denote the time interval's representation. To get the global temporal representation $H_t^g \in \mathbb{R}^{|T|\times d_t}$ of the sequence T, we need to consider the sequential influence among different time intervals. Therefore, we apply the effective recurrent neural network LSTM [9] to further encode each time interval's representation, which can be formulated as follows:

$$H_t^g[i] = LSTM(concat(H_w[i], H_d[i]), H_t^g[i-1]) \tag{2}$$

where the output of LSTM at the i-th step (i.e., $H_t^g[i] \in \mathbb{R}^{d_t}$) could incorporate sequential information of previous time intervals.

Encoding G given T. The aim of encoding G is to generate global spatial representations $H_s^g \in \mathbb{R}^{|\mathcal{P}|\times d_s}$ for all partitions in G. At first, we propose the module *Node Embedding* to generate each partition's embedding, similar to time interval embedding. In particular, we first use one-hot embedding $O_g[k] \in \mathbb{R}^{|\mathcal{P}|}$ to represent the partition P_k and then apply a fully connected neural network FC_3 to embed $O_g[k]$ into a dense vector $H_g[k] \in \mathbb{R}^{d_g}$. However, the significant point is to consider the correlation between different partitions given the graph structure, which cannot be captured by H_g. To address this issue, we first build the adjacent matrix A to represent the graph, where $A[k_i, k_j] = 1$ if there is an edge in $\mathcal{E}$ linking the two partitions P_{k_i} and P_{k_j}, and then apply the currently effective graph neural network GCN [14] to further encode H_s. Notably, to incorporate the given temporal condition T, we first generate a new adjacent matrix $\hat{A}_i$ for each time interval t_i, where $\hat{A}_i[k_i, k_j] = A[k_i, k_j] \odot FC_4(H_t^g[i])$ means using the fully connected neural network FC_4 to generate different weight values for different new adjacent matrices at different time intervals. Later, we respectively use GCN to further encode H_g into $H_g^{t_i}$ based on $\hat{A}_i$ at each time interval t_i, which leads to a sequence of spatial representations $[H_g^{t_1}, \cdots, H_g^{t_{|T|}}]$. At last, we compute their mean value as the global spatial representation $H_s^g = Avg([H_g^{t_1}, \cdots, H_g^{t_{|T|}}])$, where $Avg(\cdot)$ is the average pooling function and the partition P_k corresponds to the representation $H_s^g[k] \in \mathbb{R}^{d_s}$.

4.2 Spatio-temporal Encoder

Actually, the goal of the spatio-temporal encoder is to convert raw speed histograms $hist_{L_k}^T$ into hidden representations $\hat{H}_{L_k}^T$ for each partition $\mathcal{P}_k = \langle L_k, E_k \rangle$. Notably, global spatial and temporal contexts should be taken into account. To achieve this goal, as shown in Figure 4, we first generate spatial representations $H_s^l[k]$ and temporal representations $H_t^l[k]$ from $hist_{L_k}^T$, and then fuse them with global-spatial representation $H_s^g[k]$ and global temporal representation H_t^g, respectively. At last, the fused spatial and temporal representations are further fused with two attention modules: the spatial-to-temporal (*S2T Attention*) and the temporal-to-spatial (*T2S Attention*), which would lead to the final spatio-temporal representation $\hat{H}_{L_k}^T$. Next, according to Figure 2, we will introduce the whole procedure, which includes three stages as follows.

*phase 1 [**local encoder**]:* In this stage, we aim to extract spatial and temporal representations from the local features $hist_{L_k}^T$ for each given partition. In particular, $hist_{L_k}^T$ can be regarded as a set of $|L_k| \times |T|$ histograms, where $|L_k|$ and $|L|$ denote the number of links and time intervals, respectively. Therefore, to generate the local spatial representation $H_s^l[k] \in \mathbb{R}^{|\mathcal{P}_k|\times d_s}$, we need to compress the temporal dimension, so we apply an *LSTM* module to capture the temporal sequence characteristic and then merge all elements in the temporal sequence with the average pooling operator, which is formulated as follows:

$$H_s^l[k] = Avg(LSTM(hist_{L_k}^T, dim = 1), dim = 2) \tag{3}$$

In contrast, we need to compress the spatial dimension for generating the local temporal representation $H_t^l[k] \in \mathbb{R}^{|T|\times d_t}$, so we apply an *GCN* module to capture the spatial graph characteristic and then merge all elements in the spatial graph with the average pooling operator, which is formulated as follows:

$$H_t^l[k] = Avg(GCN(hist_{L_k}^T, dim = 2), dim = 1) \tag{4}$$

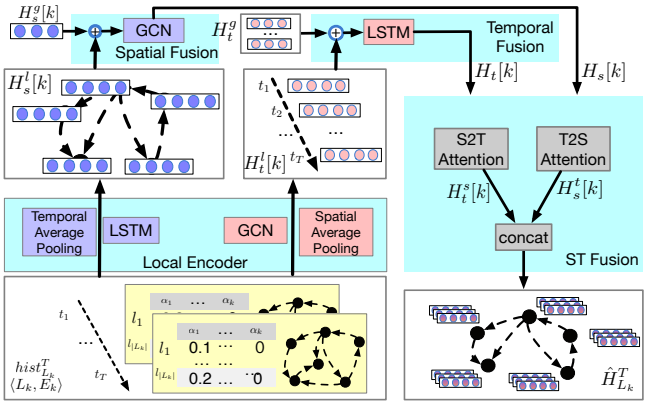

Figure 4: The Framework of Spatio-temporal Encoder

*phase 2 [**spatial fusion & temporal fusion**]:* In the spatial fusion, we aim to further encode each link's representation $H_s^l[k]$ by incorporating the corresponding partition's global code $H_s^g[k]$. In partition, we add the global code into each link's code, and then use a *GCN* module to capture the local correlation between different links and generate the fused representation $H_s[k]$, which is formulated as follows:

$$H_s[k] = GCN(H_s^l[k] \oplus H_s^g[k]) \tag{5}$$

As for temporal fusion, we consider each time interval's global code $H_t^g[t_i]$ for each local temporal code $H_t^l[k][t_i]$. Hence, we first add $H_t^g[t_i]$ into $H_t^l[k][t_i]$, and then apply an *LSTM* module to capture the sequential characteristic, which is formulated as follows:

$$H_t[k][t_i] = LSTM(H_t^l[k][t_i] \oplus H_t^g[t_i]), 1 \le t_i \le |T| \tag{6}$$

*phase 3 [**ST fusion**]:* According to the previous steps, there are two representations $H_s[k] \in \mathbb{R}^{|L_k| \times d_s}$ and $H_t[k] \in \mathbb{R}^{|T| \times d_t}$ for each partition P_k, requiring a fusion operator to generate the final spatio-temporal tensor $\hat{H}_{L_k}^T \in \mathbb{R}^{|L_k| \times |T| \times (d_s + d_t)}$. One intuitive fusion method is to concatenate them with the formula $\hat{H}_{L_k}^T[l_i, t_j] = concatenate(H_s[k][l_i], H_t[k][t_j])$ for each link l_i at each time interval t_j. However, this method is too straight to capture the spatio-temporal correlation between the two representations. To address this issue, we leverage the attention mechanism to convert one representation into the other representation's space, which is implemented in the two modules *T2S Attention* and *S2T Attention*. In *T2S Attention*, we aim to use all temporal representations $H_s^t[k]$ to enhance each spatial representation $H_s[k][l_i]$. In particular, we first compute the attention score between the spatial representation $H_s[k][l_i]$ and each temporal representation $H_t[k][t_j]$, and then use the scores to calculate the weighted sum of $H_t[k]$, which would be added with $H_s[k][l_i]$ to generate the enhanced result $H_s^t[k]$. The whole procedure can be formulated as follows:

$$\begin{aligned} Score_s[l_i, t_j] &= (H_s[k][l_i] \times W_q^s) \times (H_t[k][t_j] \times W_k^s)^T \\ Weight_s[l_i] &= Softmax_{j \in [1,|T|]}(\{Score_s[l_i, t_j]\}) \in \mathbb{R}^{|T|} \\ H_s^t[k][l_i] &= H_s[k][l_i] + \sum_{j=1}^{|T|} Weight_s[l_i][t_j] \cdot (H_t[k][t_j] \times W_v^s) \end{aligned} \tag{7}$$

where $W_q^s \in \mathbb{R}^{d_s \times d_h}$, $W_k^s \in \mathbb{R}^{d_t \times d_h}$ and $W_v^s \in \mathbb{R}^{d_t \times d_s}$ are learnable matrix parameters aligning the dimensions of different representations. In contrast, the goal in *S2T Attention* is to use all spatial representations $H_s[k]$ to enhance each temporal representation $H_t[k][t_j]$, which can be formulated as follows:

$$\begin{aligned} Score_t[t_j, l_i] &= (H_t[k][t_j] \times W_q^t) \times (H_s[k][l_i] \times W_k^t)^T \\ Weight_t[t_j] &= Softmax_{i \in [1,|L_k|]}(\{Score_t[t_j, l_i]\}) \in \mathbb{R}^{|L_k|} \\ H_t^s[k][t_j] &= H_t[k][t_j] + \sum_{i=1}^{|L_k|} Weight_t[t][l_i] \cdot (H_s[k][l_i] \times W_v^t) \end{aligned} \tag{8}$$

At last, we concatenate two enhanced results into the final spatio-temporal representation $\hat{H}_{L_k}^T[l_i, t_j] = concat(H_s^t[k][l_i], H_t^s[k][t_j])$.

4.3 Histogram Generator

We have encoded all features (a.k.a., global and local features) and generated the associated representation $\hat{H}_{L_k}^T$ for each partition P_k. Subsequently, we merge them into the whole histogram tensor $\hat{H}_L^T = concatenate(\{\hat{H}_{L_k}^T\}_{k=1}^{|\mathcal{P}|}) \in \mathbb{R}^{|L| \times |T| \times (d_s + d_t)}$. Next, we need to generate the estimated histograms $\hat{hist}_{L_k}^T$ based on $\hat{H}_{L_k}^T$. As shown in Figure 2, we design the module *Histogram Generator* to achieve the goal. In particular, this generation procedure is composed of the following two steps:

*phase 1 [**estimating for all links**]:* To make the estimation more efficient (i.e., batch operators on GPUs make the tensor computation more efficient), we apply a two-layer fully connected neural network to batch estimate all speed histograms based on all links' spatio-temporal representations $\hat{H}_L^T$, and the associated process is formulated as:

$$\begin{aligned} \hat{hist''}_L^T &= ReLU(\hat{H}_L^T \times W_e^1 + b_e^1) \times W_e^2 + b_e^2 \in \mathbb{R}^{|L| \times |T| \times m} \\ \hat{hist'}_L^T[i, j] &= Softmax(\hat{hist''}_L^T[i, j, :]) \in \mathbb{R}^m \end{aligned} \tag{9}$$

where $W_e^1 \in \mathbb{R}^{(d_s + d_t) \times d_h}$ and $b_e^1 \in \mathcal{R}^{d_h}$ denote the weight matrix and bias vector parameters of the first layer, $W_e^2 \in \mathbb{R}^{d_h \times m}$ and $b_e^2 \in \mathcal{R}^m$ indicate the affiliated parameters of the second layer, $Relu(\cdot)$ and $Softmax(\cdot)$ are two activation functions. Notably, the usage of $Softmax(\cdot)$ is to make each speed histogram under the constraint of $\sum_{i=1}^m \alpha_i = 1$.

*phase 2 [**refining with label matrix** M_L^T]:* Our primary aim is to accurately impute data at missing positions, which are indicated by zero values in the label matrix M_L^T. Therefore, we need to refine the estimated result $\hat{hist'}_L^T$ into the final result $\hat{hist}_L^T$ with the label matrix M_L^T. Formally, the refinement process is formalized as follows:

$$\hat{hist}_L^T = (1 - M_L^T) \odot \hat{hist'}_L^T + M_L^T \odot hist_L^T \tag{10}$$

where $hist_L^T$ represents raw input histograms.

In summary, we denote the whole generation process as $\hat{hist}_L^T = Generator(\hat{H}_L^T, M_L^T | W_e^1, W_e^2, b_e^1, b_e^2)$ for simplicity.

5 MODEL LEARNING

In this section, we first focus on the auxiliary training task, aiming to leverage the auto-encoder model to supervise the learning of spatio-temporal representations $\hat{H}_L^T$. Next, we discuss how to train **Nuhuo** with training data on the two tasks.

5.1 Auto-Encoder Branch

Motivation. Rethinking the imputation problem, our goal is to estimate missing speed histograms based on observable speed histograms and other observable contexts (i.e., road network graph and time interval sequence). For simplicity, we denote observable histograms, missing histograms, and observable contexts as x_o, x_m and c_e respectively. Therefore, the encoder-generator framework of our model **Nuhuo** is equivalent to first encoding both x_o and c_e into hidden representations $z = \langle z_o, z_m \rangle$, and then generating $x = \langle x_o, x_m \rangle$ based on z, where z_o and z_m correspond to the hidden representations of x_o and x_m respectively. In particular, the optimization target of **Nuhuo** can be formulated as follows:

$$\begin{aligned} &\text{argmin}_\theta - \log p_\theta(x, z|x_o, c_e) \\ &= -(\underbrace{\log p_\theta(z|x_o, c_e)}_{encoder} + \underbrace{\log p_\theta(x|z)}_{\text{generator}}) \end{aligned}$$

However, there are no explicit supervised signals for optimizing the encoder part $\log p_\theta(z|x_o, c_e)$ in the training data, which makes it ineffective to generate the hidden representation z. To address this issue, we apply the auto-encoder framework [28] to generate supervised signal $z' = \langle z'_m, z'_o \rangle$ from $x = \langle x_m, x_o \rangle$. In particular, the framework consists of an encoder module En (i.e., $\langle z' \rangle = En(x)$) and a decoder module De (i.e., $x = De(z')$), where we can train the two modules with the loss of reconstruction on histograms. Therefore, the optimization target of **Nuhuo** is reformulated as follows:

$$\begin{aligned} &\text{argmin}_\theta - (\log p_\theta(x, z|x_o, c_e, z') + \log p_\gamma(z'|x) \\ &= -(\underbrace{\log p_\theta(z|x_o, c_e, z')}_{\text{enhanced encoder}} + \underbrace{\log p_\theta(x|z)}_{\text{generator}} + \underbrace{\log p_\gamma(z'|x)}_{\text{encoder } En} + \underbrace{\log p_\theta(x|z'))}_{\text{decoder } De} \end{aligned}$$

In particular, the raw encoder $p_\theta(z|x_o, c_e)$ is enhanced by the supervised signal z', which leads to the new encoder $p_\theta(z|x_o, c_e, z')$. Notably, the decoder $De(\cdot)$ is the same as the generator, and both of them aim to convert representations from the same hidden space into the same speed histograms.

Encoder module. As shown in Fig 2, the encoder module En is designed to covert each ground-truth histogram $\bar{hist}_L^T[l_i, t_j] \in \mathbb{R}^m$ into a hidden representation $\bar{H}_L^T[l_i, t_j] \in \mathbb{R}^{d_s+d_t}$, and we apply a two-layer fully connected neural network to achieve a goal, which is formulated as:

$$\bar{H}_L^T = ReLU(\bar{hist}_L^T \times W_a^1 + b_a^1) \times W_a^2 + b_a^2 \tag{11}$$

where $W_a^1 \in \mathbb{R}^{m \times d_h}$ and $b_a^1 \in \mathbb{R}^{d_h}$ denote the weight matrix and bias vector parameters of the first layer, $W_a^2 \in \mathbb{R}^{d_h \times (d_s+d_t)}$ and $b_a^2 \in \mathbb{R}^{(d_s+d_t)}$ indicate the affiliated parameters of the second layer.

Decoder Module. As mentioned before, the decoder should be the same as the generator in the main branch. Hence, we reuse the generator module *Histogram Generator* in Sec. 4.3 to decode the new estimated histograms $\tilde{hist}_L^T$ from the hidden representations $\bar{H}_L^T$. Similarly, the whole decoder procedure can be denoted as $\tilde{hist}_L^T = Generator(\bar{H}_L^T, M_L^T | W_e^1, W_e^2, b_e^1, b_e^2)$.

5.2 Model Training

In this section, we will first introduce the overall learning objective, which is composed of three losses. Afterwards, we will illustrate the whole training algorithm.

Algorithm 1: Model Learning for **Nuhuo**

Input: training inputs $\langle L, E \rangle, T, hist_L^T, M_L^T$, training labels $\bar{hist}_L^T$, five parts of the whole model (local encoder $\mathcal{M}_{lc}$, global encoder $\mathcal{M}_{gl}$, fusion module $\mathcal{M}_{fs}$ (i.e., three fusion sub-modules), auto-encoder $\mathcal{M}_{ae}$, auto-decoder (i.e., histogram generator) $\mathcal{M}_{ge}$, learning rate lr, training epochs ep, batch size bs, loss weight λ.

Output: parameters $\theta_{lc}, \theta_{gl}, \theta_{fs}, \theta_{ae}, \theta_{ge}$, for the five parts $\mathcal{M}_{lc}$, $\mathcal{M}_{gl}$, $\mathcal{M}_{fs}$, $\mathcal{M}_{ae}$, $\mathcal{M}_{ge}$

generate $\langle \mathcal{P}, \mathcal{E} \rangle$ based on $\langle L, E \rangle$;
partition $\langle L, E \rangle, hist_L^T, M_L^T$ and $\bar{hist}_L^T$ into local features $\{\langle L_k, E_k \rangle, hist_{L_k}^T, M_{L_k}^T\}_{k=1}^{|\mathcal{P}|}$ and labels $\{\bar{hist}_{L_k}^T\}_{k=1}^{|\mathcal{P}|}$;
initialize $\theta_{lc}, \theta_{gl}, \theta_{fs}, \theta_{ae}, \theta_{ge}$ with normal distribution;
for $i \leftarrow 1 \cdots ep$ **do**
  $\theta_{lc}, \theta_{gl}, \theta_{fs}, \theta_{ae}, \theta_{ge} \leftarrow ModelTrain(\langle \mathcal{P}, \mathcal{E} \rangle, T,$
  $\{\langle L_k, E_k \rangle, hist_{L_k}^T, M_{L_k}^T\}_{k=1}^{|\mathcal{P}|}, \{\bar{hist}_{L_k}^T\}_{k=1}^{|\mathcal{P}|}, lr, bs, \lambda);$
  using $\theta_{lc}, \theta_{gl}, \theta_{fs}, \theta_{ae}, \theta_{ge}$ to respectively update $\mathcal{M}_{lc}$, $\mathcal{M}_{gl}$, $\mathcal{M}_{fs}$, $\mathcal{M}_{ae}$ and $\mathcal{M}_{ge}$;

Function ModelTrain

Input: $\langle \mathcal{P}, \mathcal{E} \rangle, T, \{\langle L_k, E_k \rangle, hist_{L_k}^T, M_{L_k}^T, \bar{hist}_{L_k}^T\}_{k=1}^{|\mathcal{P}|}, lr, bs, \lambda$

training iterations $TI = \lfloor \frac{|\mathcal{P}|}{b_s} \rfloor$;
shuffle partitions $\{\langle L_k, E_k \rangle, hist_{L_k}^T, M_{L_k}^T, \bar{hist}_{L_k}^T\}_{k=1}^{|\mathcal{P}|}$;
all global representations $H_s^g, H_t^g \leftarrow \mathcal{M}_{gl}(\langle \mathcal{P}, \mathcal{E} \rangle, T)$;
for $i \leftarrow 1 \cdots TI$ **do**
  $k_1 = (i-1)b_s + 1, k_2 = i \times b_s$;
  fetch the batch $\{\langle L_k, E_k \rangle, hist_{L_k}^T, M_{L_k}^T, \bar{hist}_{L_k}^T\}_{k=k_1}^{k_2}$;
  fetch the global representations $\{H_s^g[k], H_t^g[k]\}_{k=k_1}^{k_2}$;
  $\{H_s^l[k], H_s^g[k]\}_{k=k_1}^{k_2} \leftarrow \mathcal{M}_{lc}(\{\langle L_k, E_k \rangle, hist_{L_k}^T\}_{k=k_1}^{k_2})$;
  $\{\hat{H}_{L_k}^T\}_{k=k_1}^{k_2} \leftarrow \mathcal{M}_{fs}(\{H_s^g[k], H_t^g[k], H_s^l[k], H_t^l[k]\}_{k_1}^{k_2})$;
  $\{\hat{hist}_{L_k}^T\}_{k=k_1}^{k_2} \leftarrow \mathcal{M}_{ge}(\{\hat{H}_{L_k}^T\}_{k=k_1}^{k_2})$;
  $\{\tilde{hist}_{L_k}^T\}_{k=k_1}^{k_2} \leftarrow \mathcal{M}_{ge}(\mathcal{M}_{ae}(\bar{H}_{L_k}^T\}_{k=k_1}^{k_2}))$;
  $\mathcal{L} \leftarrow$ using Equations 12–15 and λ;
  $\theta_{lc}, \theta_{gl}, \theta_{fs}, \theta_{ae}, \theta_{ge} \leftarrow AdamOpt(\mathcal{L}, lr)$;
return $\theta_{lc}, \theta_{gl}, \theta_{fs}, \theta_{ae}, \theta_{ge}$;

Overall learning objective. As shown in Fig 2, the first loss $loss_1$ is designed to measure the difference between the estimated histogram $\hat{hist}_L^T$ and the ground-truth histogram $\bar{hist}_L^T$. Considering that each speed histogram indicates a speed distribution, we leverage the KL-divergence function to compute the distance, which is defined as follows:

$$\begin{aligned} loss_1 &= \frac{\sum_{l \in L} \sum_{t \in T} KL(\hat{hist}_l^t || \bar{hist}_l^t)}{|L| \times |T|} \\ &= \frac{\sum_{l \in L} \sum_{t \in T} \sum_{1 \le i \le m} (\hat{hist}_l^t[i] \log \frac{\hat{hist}_l^t[i]}{\bar{hist}_l^t[i]})}{|L| \times |T|} \end{aligned} \tag{12}$$

where each ground-truth histogram $\bar{hist}_l^t$ corresponds to the estimated result $\hat{hist}_l^t$. Similarly, the third loss $loss_3$ is proposed to measure the distance between the reconstructed histogram $\tilde{hist}_L^T$ and the ground-truth histogram $\bar{hist}_L^T$ in the auto-encoder branch. Therefore, we also leverage the KL-divergence function to compute

it, denoting as follows:

$$loss_3 = \frac{\sum_{l \in L} \sum_{t \in T} KL(\tilde{hist}_l^t || \bar{hist}_l^t)}{|L| \times |T|}. \quad (13)$$

At last, the second loss $loss_2$ is introduced to supervise the learning of hidden representations $\hat{H}_L^T$ using the auto-encoder branch. Hence, we need to measure the similarity between $\hat{H}_L^T$ and the encoded result $\bar{H}_L^T$ in the auto-encoder branch. In particular, we leverage the MSE(Mean-Square Error) function to compute the similarity, which is denoted as follows:

$$loss_2 = MSE(\hat{H}_L^T, \bar{H}_L^T) = \frac{\sum_{l \in L} \sum_{t \in T} \sum_i (\hat{H}_L^T[i] - \bar{H}_L^T[i])^2}{|L| \times |T| \times (d_s + d_t)}. \quad (14)$$

For simplicity, we denote the sum of $loss_2$ and $loss_3$ as the auxiliary loss $\mathcal{L}_{au} = loss_2 + loss_3$ due to that the two losses are introduced by the auxiliary task in the auto-encoder branch. In contrast, we denote $loss_1$ as the main loss $\mathcal{L}_{ma} = loss_1$. Therefore, we can compute the overall learning objective as follows:

$$\mathcal{L} = \mathcal{L}_{ma} + \lambda \cdot \mathcal{L}_{au} \quad (15)$$

where λ is the hyper-parameter to control the weights of the kinds of losses.

Offline training. Algorithm 1 outlines the training process. At first, we extract different features from given inputs. Specifically, the whole link-connected graph $\langle L, E \rangle$ is divided into $|\mathcal{P}|$ parts, where the k-th part includes local features $\langle L_k, E_k \rangle hist_{L_k}^T, M_{L_k^T}$ and labels $\bar{hist}_{L_K}^T$. Next, we initialize all parameters for the whole model with a normal distribution (lines 1-3). Then we iteratively train the whole model with the given epochs ep (lines 4-7). *ModelTrain* explains the training process for each epoch. We first compute the training iterations TI based on a given batch size b_s, and then shuffle all partitions. In addition, we leverage the global encoder $\mathcal{M}_{gl}$ to encode global features $\langle \mathcal{P}, \mathcal{E} \rangle, T$ into representations H_s^g, H_t^g (line 1-3). In each iteration, we collect b_s partitions and their corresponding global representations for generating estimated histograms. Moreover, we use the above objective function to compute the total loss $\mathcal{L}$ and utilize Adam Optimizer [13] to optimize all parameters by minimizing $\mathcal{L}$ (lines 5-13).

6 EXPERIMENTS

In this section, we aim to rigorously assess the effectiveness, efficiency, and scalability of our method, substantiating our underlying motivations and claims. Specifically, we explore the model's handling of **regionality** by examining the effectiveness of the partition number $|\mathcal{P}|$ in Section 6.6. Subsequent ablation studies are conducted to ascertain the model's capacity to simultaneously capture **regionality** and **proximity**, as well as its ability to address **volatility** through the auto-encoder module, as outlined in Section 6.3. Finally, we investigate the impact of **data sparsity** in Section 6.5, culminating in a comparative analysis of the robustness and scalability of our method against existing methods.

6.1 Experimental Setup

Datasets. (1) **LCG.** We respectively construct **LCG** for two road networks (i.e., *Chengdu Road Network (CD)* and *Xi'an Road Network (XA)*) extracted from OpenStreetMap [2]. In particular, *CD* includes 16,874 links and 50,263 edges, and *XA* includes 12,028 links and 36,304 edges. We divide each graph into disjoint partitions. As shown in Table 1, we vary the number $|\mathcal{P}|$ across the set $\{1, 32, 64, 128, 256\}$ to determine the optimal number of partitions, where $|\mathcal{P}| = 1$ means we do not partition the whole road network.

Table 1: Parameter settings

Parameters	$\|\mathcal{P}\|$	ρ	m	Δt/minutes	T/hours
Values	1, 32, 64, **128**, 256	**0.2**, 0.4, 0.6, 0.8	2, 4, **8**, 16	5, 10, **15**, 30	**1**, 6, 12, 24

Speed Histogram. We extract speed information from taxi orders of Didi Chuxing [1], where thre are $5.8M$ and $3.4M$ trajectories on *CD* and *XA* [35, 36] respectively, both from 10/01/2016 to 11/30/2016. In particular, we divide the whole speed scale into m slots and then count the ratio of trajectories in each slot as the speed histogram during a time interval Δt. Following [20], we construct the historical average speed histogram as the ground truth for links with fewer than 5 speed records. To emulate scenarios of missing information, we selectively omit the actual speed histograms from a random subset of links that possess adequate speed records, thereby excluding those represented by historical average speed histograms. Consequently, for each time interval, we can adjust the missing rate $\rho = \frac{|L_m|}{|L_a|}$, where $|L_m|$ represents the number of links lacking speed histograms during that interval, and $|L_a|$ represents the number of links with actual speed histograms. This manipulation allows for a comprehensive evaluation of system scalability under varying conditions of data completeness. As shown in Table 1, our experiments cover various settings of the relative parameters ρ, m, Δt, and T, with their respective default values distinguished in bold.

(3) **Training, Validation and Test.** We divide a dataset into training, validation and test data by splitting the time intervals with the ratios of 75%:10%:15%. Specifically, the corresponding dates for training, validation, and test data are respectively [10/01/2016, 11/15/2016], [11/16/2016, 11/21/2016], [11/22/2016, 11/30/2016].

Baseline methods. We compare our models with eight methods:

- **Avg**: We extract all speeds from training data for each link, and then compute the traffic speed histogram as the result.
- **MLP** [8]: This a fully connected neural network based on the deep residual structure, which is a general model for the task of traffic prediction.
- **TCN** [15]: This is a deep learning method based on a temporal convolutional network, which is used to encode temporal correlations for estimating speed histograms.
- **MetaNet** [22]: This method applies three stacked RNNs combined with an attention mechanism and meta-knowledge to forecast the average speed.
- **GCWC** [10]: This is a graph neural network, which uses GCNs to capture spatial dependencies between the stochastic speeds.
- **SSTGCN** [20]: This is an end-to-end method, combining TCNs and GCNs to capture the spatio-temporal correlations.
- **STCPA** [30]: This is the current state-of-the-art method, which captures complex traffic correlations among the spatial and temporal dimensions via the attention mechanism.
- **PriSTI** [18]: This approach constitutes a general method for spatio-temporal imputation, utilizing the attention mechanism and the diffusion model to deduce missing values from noise.

Notably, the last seven methods are *learning-based* methods. For a fair comparison, we make their parameter scale (a.k.a., model size) similar to ours, which is shown in Sec. 6.4.

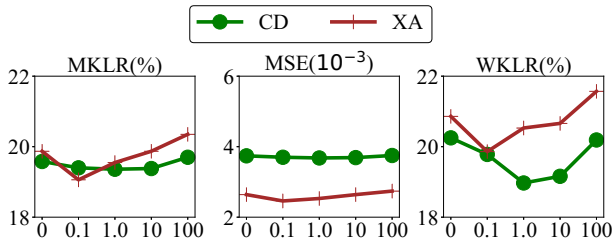

Figure 5: Loss on Validation Data vs. the Loss Weight λ

Evaluation metrics. We evaluate our proposed methods and baseline methods based on two metrics: MKLR (Mean Kullback-Leibler divergence Ratio) and MSE (Mean Square Error), which are widely adopted by the baselines we compare with. Specifically, suppose the ground truth is represented as $\mathbf{y} = \{y_i\}$ and the predicted result is denoted as $\hat{\mathbf{y}} = \{\hat{y}_i\}$, where $1 \leq i \leq N$, these metrics are computed as follows: $MKLR(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_i^N I_i KL(y_i||\hat{y}_i)}{\sum_i^N I_i KL(y_i||avg_i)}$, $MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^N (y^i - \hat{y}^i)^2$, where avg_i is the estimated result of the **Avg** method, I_i is an indicator of whether the corresponding estimated speed histogram needs to be evaluated. In particular, we set $I_i = 0$ if the corresponding edge is not covered by traffic data; otherwise, we set $I_i = 1$. In addition, the function $KL(\cdot||\cdot)$ computes the KL-divergence between two distributions, i.e., two speed histograms. The lower a KL-divergence value is, the more similar the two histograms are, indicating more accurate estimation or prediction. Hence, lower MKLR values indicate higher improvements over **Avg**. Furthermore, acknowledging the varying popularity of individual links within a road network, we assign a weight to each link based on the number of trajectories covering it. These weights are then utilized to compute a modified metric, the Weighted Kullback-Leibler Divergence Ratio (WKLR), denoted as $WKLR(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_i^N W_i I_i KL(y_i||\hat{y}_i)}{\sum_i^N W_i I_i KL(y_i||avg_i)}$, where W_i represents the weight reflecting the popularity of the link. This means that the more frequently a link is traversed, the greater its influence on the WKLR metric, thereby aligning the importance of each link with its actual use within the network.

Environment settings. All deep learning methods were implemented with PyTorch 1.0 and Python 3.6, and trained with a Tesla K40 GPU. The platform ran on Ubuntu 16.04 OS. In addition, we used Adam [13] as the optimization method with the mini-batch size of 100. The initial learning rate was 0.001.

6.2 Setting of Model's Hyper-parameters

we consider the following hyper-parameters: (1) the number (N) of graph neural network (GCN) layers; (2) the spatial and temporal embedding sizes (d_w, d_d, d_g), where $d_w = \frac{d_d}{10}$; (3) the settable-dimension size (d_h) of some hidden representations. (4) the settable-dimension sizes (d_s, d_t) of the spatial representations and the temporal representations. In particular, given a hyper-parameter, we first select its value range according to the experience under some constraints (e.g., the limitation of GPU memory). Then, we conduct experiments on the validation *CD* and *XA* to determine its optimal value. As shown in Fig. 8, we plot the MKLR for different hyper-parameters. In summary, we set each hyper-parameter with the value corresponding to the optimal performance as follows: (1) For *CD*, we have $N = 1$, $d_w = \frac{d_d}{10} = 20$, $d_g = 10$, $d_h = 150$, $d_s = 100$, $d_t = 5$. (2) For *XA*, we have $N = 1$, $d_w = \frac{d_d}{10} = 20$, $d_g = 10$, $d_h = 50$, $d_s = 20$, $d_t = 5$.

Table 2: Effectiveness Results on Test Data

	CD			XA		
Method	MKLR(%)	MSE(10^{-3})	WKLR(%)	MKLR(%)	MSE(10^{-3})	WKLR(%)
Avg	100	31.46	100	100	24.13	100
MLP	65.85	25.03	67.94	66.20	19.34	67.26
TCN	44.79	15.59	41.27	33.34	9.06	30.53
MetaNet	32.35	9.89	32.03	31.79	9.23	33.21
GCWC	32.09	10.23	32.61	34.78	10.55	35.32
SSTGCN	26.93	7.52	27.78	28.07	7.60	28.81
STCPA	24.44	5.59	24.05	26.01	4.76	25.91
PriSTI	26.79	7.22	26.20	27.99	5.58	28.32
NG	19.88	3.91	20.25	20.29	2.94	22.14
NL	20.38	3.90	19.76	21.24	2.90	21.17
NF	19.79	3.82	20.26	20.35	2.74	22.20
NA	19.58	3.74	21.67	19.87	2.64	22.79
Nuhuo	**19.36**	**3.68**	18.97	**19.06**	**2.46**	19.86

6.3 Effectiveness Comparison

Apart from comparing **Nuhuo** with baseline methods, we replace our **Nuhuo** by four variations, namely **NG**, **NL**, **NF** and **NA**, to evaluate the effectiveness of different parts of encodings in **Nuhuo**. In **NG**, we utilize fully connected neural networks (FCN) to generate global encodings in the global encoder. This is instrumental for evaluating the effectiveness of our proposed method, **Nuhuo**, particularly in terms of its ability to capture regionality. In **NL**, we use FCN to generate local encodings in the local encoder, which is designed to evaluate the effectiveness of **Nuhuo** for capturing the proximity. In **NF**, we leverage FNC to fuse global and local encodings, which is designed to demonstrate the significant advantage of fusion in capturing the spatially multi-view correlations. At last, we remove the auto-encoder branch in **NA** to evaluate its effectiveness. This modification also allows us to understand the specific influence of volatility on the overall performance.

Table 2 reports the evaluation results of all methods with the default settings of Δt, and we have the following observations:

(1) **Avg** is worse than deep learning-based methods because the latter can approximately fit any function. In addition, it is not sufficient for providing accurate predictions when historical data are sparse for the **Avg** method.

(2) When examining the outcomes of **NG**, **NL**, **NF**, **NA**, and **Nuhuo**, we observe that the local encoder is the most crucial component of **Nuhuo**, as **NL** performs worst on *CD* and *XA*. Furthermore, the significance of other modules varies depending on the dataset. For instance, for *XA*, the fusion module is more significant than the global module.

(3) **Nuhuo** exhibits the best performance across all metrics. For instance, **Nuhuo** outperforms the best existing methods (e.g., **SSTGCN**, **STCPA** and **PriSTI**) by 20% on MKLR for the test data of *CD*. The reason behind this success is that our method more effectively encodes and fuses the given global and local features.

(4) When comparing the MSE metric on *CD* and *XA*, the performance of all methods except **Avg** is better on *XA* than that on *CD*.

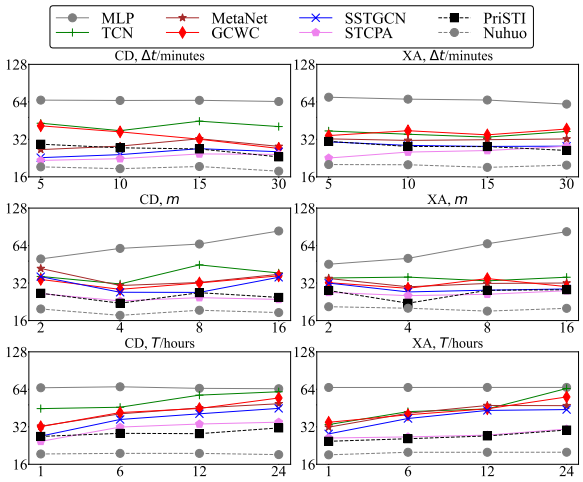

Figure 6: MKLR(%) vs. Variables Δt & m & T. (The title of each subfigure is labeled in the form of "A, B", where "A" and "B" respectively refer to one dataset and a kind of variable.)

This can be attributed to two primary factors: Firstly, the traffic speed distribution on *XA* exhibits greater complexity, including extreme values, when compared to *CD*. Consequently, **Avg** lacking the capability to adapt to input dynamics fails to capture these intricate distributions. Secondly, the speed distributions on *XA* exhibit stronger correlations with the provided features, making them better suited for handling by advanced deep-learning techniques.
(5) In comparing MKLR and WKLR metrics, most methods perform worse on WKLR, suggesting higher prediction difficulty for more popular links. However, our method shows a smaller performance decline on WKLR and can sometimes enhance performance, particularly for the dataset *CD*.

Furthermore, we evaluate the robustness of different models by respectively varying the values of Δt, m, and T in Figure 6. We have the following observations:
(1) As Δt increases, the metric of most methods exhibits a decline. The reason is that larger time intervals inherently contain more traffic data, leading to smoother distributions of traffic speed and fewer missing values. For instance, a one-hot histogram represented as $[0, 0, 1, 0]$ poses greater predictive challenges compared to a uniform distribution $[0.25, 0.25, 0.25, 0.25]$.
(2) Most methods exhibit fluctuating performance with varying slot size m due to the trade-off between data sparsity and distribution smoothness. Larger slot sizes increase data sparsity but also smooth out the distribution. While sparsity generally deteriorates performance, smoothness can enhance it. Specifically, **MLP**'s performance significantly declines, primarily affected by increased sparsity.
(3) Most methods' performance deteriorates with an increase in T, as more time intervals add complexity to prediction tasks. However, **MLP** and **Nuhuo** are less affected: the former due to its non-reliance on sequential features, and the latter due to its robustness in capturing such features.
(4) No matter how Δt, m and T change, our method **Nuhuo** consistently has the best performance. In particular, the performance disparity between **Nuhuo** and the three superior baselines, namely **SSTGCN**, **STCPA** and **PriSTI**, widens as the dataset complexity increases (e.g., the value of T increases). This outcome underscores the enhanced robustness of our proposed method, **Nuhuo**, in comparison to other advanced techniques. Furthermore, the incorporation of an auto-encode module ensures that **Nuhuo** yields stable encoding outcomes across diverse configurations.

Table 3: Efficiency of Test Result (Δt=15*min*)

	memory usage (Byte)		training time (minutes/ep)		estimation time (seconds/K)	
	CD	*XA*	*CD*	*XA*	*CD*	*XA*
Avg	1.5K	1.5K	-	-	0.48	0.31
MLP	8.6K	8.6K	1.52	0.80	1.13	1.06
TCN	7.4M	5.6M	2.00	0.95	1.16	1.08
MetaNet	4.8M	3.7M	6.46	4.32	4.14	3.76
GCWC	5.3M	3.9M	2.50	2.57	1.18	1.06
SSTGCN	5.5M	4.0M	3.97	3.28	2.45	2.48
STCPA	721K	721K	4.47	2.43	2.77	2.71
PriSTI	1.1M	1.1M	3.94	2.32	2.62	1.51
Nuhuo	2.6M	2.4M	3.22	1.44	1.62	1.25

6.4 Efficiency Comparison

For efficiency evaluation, we consider three metrics: memory usage, training time, and estimation time. Memory usage quantifies the amount of memory required to implement the respective methods, serving as an indicator of memory efficiency. Training time assesses the offline learning efficiency, particularly for methods based on neural networks. Specifically, we calculate the average time taken for one epoch for each method. Estimation time serves as a measure of online prediction efficiency. To elucidate, we employ various methods to estimate results for 1,000 samples and subsequently record the latency for each. The outcomes of these evaluations are presented in Table 3. From our observations, we deduce the following:
(1) Methods such as **TCN**, **MetaNet**, **GCWC**, **SSTGCN**, and **Nuhuo** demand more memory compared to their counterparts. This increased requirement can be attributed to their need to embed links. The size of these embeddings is directly proportional to the number of links. In contrast, other learning techniques merely require the loading of model parameters, the size of which remains consistent across different datasets.
(2) Owing to the computational intricacies of the graph neural network, methods like **GCWC**, **SSTGCN**, **PriSTI**, and **Nuhuo** necessitate longer training and estimation durations compared to **Avg**, **MLP**, and **TCN**. Notably, the presence of a loop operator within the **STCPA** framework results in significantly extended training and estimation times for **STCPA**.
(3) When compared with prevailing state-of-the-art methods, specifically **SSTGCN**, **STCPA** and **PriSTI**, our proposed model, **Nuhuo**, exhibits superior efficiency.

6.5 Scalability Comparison

We evaluate the scalability of various learning methods from two perspectives. Firstly, we adjust the size of the training dataset by extracting subsets that represent 25%, 50%, 75%, and 100% of the total data. Secondly, we alter the ratio ρ of links to ascertain the impact of training data sparsity on model performance. Specifically, ρ values are modified within the set $[0.2, 0.4, 0.6, 0.8]$, indicating

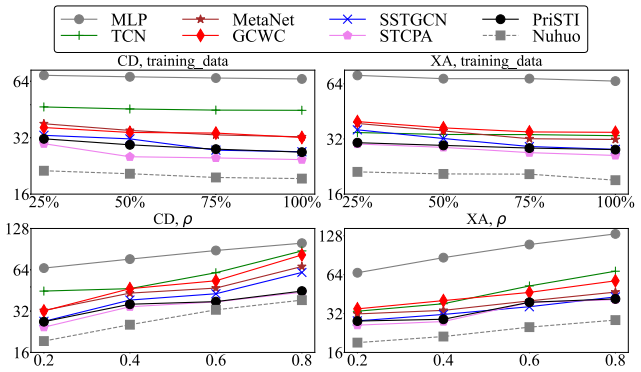

Figure 7: MKLR(%) vs. the Scalability. (The title of each subfigure is labeled in the form of "A, B", where "A" refers to the dataset and "B" refers to the scalability type.)

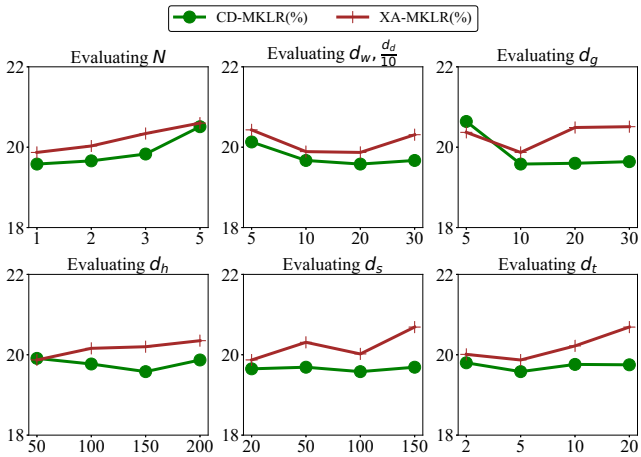

Figure 8: MKLR(%) vs. Hyper-parameters

the proportion of links that require imputation at each time interval, with 0.2 representing a 20% imputation need. These measures collectively provide insight into the robustness and adaptability of the learning methods under varying data quantity and quality scenarios. Observations derived from Figure 7 are as follows:

(1) All evaluated methods demonstrate improved performance with increased volumes of training data, indicating that larger datasets, which cover a more diverse array of scenarios, contribute to more effective and comprehensive model training. Conversely, an increase in the proportion of imputation (denoted as ρ) inversely affects performance, likely due to the diminished quality and representativeness of the training data.

(2) Among the methods evaluated, our method **Nuhuo** is more scalable and robust compared to other deep learning techniques. For instance, the MKLR of **STCPA** on the *CD* dataset escalates by $\frac{29.73-24.44}{24.44} = 21.64\%$ when restricted to 25% of the training data. In contrast, the increment for our method, **Nuhuo**, is a mere $\frac{21.32-19.36}{19.36} = 10.12\%$. This disparity underscores that while existing methods necessitate substantial data to maintain optimal performance, our method exhibits stability, a trait attributed to the integration of the auto-encoder module.

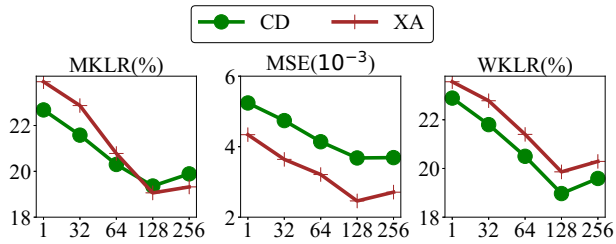

Figure 9: Loss on Validation Data vs. the Partition Number $|\mathcal{P}|$

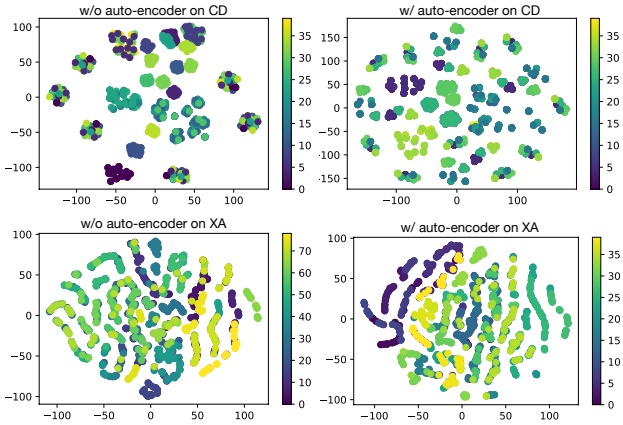

Figure 10: Visualization of t-SNE Embeddings learned w/ and w/o the auto-encoder module on *CD* and *XA*

(3) Our method, **Nuhuo**, consistently outperforms its counterparts. Notably, in several instances (e.g., reducing training data on datasets *CD* and *XA*), as the sampling rate diminishes, the performance gap between **Nuhuo** and other methods widens.

6.6 Effectiveness of Loss Weight and Partition

To fine-tune the loss weight λ, we vary it from 0 to 100 with the scale of 10 when training. We compute the two metrics for the validation data. The result is plotted in Figure 5, from which we find that the performance first improves with the increase of λ but is much worsened when exceeding a certain threshold. This is because there is a trade-off between the main loss and the auxiliary loss. Based on the majority voting rule, the best values of λ for *CD* and *XA* are 1.0 and 0.1 respectively, set as the default values in subsequent experiments.

Additionally, we further optimize the partition number $|\mathcal{P}|$. The outcomes are illustrated in Figure 9, and they align closely with the observations made for λ. In essence, the partition number signifies the granularity of regions, thereby affecting the balance between global and local correlations. Specifically, performance is notably inferior in scenarios without partitions (i.e., when $|\mathcal{P}| = 1$), affirming the significant role that partitioning plays in enhancing the effectiveness of the method. For clarity, in the default configuration, both datasets are optimally divided into 128 partitions.

6.7 Encoding Visualization

To further analyze the effectiveness of encoding modules within **Nuhuo**, we employ t-SNE [26] to visualize the distribution of spatio-temporal encodings pertaining to prediction results on datasets *CD*

and *XA*. Specifically, we select 40 links from each road network and subsequently extract their corresponding spatio-temporal encodings from $\hat{H}_L^T$. These encodings, aggregated by the hour, are averaged to represent the encoding for that specific hour, resulting in 40×24 encodings per dataset. The t-SNE embeddings for these encodings are then computed and depicted in Figure 10. To further elucidate the impact of the auto-encoder module, we also present embeddings derived without the auxiliary task. Observations from the figure include:

(1) The disparity in traffic states across different links on *CD* is more pronounced than on *XA*. Consequently, most learning methods exhibit suboptimal performance on *CD*, as evidenced in Table 2. This suggests that traffic states exhibit greater variability across different links within the *CD* dataset.

(2) Leveraging the auto-encoder module enables our method to more effectively differentiate between links. This enhancement bolsters the method's resilience, particularly in addressing the instability of encodings arising from missing values.

7 RELATED WORK

7.1 Network-based Traffic Estimation

There are two network-based traffic estimation problems: forecasting and imputation. In particular, the forecasting task aims to estimate future traffic based on historical observations, while the imputation problem means the estimation of missing traffic, especially the traffic speed.

Traffic forecasting. Deep learning has emerged as the preeminent choice due to its exceptional capacity for approximating arbitrary functions, with the central concern being the effective encoding of spatio-temporal traffic features. Notably, the work of Li et al. [16] has advocated the utilization of Graph Convolution Networks (GCN) for modeling spatial dependencies and recurrent neural networks (RNN) to capture temporal dynamics. However, it has been observed that such an approach may fall short in capturing global correlations and dynamic traffic data patterns. In response to this limitation, Guo et al. [7] and Jiang. [11] have incorporated attention mechanisms, Fang et al. [4] have introduced a multi-resolution temporal module along with a global correlated spatial module, and Pan et al. [22] have leveraged meta-learning to construct meta Graph Neural Network (GNN) and meta RNN models, enabling the capture of dynamic correlations in traffic data.

Traffic speed imputation. The work [10] is the first approach that studies stochastic speed estimation, utilizing GCNs to encode road network topology and estimate missing speed distributions. In particular, it applied a graph pooling layer, reducing graph dimensionality and demonstrating the superior ability of GCNs to capture complex spatial correlations. However, this method neglected temporal correlations, limiting its accuracy. In contrast, Muñiz-Cuza et al. [20] addressed this limitation by introducing a temporal component based on Temporal Convolutional Networks (TCNs) to capture temporal correlations between consecutive time intervals. However, both methods rely on GCN, which may not perform well under conditions of severe data sparsity. Hence, Xu et al. [30] and Liu et al.[18] incorporated the attention mechanism to dynamically capture intricate traffic data correlations, addressing data sparsity issues. Furthermore, to enhance the performance, [30] introduces a cost-effective imputation cycle consistency strategy to create cycles and generate approximate ground truth, while [18] apply a diffusion model to generate data from random noise. However, existing methods cannot fully exploit the hybrid spatio-temporal correlation at both region-level and link-level. These inspire us to design our efficient models.

7.2 Deep Learning for Spatio-temporal Data

As AI advances, deep learning is increasingly used in managing and mining spatio-temporal data. First, Recurrent Neural Networks (RNNs) are now applied to model trajectories. For instance, Wu et al. [29] demonstrated the superiority of RNNs over older models in predicting movements. Similarly, the study in [5] employs RNNs to effectively decipher user mobility patterns. Dong et al. [3] even designed a stacked RNN model to identify driving styles. Second, Convolutional Neural Networks (CNNs) play a role too. Song et al.[25] use them for simulating human mobility and transportation. In[37], they treat road network crowd density as images, using a deep network to forecast crowd flows. Third, Graph Neural Networks (GNNs) are used to tackle traffic prediction problems, considering road network structures in tasks like travel demand prediction [31] and traffic flow prediction [17, 22]. Fourth, attention mechanisms help capture complex spatio-temporal correlations. For example, Yuan et al. [36] use attention to jointly predict travel demands and traffic flows.

8 CONCLUSION

This paper has introduced **Nuhuo**, a novel and comprehensive framework designed to address the complex challenge of accurately imputing missing traffic speed histograms. We have identified and tackled three critical yet previously unexplored dimensions that are essential for achieving precise estimations: regionality and proximity, sparsity, and volatility. At first, we employed a global partition graph, enabling the capture of both regional and proximal correlations within the road network. Next, we introduced a disentangled feature encoding pipeline that effectively mitigates issues related to input sparsity by separately handling spatial and temporal dimensions. In addition, we incorporated a self-supervised learning task using an auto-encoder framework, thereby enhancing the stability and robustness of our encoding models. Extensive evaluations on two real-world datasets have confirmed the efficacy of **Nuhuo**. However, deploying **Nuhuo** effectively requires comprehensive historical traffic data to train the model, including various trajectory data. Therefore, our future work aims to employ few-shot learning methods to mitigate the limitations posed by scarce training data.

ACKNOWLEDGMENTS

This study is supported by NCS Pte Ltd through the Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU), and is also supported in part by a Singapore MOE AcRF Tier-2 grant MOE-T2EP20221-0015 and a Singapore MOE AcRF Tier-1 project RT6/23. Guoliang Li is supported by National Key R&D Program of China (2023YFB4503600), NSF of China (61925205, 62232009, 62102215), and Zhongguancun Lab.

REFERENCES

[1] 2021. GAIA. https://outreach.didichuxing.com/research/opendata/.
[2] 2021. OpenStreetMap. https://www.openstreetmap.org.
[3] Weishan Dong, Jian Li, Renjie Yao, Changsheng Li, Ting Yuan, and Lanjun Wang. 2016. Characterizing driving styles with deep learning. *CoRR* (2016).
[4] Shen Fang, Qi Zhang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2019. Gstnet: Global spatial-temporal network for traffic flow prediction. In *IJCAI*. 10–16.
[5] Qiang Gao, Fan Zhou, Kunpeng Zhang, Goce Trajcevski, Xucheng Luo, and Fengli Zhang. 2017. Identifying Human Mobility via Trajectory Embeddings.. In *IJCAI*. 1689–1695.
[6] Chenjuan Guo, Bin Yang, Jilin Hu, and Christian S. Jensen. 2018. Learning to Route with Sparse Trajectory Sets. In *ICDE*. 1073–1084.
[7] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI*, Vol. 33. 922–929.
[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.
[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[10] Jilin Hu, Chenjuan Guo, Bin Yang, and Christian S. Jensen. 2019. Stochastic Weight Completion for Road Networks Using Graph Convolutional Networks. In *ICDE*. 1274–1285.
[11] Yue Jiang, Xiucheng Li, Yile Chen, Shuai Liu, Weilong Kong, Antonis F. Lentzakis, and Gao Cong. 2024. SAGDFN: A Scalable Adaptive Graph Diffusion Forecasting Network for Multivariate Time Series Forecasting. In *ICDE*.
[12] George Karypis and Vipin Kumar. 1998. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of parallel and distributed computing* 48, 1 (1998), 71–95.
[13] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR(Poster)*.
[14] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR (Poster)*.
[15] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. 2017. Temporal convolutional networks for action segmentation and detection. In *CVPR*. 156–165.
[16] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *ICLR(Poster)* (2018).
[17] Yanan Li, Haitao Yuan, Zhe Fu, Xiao Ma, Mengwei Xu, and Shangguang Wang. 2023. ELASTIC: Edge Workload Forecasting based on Collaborative Cloud-Edge Deep Learning. In *WWW*. 3056–3066.
[18] Mingzhe Liu, Han Huang, Hao Feng, Leilei Sun, Bowen Du, and Yanjie Fu. 2023. PriSTI: A Conditional Diffusion Framework for Spatiotemporal Imputation. In *IEEE*. 1927–1939.
[19] Harvey J Miller. 2004. Tobler's first law and spatial analysis. *Annals of the association of American geographers* 94, 2 (2004), 284–289.
[20] Carlos Enrique Muñiz-Cuza, Nguyen Ho, Eleni Tzirita Zacharatou, Torben Bach Pedersen, and Bin Yang. 2022. Spatio-temporal graph convolutional network for stochastic traffic speed imputation. In *SIGSPATIAL*. 14:1–14:12.
[21] Bei Pan, Yu Zheng, David Wilkie, and Cyrus Shahabi. 2013. Crowd sensing of traffic anomalies based on human mobility and social media. In *SIGSPATIAL*. 344–353.
[22] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban traffic prediction from spatio-temporal data using deep meta learning. In *SIGKDD*. 1720–1730.
[23] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. Anytime Stochastic Routing with Hybrid Learning. *Proc. VLDB Endow.* 13, 9 (2020), 1555–1567.
[24] Simon Aagaard Pedersen, Bin Yang, Christian S Jensen, and Jesper Møller. 2023. Stochastic Routing with Arrival Windows. *ACM Transactions on Spatial Algorithms and Systems* 9, 4 (2023), 1–48.
[25] Xuan Song, Hiroshi Kanasugi, and Ryosuke Shibasaki. 2016. DeepTransport: Prediction and Simulation of Human Mobility and Transportation Mode at a Citywide Level.. In *IJCAI*. 2618–2624.
[26] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *JMLR* 9, 11 (2008).
[27] Yong Wang, Guoliang Li, and Nan Tang. 2019. Querying Shortest Paths on Time Dependent Road Networks. *PVLDB* 12, 11 (2019), 1249–1261.
[28] Yasi Wang, Hongxun Yao, and Sicheng Zhao. 2016. Auto-encoder based dimensionality reduction. *Neurocomputing* 184 (2016), 232–242.
[29] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. 2017. Modeling Trajectories with Recurrent Neural Networks. In *IJCAI*. 3083–3090.
[30] Qianxiong Xu, Sijie Ruan, Cheng Long, Liang Yu, and Chen Zhang. 2022. Traffic Speed Imputation with Spatio-Temporal Attentions and Cycle-Perceptual Training. In *CIKM*. 2280–2289.
[31] Ying Xu and Dongsheng Li. 2019. Incorporating graph attention and recurrent architectures for city-wide taxi demand prediction. *Geo-Inf* 8, 9 (2019), 414.
[32] Haitao Yuan and Guoliang Li. 2019. Distributed In-memory Trajectory Similarity Search and Join on Road Network. In *ICDE*. 1262–1273.
[33] Haitao Yuan and Guoliang Li. 2021. A Survey of Traffic Prediction: from Spatio-Temporal Data to Intelligent Transportation. *Data Sci. Eng.* 6, 1 (2021), 63–85.
[34] Haitao Yuan, Guoliang Li, and Zhifeng Bao. 2022. Route Travel Time Estimation on A Road Network Revisited: Heterogeneity, Proximity, Periodicity and Dynamicity. *PVLDB* 16, 3 (2022), 393–405.
[35] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2020. Effective Travel Time Estimation: When Historical Trajectories over Road Networks Matter. In *SIGMOD*. 2135–2149.
[36] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2021. An Effective Joint Prediction Model for Travel Demands and Traffic Flows. In *ICDE*.
[37] Junbo Zhang, Yu Zheng, Dekang Qi, Ruiyuan Li, and Xiuwen Yi. 2016. DNN-based prediction model for spatio-temporal data. In *SIGSPATIAL*. 1–4.