An Effective Joint Prediction Model for Travel Demands and Traffic Flows

Haitao Yuan†, Guoliang Li†, Zhifeng Bao‡, Ling Feng†
†Tsinghua University, China ‡RMIT University, Australia
†{yht16@mails.,liguoliang@,fengling@}tsinghua.edu.cn, ‡zhifeng.bao@rmit.edu.au

***Abstract*—In this paper, we study how to jointly predict travel demands and traffic flows for all regions of a city at a future time interval. From an empirical analysis of traffic data, we outline three desired properties, namely region-level correlations, temporal periodicity and inter-traffic correlations. Then, we propose a comprehensive neural network based traffic prediction model, where various effective embeddings or encodings are designed to capture the aforementioned properties. First, we design effective region embeddings to capture two forms of region-level correlations: spatially close regions have similar embeddings, and regions with similar properties (e.g., the number of POIs and number of roads in a region) other than locations have similar embeddings. Second, we extract the "day-in-week" and "time-in-day" and utilize the temporal periodicity in designing the embeddings for time intervals. Third, we propose an effective encoding for past traffic data which captures two forms of inter-traffic correlations – the correlation between past and future traffic, and the correlation between travel demands and traffic flows within past traffic data. Extensive experiments on two real datasets verify the high effectiveness of our model.**

I. Introduction

The prevalence of global positioning services enables a massive amount of traffic data being collected, which in turn helps predict future traffic. Traffic prediction is crucial to many transportation services such as taxi dispatching, route planning, and congestion avoidance [35], [33], [24]. There have been considerable studies on this topic, such as traffic flow prediction [11], [30], [28], [16], [7], [13], [21], [12], [19] and travel demand prediction [31], [8], [27], [1], [5], [18].

To our best knowledge, existing methods focus on the prediction for only a single type of traffic data. For example, Yao et al. [31] and Geng et al. [8] propose different models for travel demands prediction, while Zhang et al. [34] and Pan et al. [21] design different models for traffic flows prediction. We argue that the correlation between travel demands and traffic flows should not be negligible, and instead, utilizing such correlations can further enhance the prediction accuracy for each single type of traffic data. For example, if there are increasing travel demands in a region, the traffic flows in that region would accordingly increase in a near future.

Therefore, we study how to jointly predict travel demands and traffic flows, with a focus on region-level traffic prediction – given an input that consists of a city (composed of a set of disjoint regions), a future time interval and some past traffic data, we aim to estimate both the travel demands and traffic flows during a future time interval for all regions of the city.

From our investigation real-world traffic data in Example 1, we find three properties that a desirable joint prediction model should capture: 1) inter-traffic correlations between different types of traffic data, 2) region-level correlations, and 3) temporal periodicity of traffic data.

Example 1: Figure 1 presents the travel demands and traffic flows of several regions in *Chengdu*, one of the five largest cities in China. Here, *Chengdu* is split into 64 regions, each with a size of $1km \times 1km$ (Figure 1 (a)). Next, we elaborate each of aforementioned properties.

(1) Inter-traffic Correlations: In Figure 1 (b), we first plot the travel demands and traffic outflows for region 7 on October 1st. We can find that outflows at a later moment are influenced by demands earlier. Next, we plot travel demands and traffic inflows for region 54 on October 1st, and we can find that inflows earlier can influence demands at a later moment.

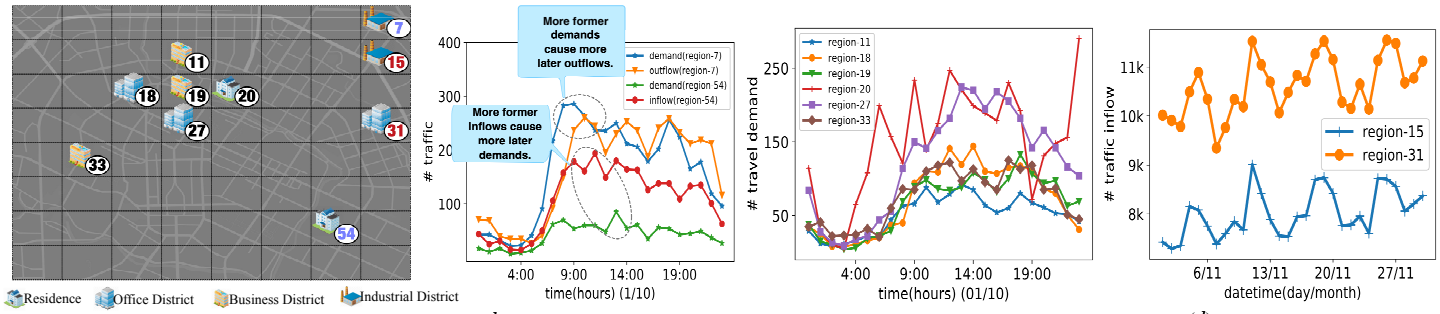
(2) Region-level Correlations – Figure 1 (c) shows the travel demands of six regions (with IDs 11, 18, 19, 20, 27 and 33) on October 1st, where 11, 18, 20 and 27 are neighbors of 19. We observe that regions that have similar traffic can be correlated in two aspects: i) they are similar location-wise, e.g. regions 19 and its neighbors 11 and 18. ii) they are similar at dimensions other than location, e.g. regions 19 and 33 both are business districts, and have similar travel demands.

(3) Temporal Periodicity: Figure 1 (d) shows the traffic inflows of regions 15 and 31, November 1st -30th, where a strong weekly periodicity is exhibited.

Astute readers may wonder the possibility to extend existing prediction methods primarily designed for a single type of traffic data to achieve a joint prediction. Unfortunately, the answer is no because: (1) They simply treat past traffic data as sequences and only a coarse-grained trend of traffic data is learned from such sequences. However, correlations among regions and temporal periodicity should be embedded in the spatio-temporal representations of regions and time intervals. (2) The inter-traffic correlations among given past traffic data have never been explicitly modeled or leveraged.

Therefore, we propose a comprehensive model called `DeepTP`. `DeepTP` can effectively encode the above three properties in a fine-grained manner and result in a unified spatio-temporal representation. In particular, we encode the given input as follows.

Region embedding (to capture region-level correlations): Three spatial graphs, (*connectivity-aware graph*, *neighbor-aware graph* and *similarity-aware graph*), are constructed to



(a) 64 regions of Chengdu(8×8) (b) the inter-traffic correlations (c) the region-level correlation (d) the temporal periodicity

Fig. 1. The Findings about Traffic Prediction

achieve our goal. Specifically, we utilize road network connections and spatial neighborhoods to create *connectivity-aware graph* and *neighbor-aware graph*, such that the first form of correlation is captured, i.e., spatially close regions have similar embeddings. The *similarity-aware graph* considers similarities other than locations (e.g., the number of POIs and the number of roads) for each region, to capture the second form of correlation. Subsequently, we use the graph neural network model GAT (Graph Attention Networks) [26] to generate representations based on these three spatial graphs.

Future time interval embedding (to capture temporal periodicity): For each time interval t, we extract its "day-in-week" and "time-in-day", whose embeddings are used to represent t. To capture the temporal periodicity, we respectively build *connection graphs* for "day-in-week" and "time-in-day", and then leverage certain unsupervised graph embedding methods (e.g., node2vec [10]) to initialize each time interval's embedding.

Past traffic data encoding (to capture inter-traffic correlations): Specifically, we aim to capture i) the correlation between past and future traffic data, and ii) the inter-traffic correlations (between traffic flows and travel demands) for past traffic data. To capture the former, we apply sequence encoding models to encode past information, and propose some novel fusion frameworks to merge encoding results of past and future traffic data into a unified representation. To capture the latter, we further enhance this unified representation by exploiting inter-traffic similarities among regions. In particular, we first compute the inter-traffic similarity for any two regions based on past traffic data, and then use it to enhance the three spatial graphs we earlier created in capturing the region-level correlations.

Finally, we fuse the above representations and design an estimation model to predict future traffic data for each region. In summary, we make the following contributions:

- We design a comprehensive neural network model, `DeepTP`, that can fully exploit regions, time intervals, road networks and past traffic data to achieve a joint prediction of travel demands and traffic flows. To our best knowledge, this is the first work studying the joint prediction of traffic data (Section III).
- We construct three spatial graphs to capture two different forms of region-level correlations, based on which we propose effective embedding methods to generate hidden representations for regions (Section IV-A).
- We propose effective embedding methods to generate hidden representations for time intervals by utilizing the temporal periodicity. (Section IV-B).
- We design an effective encoding model to generate the representation for the past traffic data, which can fully exploit the region-level and inter-traffic correlations. (Section V).
- We conduct a comprehensive evaluation on two real world datasets. The results show that our method outperforms existing approaches significantly (Section VII).

II. Problem Formulation

A. Preliminaries

A road network is a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V} = \{v_i\}$ is a node set, $\mathcal{E} = \{e_{ij}\}$ is an edge set, and an edge e_{ij} represents the road segment from v_i to v_j. Similar to [34], [31], [21], we regard a city as a rectangle, and then split it into $H \times W$ grids, where each grid corresponds to a region $r_{i,j}$ ($i \in [1, H]$ and $j \in [1, W]$). Then the whole city can be denoted as the set $R = \{r_{i,j}\}$. In addition, $r_{i,j}$ indicates the $i \times W + j$-th region. Let $k = i \times W + j$, then we use r_k and $r_{i,j}$ interchangeably in this paper.

Connectivity-aware Graph. We build a *connectivity-aware graph* $A_c = \langle R, \mathcal{E}_c \rangle$ to model the connectivity among regions. The weight of an edge $e^c_{ij} \in \mathcal{E}_c$ indicates the number of edges linking r_i and r_j in the road network $\mathcal{G}$. Taking Figure 2 for example, there are $4 \times 4 = 16$ regions and the corresponding *connectivity-aware graph* includes 16 nodes. There are two road edges e_1 and e_2 coming from r_5 to r_1, so the weight of the region link e^c_{51} is set to 2. If a road edge in the road network passes through k ($k \geq 1$) regions, the road edge would correspond to $k-1$ region links. For example, e_1 passes through r_9, r_5 and r_1, so it corresponds to e^c_{95} and e^c_{51}. In this paper, for simplicity, A_c is used to represent both the graph and its adjacency matrix $A_c \in \mathbb{R}^{N \times N}$, where $A_c[i, j]$ means the weight of the edge e^c_{ij}.

Neighbor-aware Graph. To account for the spatial neighboring correlation between regions, we build a *neighbor-aware graph* $A_n = \langle R, \mathcal{E}_n \rangle$. In A_n, an edge $e^n_{ij} \in \mathcal{E}_n$ indicates that r_i and r_j are neighboring regions. As shown in Figure 2, each region has at most eight neighbors. Here, A_n also represents its adjacency matrix, where $A_n[i, j]$=1 if there exists an edge e^n_{ij} in $\mathcal{E}_n$; otherwise, it is 0.

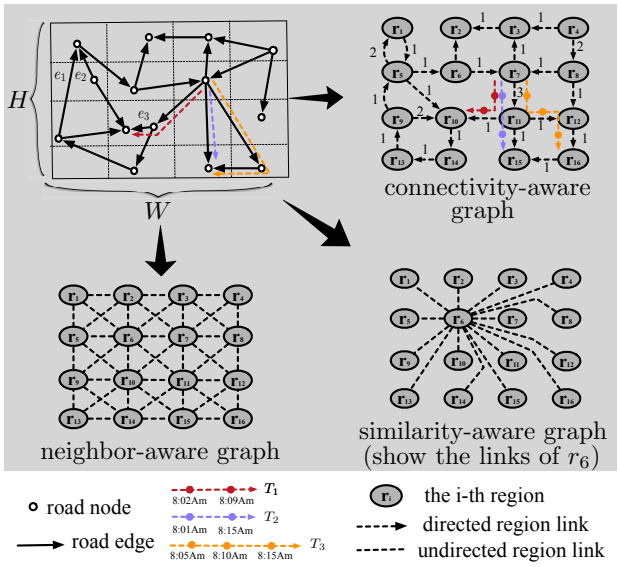


Fig. 2. Road Network & Regions

Similarity-aware Graph. Each region is affiliated with some properties (e.g., number of POIs, number of roads and the type of regions). Intuitively, regions with similar properties may have similar traffic tendency. For example, if a region belongs to a residence area, there should be many demands and outflows in morning peak hours, and many inflows in the evening peak hours. To capture such a similarity among regions, we build a fully connected undirected graph $A_s = \langle R, \mathcal{E}_s \rangle$, namely *similarity-aware graph*. As shown in Figure 2, we plot only the edges linking with the region r_6 for simplicity. Also, $A_s \in \mathbb{R}^{N \times N}$ can represent its adjacency matrix, where $A_s[i,j]$ is computed with $cos(p_{r_i}, p_{r_j})$. Here, $cos(\cdot,\cdot)$ is the cosine similarity function, and the vectors p_{r_i} and p_{r_j} denote the properties of r_i and r_j, respectively.

B. Problem Definition

For a region r and a time interval t (e.g., 10 minutes), we consider two types of traffic data: travel demand and traffic flow, denoted as d_r^t and f_r^t, respectively. d_r^t is the total number of potential travel demands (e.g., taxi orders) starting from r during t. f_r^t can be further split into two types: the traffic inflow if_r^t that represents the total number of crowds arriving at r during t, and the traffic outflow of_r^t that indicates the total number of crowds leaving from r during t.

Definition 1 (Travel Demands and Traffic Flows): Given a time interval t and a city $R = \{r_{i,j}\}$ ($i \in [1,H]$ and $j \in [1,W]$), the associated travel demands and traffic flows are denoted as $\mathbf{D}_R^t = [[d_r^t]_j]_i \in \mathbb{R}^{H \times W \times 1}$ and $\mathbf{F}_R^t = [[if_r^t, of_r^t]_j]_i \in \mathbb{R}^{H \times W \times 2}$. Specifically, we jointly represent the travel demands and traffic flows as $\mathbf{DF}_R^t = [[d_r^t, if_r^t, of_r^t]_j]_i \in \mathbb{R}^{H \times W \times 3}$. For simplicity, we use $\mathbf{D}^t$, $\mathbf{F}^t$ and $\mathbf{DF}^t$ to represent $\mathbf{D}_R^t$, $\mathbf{F}_R^t$ and $\mathbf{DF}_R^t$.

For example, we illustrate three trajectories (T_1, T_2 and T_3) during 8:00Am–8:20Am in Figure 2. When we set the size of a time interval as 10 minutes, we would collect travel demands and traffic flows at two time intervals (t_1 =[8:00Am–8:10Am] and t_2 =[8:10Am–8:20Am]), respectively. Then, we have $d_{r_7}^{t_1} = 3$. We also have $of_{r_7}^{t_1} = 3$ and $if_{r_{11}}^{t_1} = 3$, because the time (8:02Am, 8:01Am and 8:05Am) of leaving from r_7 to r_{11} is at t_1 for the three trajectories.

Last, we formally define our traffic prediction problem.

Definition 2 (Traffic Prediction Problem): Given a city R, a future time interval I, and past traffic data $\mathbf{DF}^{[I-l:I-1]} = \{\mathbf{DF}^{I-l}, \cdots, \mathbf{DF}^{I-1}\}$, where l is the number of past time intervals, the problem is to predict travel demands $\mathbf{D}^I$ and traffic flows $\mathbf{F}^I$. For simplicity, we represent $\mathbf{DF}^{[I-l:I-1]} = \{\mathbf{DF}^{I-l}, \cdots, \mathbf{DF}^{I-1}\}$ with $\mathbf{DF}^{[1:l]} = \{\mathbf{DF}^1, \cdots, \mathbf{DF}^l\}$.

III. Model Overview

In this section, we outline the architecture of our proposed model DeepTP and how it incorporates the desired properties reported in Example 1. As shown in Figure 3, DeepTP contains four modules: the future spatio-temporal information encoding module $\mathcal{M}_c$ (to be elaborated in Section IV), the past traffic sequence encoding module $\mathcal{M}_p$ (Section V-A), the graph-based correlation encoding module $\mathcal{M}_g$ (Section V-B), and the final estimation module $\mathcal{M}_e$ (Section VI-A).

In $\mathcal{M}_c$, we first design the region embedding and the time interval embedding models, which respectively account for the region-level correlations and temporal periodicity, to embed regions R and the future time interval I. In addition, we use a fully connected neural network (FC_c) to encode various context features (if available), such as event features (e.g., holiday) and meteorological features (e.g., weather). At last, we design the *sd-Fusion* module to fuse the above representations into the code h_{sd}.

When designing the past traffic sequence encoding module $\mathcal{M}_p$, we regard past traffic data ($\mathbf{DF}^{[1:l]}$) as a sequence and use a sequence encoding model GRU [4] to capture the temporal dependency of traffic. In particular, we successively take the traffic data ($\mathbf{DF}^1, \cdots, \mathbf{DF}^l$) as the input of GRU and generate hidden codes ($h_{df}^1, h_{df}^2, \cdots, h_{df}^l$) for past time intervals. In order to represent past and future features together, we design the *pc-Fusion* module to fuse these past hidden codes with the code h_{sd} to generate the fused code h_{pc}.

To capture the inter-traffic correlations among regions, we design the module $\mathcal{M}_g$. We compute inter-traffic correlations (denoted as H_{tr}) between any two regions to enhance the three spatial graphs, based on which we apply a graph neural network model GAT (Graph Attention Networks) [26] to further encode h_{pc} into three codes, $h_g^{\hat{c}}$, $h_g^{\hat{n}}$ and $h_g^{\hat{s}}$, respectively.

In $\mathcal{M}_e$, we first merge the four codes h_{pc}, $h_g^{\hat{c}}$, $h_g^{\hat{n}}$ and $h_g^{\hat{s}}$, and then use a fully connected neural network FC_e to generate the final traffic estimation $\hat{\mathbf{DF}}^I$. In addition, we design several loss functions to compute the loss between estimated and actual traffic, to facilitate our training of DeepTP.

IV. Spatial and Temporal Information Encoding

In Section IV-A, we propose a model to embed regions into hidden representations by capturing the region-level correlations. In Section IV-B, we describe the time interval embedding model. Last, we describe the context encoding network FC_c and the fusion module *sd-Fusion* in Section IV-C.

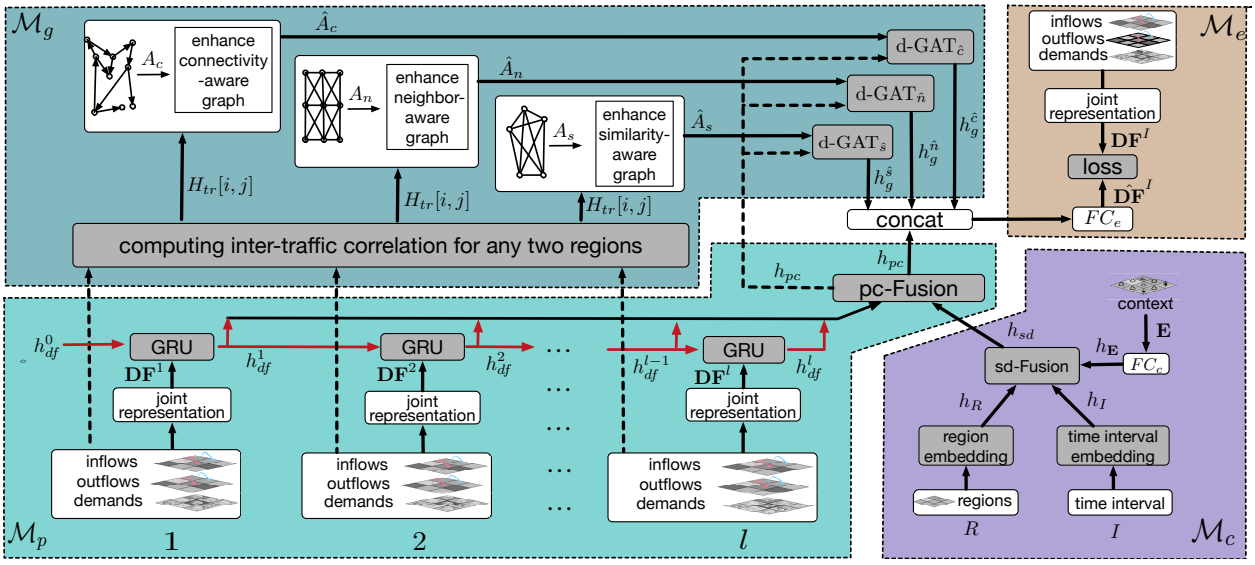


Fig. 3. The Model Framework of DeepTP

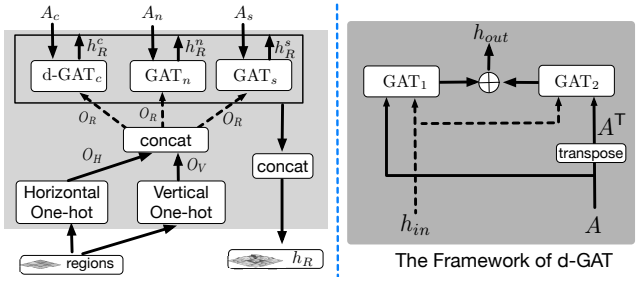


Fig. 4. The Framework of Region Embedding

A. Region Embedding

Recall Example 1 we leverage two region-level correlations to encode regions: (i) if two regions are neighbors or connected by road edges, there would be similar traffic for them; (ii) if two regions have similar/dissimilar spatial properties, there would be similar/dissimilar traffic for them. Among the three spatial graphs constructed in Section II-A, *connectivity-aware graph* A_r and *neighbor-aware graph* A_n are used to capture correlation-(i), and *similarity-aware graph* A_s is used to cpature correlation-(ii).

By taking into account these three spatial graphs, we design the region embedding model in Figure 4. Note that the input of any machine learning method is usually a vector, while a region $r_{i,j}$ is identified by a vertical id i and a horizontal id j. We hence use vertical one-hot encoding and horizontal one-hot encoding to represent i and j, denoted as the vectors $o_h \in \mathbb{R}^H$ and $o_v \in \mathbb{R}^W$, respectively. Then, we represent r by concatenating o_h and o_v into a vector $o_r \in \mathbb{R}^{H+W}$. When considering all regions, the corresponding one-hot vectors would be respectively merged into the one-hot matrices $O_H = [[o_h]_{i,j}] \in \mathbb{R}^{N\times H}$, $O_V = [[o_v]_{i,j}] \in \mathbb{R}^{N\times W}$ and $O_R = [[o_r]_{i,j}] \in \mathbb{R}^{N\times(H+W)}$. Here, $N = H \times W$ dictates the number of regions. Next, we leverage a graph neural network model GAT (Graph Attention Networks) [26] to transform O_R into dense tensors. In particular, we apply two graph attention layers in our GAT models. Thus, given the adjacent matrix $A_x \in \mathbb{R}^{N\times N}$ of a spatial graph (here, x represents c, n or s) and the input O_R, the corresponding output h_R^x can be computed as follows:

$$h_g^1[i] = \mathbf{W}_g^1 O_R[i], \qquad \forall 1 \le i \le N$$

$$\alpha_g^1[i,j] = \frac{exp(a_g^1 \cdot [h_g^1[i], h_g^1[j], A_x[i,j]])}{\sum_{r_k \in \mathcal{N}_x(r_i)} exp(a_g^1 \cdot [h_g^1[i], h_g^1[k], A_x[i,k]])}, \forall 1 \le i,j \le N$$

$$h_g^2[i] = \mathbf{W}_g^2 \sum_{r_j \in \mathcal{N}_x(r_i)} \alpha_g^1[i,j] h_g^1[j], \qquad \forall 1 \le i \le N$$

$$\alpha_g^2[i,j] = \frac{exp(a_g^2 \cdot [h_g^2[i], h_g^2[j], A_x[i,j]])}{\sum_{r_k \in \mathcal{N}_x(r_i)} exp(a_g^2 \cdot [h_g^2[i], h_g^2[k], A_x[i,k]])}, \forall 1 \le i,j \le N$$

$$h_R^x[i] = \sum_{r_j \in \mathcal{N}_x(r_i)} \alpha_g^2[i,j] h_g^2[j], \qquad \forall 1 \le i \le N$$

where $\mathbf{W}_g^1 \in \mathbb{R}^{d_g^1 \times (H+W)}$ and $a_g^1 \in \mathbb{R}^{2d_g^1+1}$ are parameters of the first graph attention layer, $\mathbf{W}_g^2 \in \mathbb{R}^{d_g^2 \times d_g^1}$ and $a_g^2 \in \mathbb{R}^{2d_g^2+1}$ are parameters of the second graph attention layer, $\mathcal{N}_x(r_i) = \{r_j | A_x[i,j] > 0\}$ represents the set of neighboring regions of r_i in the graph A_x, and $h_R^x[i] \in \mathbb{R}^{d_g^2}$ represents the final dense embedding of r_i. In particular, three spatial graphs (A_c, A_n and A_s) correspond to three dense embeddings (h_R^c, h_R^n and h_R^s). For simplicity, we denote the above process as the formula $h_R^x = \text{GAT}_x(O_R|A_x)$.

Notably, *connectivity-aware graph* (A_c) is a directed graph. Thus, if an edge $e_{ij} \in \mathcal{E}_c$ but $e_{ji} \notin \mathcal{E}_c$, the neighboring region r_j would be ignored when we compute attentions for the region r_i. To address this issue, we design a new model, denoted as d-GAT, to replace the raw GAT for *connectivity-aware graph*. As shown in the right part of Figure 4, we consider both the adjacency matrix A and its transpose matrix A^T. Accordingly, we generate two embeddings based on A and A^T and merge them with the element-wise plus operation. This process is formalized as $h_{out} = \text{d-GAT}(h_{in}|A) = \text{GAT}_1(h_{in}|A) \oplus \text{GAT}_2(h_{in}|A^\mathsf{T})$, where h_{in} and h_{out} correspond to input representations and output representations, respectively. Here, for *connectivity-aware graph*, we have $h_{in} = O_R$ and $h_{out} = h_R^c$, so we denote the formula as $h_R^c = \text{d-GAT}_r(O_R|A_c)$.

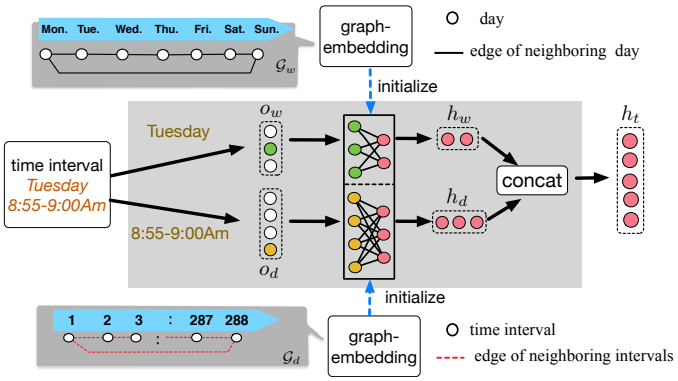


Fig. 5. The Framework of Time Interval Embedding ($\Delta t = 5min$)

With the above three dense embeddings generated, $h_R^c \in \mathbb{R}^{N \times d_g^2}$, $h_R^n \in \mathbb{R}^{N \times d_g^2}$ and $h_R^s \in \mathbb{R}^{N \times d_g^2}$, we concatenate them to get the final spatial representation $h_R = concat(h_R^c, h_R^n, h_R^s) \in \mathbb{R}^{N \times d_r}$, where $d_r = 3d_g^2$ denotes the dimension of each region's embedding.

B. Time Interval Embedding

In this section, we leverage our finding on the temporal periodicity to encode time intervals. 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). For example, given a time interval (Tuesday 8:55-9:00Am), its "day-in-week" and "time-in-day" are 2 and $\frac{9\times60}{\Delta t}$, respectively. Hence, embedding a time interval equals to embedding its "day-in-week" and "time-in-day". Next, we will explain the details.

As shown in Figure 5, given a time interval, we first extract its "day-in-week" and "time-in-day". Next, we use one-hot codes to represent them, denoted as $o_w \in \mathbb{R}^7$ and $o_d \in \mathbb{R}^{288}$ (if the size of a time interval is 5 minutes). Later, we use two fully connected neural networks to embed o_w and o_d into two dense vectors $h_w \in \mathbb{R}^{d_w}$ and $h_d \in \mathbb{R}^{d_d}$: $h_w = \mathbf{W}_w o_w$ and $h_d = \mathbf{W}_d o_d$, where $\mathbf{W}_w \in \mathbb{R}^{d_w \times 7}$ and $\mathbf{W}_d \in \mathbb{R}^{d_d \times 288}$ are corresponding parameters. In addition, it is intuitive that neighboring days or intervals have relevance on traffic. To capture this relation, we respectively construct *connection graphs* $\mathcal{G}_w$ and $\mathcal{G}_d$ for "day-in-week" and "time-in-day". Each *connection graph* is regarded as a circle where neighboring days/intervals are connected by an undirected edge. Hence, we can use some popular unsupervised graph embedding techniques (such as DeepWalk [22], Line [23], node2vec [10]) to embed day/interval nodes and use their embeddings to initialize $\mathbf{W}_w$ and $\mathbf{W}_d$. At last, we concatenate these two representations (h_w and h_d) to get the final temporal representation $h_t = concate(h_w, h_d) \in \mathbb{R}^{d_w + d_d}$. For simplicity, we denote the size of h_t as $d_t = d_w + d_d$.

To summarize, given a future time interval I, we can use the temporal embedding module to encode it into a hidden representation $h_I \in \mathbb{R}^{d_t}$.

C. Spatio-Temporal Representation Fusion

1) Context Encoding: As outlined in Section III we design a fully connected network FC_c to encode the context $\mathbf{E}$ as:

$$h_{\mathbf{E}_r} = \mathbf{W}_c^2 ReLU(\mathbf{W}_c^1 \mathbf{E}_r + b_c^1) + b_c^2 \quad (1)$$

where $\mathbf{E}_r \in \mathbb{R}^{d_c}$ represents the context of the region r, $\mathbf{W}_c^1 \in \mathbb{R}^{d_c^1 \times d_c}$ and $b_c^1 \in \mathbb{R}^{d_c^1}$ respectively denote the weight matrix and bias vector parameters of FC_c's first layer, and $\mathbf{W}_c^2 \in \mathbb{R}^{d_c^2 \times d_c^1}, b_c^2 \in \mathbb{R}^{d_c^2}$ indicate the corresponding parameters of FC_c's second layer. In addition, we select ReLU (Rectified Linear Unit) as the activation function, which is calculated as $ReLU(x) = max(0, x)$. The reason of using ReLU is that it can alleviate the vanishing gradient problem and thus get better gradient propagation, as reported in [9]. Thus, for the whole city R, we can get the corresponding representation $h_{\mathbf{E}} = [[h_{\mathbf{E}_r}]_j]_i \in \mathbb{R}^{H \times W \times d_2}$. Similar to the spatial representation h_R, we can also regard $h_{\mathbf{E}}$ as a matrix with the shape $\mathbb{R}^{N \times d_c^2}$, where $N = H \times W$ represents the number of all regions.

2) Fusing Spatio-temporal Representations: Based on the region embedding, the time interval embedding and the context encoding, we get three spatio-temporal representations h_R, h_I and $h_{\mathbf{E}}$, respectively. Intuitively, they can be categorized into two parts: *static* and *dynamic*. (1) Once the whole model is trained, h_R and h_I would be *static*. In other words, when using the trained model to predict traffic, both the same regions' embeddings and same time intervals' embeddings would be same. (2) $h_{\mathbf{E}}$ is *dynamic* because the context input $\mathbf{E}$ varies from time to time and does not pose the temporal periodicity. To fuse these representations, one possible way is to directly concatenate them, but that would ignore the relationship between *static* and *dynamic*.

To address this issue, we propose a novel fusion framework called *sd-Fusion*. The main idea is to use the *dynamic* representation to enhance the *static* representations. As shown in Figure 6, given a region r, we first consider its associated spatial embedding $h_r = h_R[r]$, context encoding $h_{\mathbf{E}_r}$ and temporal embedding h_t ($h_t = h_I$ is the same across different regions). Next, we use two fully connected neural networks (FC_{sd}^1 and FC_{sd}^2) to convert $h_{\mathbf{E}_r}$ into the domain of h_r and the domain of h_t, respectively. Later, we enhance h_r and h_t into h_r' and h_t' respectively, in a weighted summation manner.

$$h_{\mathbf{E}_r}^1 = \mathbf{W}_{sd}^1 h_{\mathbf{E}_r} + b_{sd}^1, \qquad h_{\mathbf{E}_r}^2 = \mathbf{W}_{sd}^2 h_{\mathbf{E}_r} + b_{sd}^2$$

$$\alpha_{sd}^1 = \frac{exp(h_{\mathbf{E}_r}^1)}{1 + exp(h_{\mathbf{E}_r}^1)}, \qquad \alpha_{sd}^2 = \frac{exp(h_{\mathbf{E}_r}^2)}{1 + exp(h_{\mathbf{E}_r}^2)}$$

$$h_r' = \alpha_{sd}^1 h_r + (1 - \alpha_{sd}^1) h_{\mathbf{E}_r}^1, \qquad h_t' = \alpha_{sd}^2 h_t + (1 - \alpha_{sd}^2) h_{\mathbf{E}_r}^2$$

Here, $\mathbf{W}_{sd}^1 \in \mathbb{R}^{d_r \times d_c^2}, b_{sd}^1 \in \mathbb{R}^{d_r}$ and $\mathbf{W}_{sd}^2 \in \mathbb{R}^{d_t \in \mathbb{R}^{d_t \times d_c^2}}, b_{sd}^2 \in \mathbb{R}^{d_t}$ respectively denote the parameters of FC_{sd}^1 and FC_{sd}^2; $\alpha_{sd}^1 \in \mathbb{R}^{d_r}, \alpha_{sd}^2 \in \mathbb{R}^{d_t}$ denote the associated weights; $h_r' \in \mathbb{R}^{d_r}, h_t' \in \mathbb{R}^{d_t}$ denote the enhanced results. At last, we concatenate the two enhanced results into the final fused spatio-temporal representation $h_{sd}[r] = concat(h_r', h_t') \in \mathbb{R}^{d_r + d_t}$ for the region r. As a result, for the whole city R, we have the spatio-temporal representation $h_{sd} \in \mathbb{R}^{N \times (d_r + d_t)}$, where N is the number of all regions.

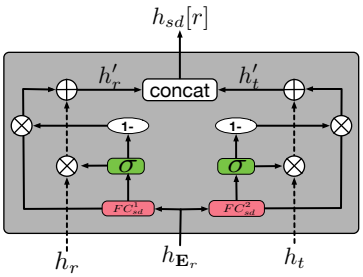


Fig. 6. The sd-Fusion Framework

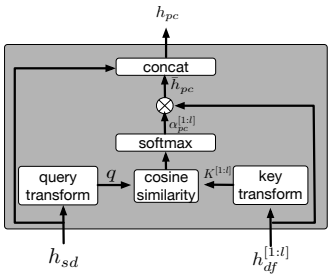


Fig. 7. The pc-Fusion Framework

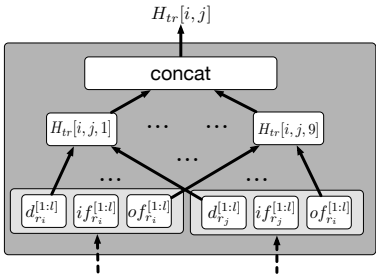


Fig. 8. Inter-traffic Correlation Computation

Also, we call h_{sd} as the future code as it implicates future information.

V. Past Information Encoding

Encoding past traffic data $\mathbf{DF}^{1:l}$ is achieved in three steps: (1) we leverage sequence encoding to encode it into basic codes; (2) to capture the relationship between past traffic and the spatio-temporal info at future time interval, we design a fusion module *pc-Fusion* to merge the basic code with the future code h_{sd} (Section V-A); (3) we incorporate inter-traffic correlations among regions into the fused result (Section V-B).

A. Past Traffic Encoding

As shown in the module $\mathcal{M}_p$ of Figure 3, we first progressively generate the basic code h_{df}^t for each past time interval t, and then merge them with the future code h_{sd} into the final hybrid representation h_{pc}.

Basic Sequence Encoding. In the local view, we jointly represent travel demands and traffic flows by concatenating them into the matrix $\mathbf{DF}^t \in \mathbb{R}^{N\times 3}$ at each previous step t, where $1 \le t \le l$. Later, in the global view, we consider $[\mathbf{DF}^1, \mathbf{DF}^2, \cdots, \mathbf{DF}^l]$ as a sequence in capturing the temporal dependency. Hence, it is intuitive to apply some sequence encoding models to generate the basic code, and we choose the RNN model GRU (Gated Recurrent Units) [4]. For each region r, given its input sequence $[\mathbf{DF}^1[r], \mathbf{DF}^2[r], \cdots, \mathbf{DF}^l[r]]$, the GRU model would recursively generate the hidden representation $h_{df}^t[r]$ at each t-th step by encoding the corresponding input $\mathbf{DF}^t[r]$ and the previous hidden representation $h_{df}^{t-1}[r]$ with the following formulae.

$$
\begin{aligned}
z^t &= \sigma(\mathbf{W}_z[h_{df}^{t-1}[r], \mathbf{DF}^t[r]] + b_z) \\
u^t &= \sigma(\mathbf{W}_u[h_{df}^{t-1}[r], \mathbf{DF}^t[r]] + b_u) \\
\tilde{h}_{df}^t[r] &= tanh(\mathbf{W}_h[z^t \otimes h_{df}^{t-1}[r], \mathbf{DF}^t[r]] + b_h) \\
h_{df}^t[r] &= (1 - u^t[r]) \otimes h_{df}^{t-1}[r] + u^t \otimes \tilde{h}_{df}^t[r]
\end{aligned}
$$

The initial value of $h_{df}^t[r]$ is $h_{df}^0[r] = \mathbf{0}$. For each time step t, $\mathbf{DF}^t[r] \in \mathbb{R}^3$ means the input vector, $z^t, u^t \in \mathbb{R}^{d_h}$ represent the activation vector of reset gate and update gate, respectively; $\tilde{h}_{df}^t[r]$ denotes the current state vector; $h_{df}^t[r] \in \mathbb{R}^{d_h}$ is the updated state vector, also known as the output vector. In addition, $\sigma(\cdot)$ and $tanh(\cdot)$ represent two kinds of activation functions, i.e., sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ and hyperbolic tangent function $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Notably, we need to learn the weight matrix parameters ($\mathbf{W_z}, \mathbf{W_u}, \mathbf{W_h} \in \mathbb{R}^{d_h \times (3+d_h)}$) and the bias vector parameters ($b_z, b_u, b_h \in \mathbb{R}^{d_h}$) in the training stage. As a result, we can get the output vector $h_{df}^t[r] \in \mathbb{R}^{d_h}$ for each region r at the step t. Besides, we merge the output vectors of all regions and get the basic code $h_{df}^t \in \mathbb{R}^{N\times d_h}$. For simplicity, this iterative process is denoted as $h_{fd}^t = GRU(\mathbf{DF}^t, h_{df}^{t-1})$.

Past-Future Fusion. Although we have progressively generated basic codes $h_{df}^{[1:l]} = \{h_{df}^1, \cdots, h_{df}^l\}$ for past features, the relation between past and future features has not been incorporated. Hence, we need to fuse $h_{df}^{[1:l]}$ with the future code h_{sd}. One way is to directly concatenate them, but features at different past time intervals should have different influence on future prediction. To achieve so, we leverage the attention mechanism, which has been widely applied in other domains [20], [25]. This mechanism includes three objects: *query*, *key* and *value*. *key* and *value* are two sets and each element in *key* has a corresponding element in *value*.

Given a *query*, it would compute the attention between the *query* and each element in *key*, and then regard the attention as the weight of the corresponding element in *value*. After that, it would take the weighted sum of all elements in *value* as the final result. As a result, the attention can capture the different influence of different elements in *value* for the *query*.

As shown in Figure 7, we regard the future code h_{sd} as the *query*, and regard basic codes $h_{df}^{[1:l]}$ as both the *key* and *value*. First, we transform h_{sd} and $h_{df}^{[1:l]}$ into the same space, denoted as q and $K^{[1:l]}$ respectively. Next, we use the *cosine* function to compute the similarity between q and each element K^t as the corresponding attention. Then, we use the *softmax* function to normalize attentions into the weights $\alpha_{pc}^{[1:l]}$, where $\sum_{1\le t\le l} \alpha_{pc}^t = 1$. Thus, we can compute the weighted sum $\bar{h}_{pc}$ based on the weights $\alpha_{pc}^{[1:l]}$ and basic code $h_{df}^{[1:l]}$. Last, we concatenate the sum $\bar{h}_{pc}$ and the future code h_{sd} as the final hybrid past-future code h_{pc}. In summary, for each region r, the whole process can be formulated as below:

$$
\begin{aligned}
&q[r] = \mathbf{W}_q h_{sd}, \quad K^t[r] = \mathbf{W}_k h_{df}^t[r], \quad \forall 1 \le t \le l \\
&\alpha_{pc}^t[r] = softmax(cos(q[r], K^t[r])), \quad \forall 1 \le t \le l \\
&\bar{h}_{pc}[r] = \sum_{1\le t\le l} \alpha_{pc}^t[r] h_{df}^t[r], \quad h_{pc}[r] = concat(\bar{h}_{pc}[r], h_{sd})
\end{aligned}
$$

Here, $\mathbf{W}_q \in \mathbb{R}^{d_a \times (d_r + d_t)}$ and $\mathbf{W}_k \in \mathbb{R}^{d_a \times d_h}$ respectively represent the transformation matrices of the *query* h_{sd} and *key* $h_{df}^{[1:l]}$; $h_{pc}[r] \in \mathbb{R}^{d_h + d_r + d_t}$ is the hybrid past-future code for each region r. Similarity, we use the code $h_{pc} \in$

$\mathbb{R}^{N\times(d_h+d_r+d_t)}$ to represent features for the whole city, where N is the number of all regions.

B. Inter-traffic Correlation Encoding

To capture the correlations between traffic flows and travel demands, we design the module $\mathcal{M}_g$ to encode the inter-traffic correlations between any two regions, which consists of three steps (Figure 3): (i) compute the inter-traffic correlation for any two regions; (ii) leverage the correlation to enhance the three spatial graphs; (iii) use the three enhanced graphs to generate new representations via graph neural networks.

Computing inter-traffic correlation. Figure 8 shows how to compute the inter-traffic correlations between two regions r_i and r_j, by utilizing their associated past traffic data. We regard the traffic data of a region r_i as three vectors, $d_{r_i}^{[1:l]} \in \mathbb{R}^l$, $if_{r_i}^{[1:l]} \in \mathbb{R}^l$ and $of_{r_i}^{[1:l]} \in \mathbb{R}^l$, which respectively denote the travel demands, traffic inflows and traffic outflow. Similarly, we get $d_{r_j}^{[1:l]}$, $if_{r_j}^{[1:l]}$ and $of_{r_j}^{[1:l]}$ for r_j. Next, we join the three vectors of r_i with that of r_j to get inter-traffic pairs, and compute the inter-traffic correlation for each pair. For example, the first pair is $(d_{r_i}^{[1:l]}, d_{r_j}^{[1:l]})$, so we compute the correlation $H_{tr}[i,j,1]$ as $H_{tr}[i,j,1] = concat(d_{r_i}^{[1:l]}, d_{r_j}^{[1:l]}) \cdot a_{tr}[1]$, where $a_{tr} \in \mathbb{R}^{9\times 2l}$ is a parameter matrix learned by training data. Similarly, we can get $H_{tr}[i,j,2], \cdots, H_{tr}[i,j,9]$. At last, we concatenate the nine correlations into $H_{tr}[i,j] \in \mathbb{R}^9$. As for all region pairs, we would have $H_{tr} \in \mathbb{R}^{N\times N\times 9}$.

Enhancing spatial graphs. The above computation of inter-traffic correlation tends to ignore the spatial correlations between regions. Since spatial correlations are represented by three spatial graphs (a.k.a, *connectivity-aware graph* A_c, *neighbor-aware graph* A_n, and *similarity-aware graph* A_s), we use the following formula to fuse inter-traffic and region-level correlations, in order to generate the enhanced spatial graphs $\hat{A}_c, \hat{A}_n$, and $\hat{A}_s$. Here, $\hat{A}_x \in \mathbb{R}^{N\times N\times 9}$ is the adjacency matrix (tensor) of the enhanced spatial graph.

$$\hat{A}_x[i,j] = \begin{cases} H_{tr}[i,j], \ if \ A_x[i,j] > 0 \\ \mathbf{0}, \ \text{otherwise} \end{cases}$$

Further encoding via enhanced spatial graphs. In Section V-A, we have generated the code h_{pc} to represent the encoding of past and future features. To exploit the inter-traffic correlation, we further encode h_{pc} via the three enhanced spatial graphs. Note that $H_{tr}[i,j] \neq H_{tr}[j,i]$ as the influence is not symmetric. Thus, all enhanced spatial graphs belong to directed graphs, and we choose to use d-GAT to encode them. Similar to the one in Section IV-A, d-GAT generates new encodings ($h_g^{\hat{c}}, h_g^{\hat{n}}, h_g^{\hat{s}} \in \mathbb{R}^{N\times d_g^2}$) via the following formulae: $h_g^{\hat{c}} = \text{d-GAT}_{\hat{c}}(h_{pc}|\hat{A}_c)$, $h_g^{\hat{n}} = \text{d-GAT}_{\hat{n}}(h_{pc}|\hat{A}_n)$ and $h_g^{\hat{s}} = \text{d-GAT}_{\hat{s}}(h_{pc}|\hat{A}_s)$.

VI. Model Learning

In this section, we first focus on the last module $\mathcal{M}_e$ of DeepTP in Section VI-A, and then discuss how to train DeepTP and use the trained model to predict traffic in Section VI-B.

A. Final Estimation Module

We have encoded all features (a.k.a., future and past information) and generated the associated representations h_{pc}, $h_g^{\hat{c}}$, $h_g^{\hat{n}}$ and $h_g^{\bar{s}}$: h_{pc} captures temporal periodicity and basic spatial correlations while $h_g^{\hat{c}}$, $h_g^{\hat{n}}$ and $h_g^{\bar{s}}$ capture inter-traffic correlations among regions. To leverage them to predict future traffic, we first concatenate them into a unified representation and then apply the model FC_e to generate the estimation (Figure 3). Since FC_e is a two-layer fully connected neural network, the associated process is formulated as:

$$\hat{\mathbf{DF}}^I = \mathbf{W}_e^2 ReLU(\mathbf{W}_e^1 concat(h_{pc}, h_g^{\hat{c}}, h_g^{\hat{n}}, h_g^{\bar{s}}) + b_e^1) + b_e^2$$

where $\mathbf{W}_e^1 \in \mathbb{R}^{d_e\times(d_h+d_r+d_t+3d_g^2)}$ and $b_e^1 \in \mathbb{R}^{d_e}$ denote the weight matrix and bias vector parameters of FC_e's first layer, $\mathbf{W}_e^2 \in \mathbb{R}^{3\times d_e}$ and $b_e^2 \in \mathbb{R}^3$ indicate the affiliated parameters of FC_c's second layer, and $\hat{\mathbf{DF}}^I$ is the estimated result.

B. Model Training and Prediction

Offline training. Algorithm 1 outlines the training process. Given a road network $\mathcal{G}$ and all regions R of a city, we build three spatial graphs *connectivity-aware graph*, *neighbor-aware graph* and *similarity-aware graph*. Next, given the time interval size Δt, we build the associated *connection graphs* $\mathcal{G}_w$ for "day-in-week" and and $\mathcal{G}_d$ for "time-in-day". Then, we initialize the temporal embedding matrices $\mathbf{W}_w, \mathbf{W}_d$ by applying the graph embedding method $node2vec$ [10] with the *connection graphs* $\mathcal{G}_w$ and $\mathcal{G}_d$. As for other parameters of DeepTP, we use normal distribution to initialize them (lines 1-5). After that, we iteratively train the whole model with the given epochs S (lines 6-7). Specifically, *ModelTrain* explains the training process for each epoch. We first compute the training iterations S' based on a given batch size b_s, and then shuffle all training data X, Y (line 1). In each iteration, we extract b_s training data from X, Y. Each element of X is composed of $\mathbf{DF}^{1:l}$, the future time interval I, and the context features $\mathbf{E}$ (lines 2-5). In particular, we first use the module $\mathcal{M}_c$ to generate the code h_{sd} to represent future features, and then use the module $\mathcal{M}_p$ and the code h_{sd} to further encode past features into the past-future hybrid code h_{pc}. Next, we use the module $\mathcal{M}_g$ to generate hidden representations $h_g^{\hat{c}}, h_g^{\hat{n}}, h_g^{\hat{s}}$ to account for inter-traffic correlations. Afterwards, we concatenate the code h_{pc} with $h_g^{\hat{r}}, h_g^{\hat{n}}, h_g^{\hat{s}}$, and use the module $\mathcal{M}_e$ to make the estimation $\hat{\mathbf{DF}}^I$ (lines 6-9). Last, we adopt the idea from [31] to design the following loss function.

$$loss(Y, \hat{Y}) = \sum_{i=1}^{|Y|}((Y[i] - \hat{Y}[i])^2 + \gamma(\frac{Y[i] - \hat{Y}[i]}{Y[i]})^2)$$

It consists of two parts: the mean square loss and the square of mean absolute percentage loss. The former is more relevant to predictions of large values while the latter can prevent training from being dominated by large value samples. We utilize Adam Optimizer [14] to optimize all parameters by minimizing the total loss $loss$ (lines 10-11).

Algorithm 1: Model Learning for DeepTP

Input: the road network $\mathcal{G}$ and the regions R of a city, the time interval size Δt, training inputs X, training labels Y, test inputs X', different modules $(\mathcal{M}_c, \mathcal{M}_p, \mathcal{M}_g, \mathcal{M}_e)$, learning rate lr, training epochs S, batch size bs, loss weight γ.
Output: parameters $\theta_c, \theta_p, \theta_g, \theta_e$ for the four models $\mathcal{M}_c$, $\mathcal{M}_p$, $\mathcal{M}_g$ and $\mathcal{M}_e$, prediction results Y'

build *connectivity-aware graph* A_c, *neighbor-aware graph* A_n and *similarity-aware graph* A_s with $\mathcal{G}$ and R;
build *connection graphs* $\mathcal{G}_w$ and $\mathcal{G}_d$ for "day-in-week" and "time-in-day" with Δt;
$\mathbf{W}_w^0 \leftarrow node2vec(\mathcal{G}_w)$, $\mathbf{W}_d^0 \leftarrow node2vec(\mathcal{G}_d)$;
initialize embedding matrices $\mathbf{W}_w, \mathbf{W}_d$ with $\mathbf{W}_w^0, \mathbf{W}_d^0$;
initialize other parameters in $\theta_c, \theta_p, \theta_g, \theta_e$ with normal distribution;
for $i \leftarrow 1 \cdots S$ **do**
    $\theta_c, \theta_p, \theta_g, \theta_e \leftarrow ModelTrain(X, Y, \mathcal{M}_c, \mathcal{M}_p, \mathcal{M}_g, \mathcal{M}_e, lr, bs, \gamma, A_c, A_n, A_s)$; using $\theta_c, \theta_p, \theta_g, \theta_e$ to update $\mathcal{M}_c$, $\mathcal{M}_p$, $\mathcal{M}_g$ and $\mathcal{M}_e$;
$Y' \leftarrow Prediction(X', \mathcal{M}_c, \mathcal{M}_p, \mathcal{M}_g, \mathcal{M}_e, A_c, A_n, A_s)$;
return $\theta_c, \theta_p, \theta_g, \theta_e, Y'$;

Function ModelTrain-offline

Input: $X, Y, \mathcal{M}_c, \mathcal{M}_p, \mathcal{M}_g, \mathcal{M}_e, lr, bs, \gamma, A_c, A_n, A_s)$
training iterations $S' = \lfloor \frac{|X|}{bs} \rfloor$, $shuffle(X, Y)$;
for $i \leftarrow 1 \cdots S'$ **do**
    collect $X_{(i-1)b_s+1:i\times b_s}, Y_{(i-1)b_s+1:i\times b_s}$;
    $[(\mathbf{DF}^{1:l}, I, \mathbf{E})] \leftarrow X_{(i-1)b_s+1:i\times b_s}$;
    $[\mathbf{DF}^I] \leftarrow Y_{(i-1)b_s+1:i\times b_s}$;
    $[h_{sd}] \leftarrow \mathcal{M}_c([(R, I, \mathbf{E})], A_c, A_n, A_s)$;
    $[h_{pc}] \leftarrow \mathcal{M}_p([(\mathbf{DF}^{1:l}, h_{sd})])$;
    $[(h_g^{\hat{c}}, h_g^{\hat{n}}, h_g^{\hat{s}})] \leftarrow \mathcal{M}_g([(\mathbf{DF}^{1:l}, h_{pc}), A_c, A_n, A_s])$;
    $[\hat{\mathbf{DF}}^I] \leftarrow \mathcal{M}_e([(h_{pc}, h_g^{\hat{c}}, h_g^{\hat{n}}, h_g^{\hat{s}})]))$;
    $loss = \sum((\mathbf{DF}^I - \hat{\mathbf{DF}}^I)^2 + \gamma(\frac{\mathbf{DF}^I - \hat{\mathbf{DF}}^\mathbf{I}}{\mathbf{DF}^I})^2)$;
    $\theta_c, \theta_p, \theta_g, \theta_e \leftarrow AdamOpt(loss, lr)$;
return $\theta_c, \theta_p, \theta_g, \theta_e$;

Function Prediction-online

Input: $X', \mathcal{M}_c, \mathcal{M}_p, \mathcal{M}_g, \mathcal{M}_e, A_c, A_n, A_s$
Output: Y'
$[(\mathbf{DF}^{1:l}, I, \mathbf{E})] \leftarrow X'$;
$[h_{sd}] \leftarrow \mathcal{M}_c([(R, I, \mathbf{E})], A_c, A_n, A_s)$;
$[h_{pc}] \leftarrow \mathcal{M}_p([(\mathbf{DF}^{1:l}, \mathbf{E}^{1:l}, h_{sd})])$;
$[(h_g^{\hat{c}}, h_g^{\hat{n}}, h_g^{\hat{s}})] \leftarrow \mathcal{M}_g([(\mathbf{DF}^{1:l}, h_{pc}), A_c, A_n, A_s])$;
$[\hat{\mathbf{DF}}^I] \leftarrow \mathcal{M}_e([(h_{pc}, h_g^{\hat{c}}, h_g^{\hat{n}}, h_g^{\hat{s}})]))$;
return $[\hat{\mathbf{DF}}^I]$;

Online prediction. In the *Prediction* function, each element of the input X' is composed of $\mathbf{DF}^{1:l}$, I and $\mathbf{E}$, so we can use the trained model to generate $\hat{\mathbf{DF}}^I$ as the estimation Y'.

VII. Experiments

A. Experimental Setup

Datasets.

(1) **Road Networks & Regions & Time Intervals.** We used two road networks: *Chengdu Road Network (CRN)* and *Xi'an Road Network (XRN)* extracted from OpenStreetMap [1]. *CRN* includes $3,191$ vertices and $9,468$ edges; *XRN* contains $4,576$ vertices and $12,668$ edges. We split the area of each road network into disjoint grids with the size of $\Delta s \times \Delta s$. In particular, we respectively set Δs as 500m, 1km, 2km and 4km to evaluate the robustness of models, and 1km is the default setting. Hence, for a road network, the number of regions is computed with $N = H \times W = \lfloor dist_{lat}^{max}/\Delta s \rfloor \times \lfloor dist_{lng}^{max}/\Delta s \rfloor$, where $dist_{lat}^{max}$ and $dist_{lng}^{max}$ denote the longest latitude and longitude distance in the area covered by the road network. Also, we set the size of a time interval (Δt) as $5min$, $10min$, $30min$ and $60min$, where the default is $10min$ and min means minutes.

(2) **Region-based Traffic Flows and Travel Demands.** Similar to [21] and [31], we generate region-based traffic flows and travel demands based on taxi orders in *Chengdu (CD)* and *Xi'an (XA)*, collected from Didi Chuxing[2]. Each order includes an OD input (origin, destination and departure time) and its associated trajectory. There are $5.8M$ and $3.4M$ orders in *CD* and *XA* [32] respectively, both from 10/01/2016 to 11/30/2016. The total number of time intervals can be calculated via $\frac{61\times24\times60min}{\Delta t}$. We regard a region r's travel demand at a time interval t as the number of taxi orders whose origin is in r and departure time falls in t. As for each region's traffic flows, we use the way described in Section II-B to generate the inflows and outflows based on the associated trajectories. We set l, the number of past time intervals (in Definition 2), as 6, 12, 24, 48 for robustness evaluation, and the default is 12.

(3) **External Data for Context Features.** Two types of external data, namely event features and meteorological features, are used. For event features, we consider two indicators: the holiday indicator (1 for holiday, 0 otherwise); the season indicator (using 1-4 to denote four seasons). For meteorological features, we collect various weather properties[3] for each region, and use one-hot codes to represent categorical properties.

(4) **Training, Validation and Test Data.** Since we can get $\frac{24\times61\times60min}{\Delta t}$ time intervals for each dataset, we divide a dataset into training, validation and test data by splitting the time intervals with the ratios of 70%:10%:20%.

Baselines. We compare with seven typical baselines:

- Historical Average (HA): it uses the average value of the given past traffic data as the predicted result.
- Autoregressive Integrated Moving Average (ARIMA): a widely used model for time series prediction, which combines moving average and autoregression [6]. We train an individual ARIMA model for each region.
- Gradient Boosting Regression Tree (GBRT): a gradient boosting decision tree based regression method that is implemented using XGBoost [3].
- Deep Sequence Model (Seq [17]): using deep sequence model (i.e., LSTM) to predict the future traffic.
- Deep Multi-View Spatio-temporal Network (DMVST [31]): it uses the CNN model to encode sptio-temporal features

[1] https://www.openstreetmap.org

[2] https://outreach.didichuxing.com/research/opendata/

[3] https://darksky.net/dev/docs

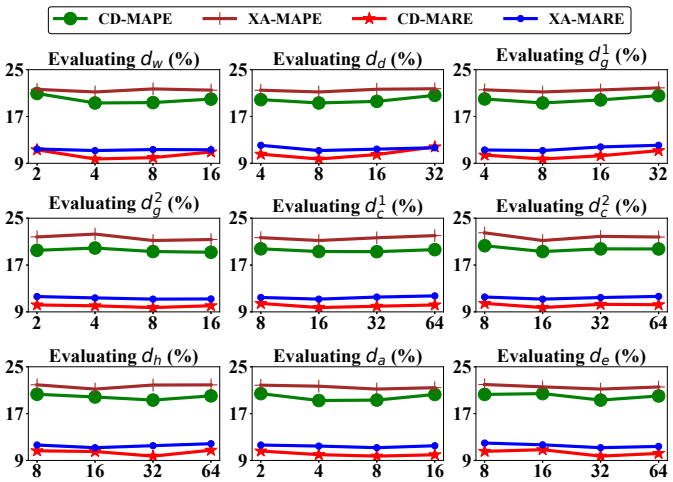


Fig. 9. MAPE & MARE vs. Hyper-parameters

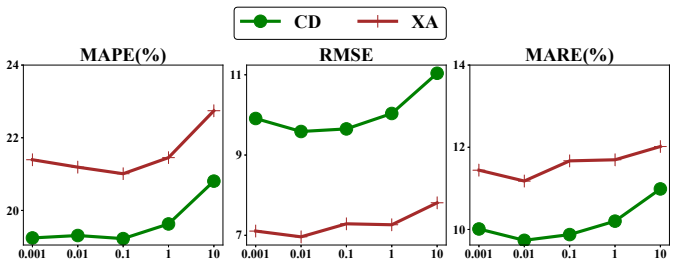


Fig. 10. Loss on Validation Data vs. the Loss Weight λ

TABLE I
EFFECTIVENESS RESULTS ON TEST DATA(Δs=1km, Δt=10min, l=12)

Methods		HA	ARIMA	GBRT	Seq	DMVST	MGCN	ST-Meta	NF	NP	NC	DeepTP
CD	RMSE	20.91	39.13	16.42	17.45	15.09	13.56	12.37	10.91	13.28	10.41	**9.59**
	MAPE(%)	41.78	69.56	52.06	28.36	25.73	22.62	22.17	21.28	22.07	21.05	**19.31**
	MARE(%)	18.68	39.05	16.95	16.53	14.49	12.89	12.03	11.08	13.09	14.04	**9.74**
XA	RMSE	13.61	28.15	10.52	9.14	8.30	9.81	9.29	8.14	7.79	7.94	**6.96**
	MAPE(%)	46.59	82.26	53.41	26.28	24.79	23.20	22.50	25.01	23.62	25.85	**21.19**
	MARE(%)	21.90	46.31	17.74	15.83	14.53	12.78	12.28	14.39	13.66	14.93	**11.18**

in the local view and then uses LSTM to generate predicted results in the global view.

- Multi-Graph Convolution Network (MGCN [8]): it encodes regions' correlations with multiple graphs, then employs the GCN model to encode each region, and uses the RNN model to predict the future traffic.
- Deep Meta Learning (ST-Meta [21]): it uses the meta learning model (i.e., the meta GAT and meta RNN models) to encode spatio-temporal features.

Environment. All machine/deep learning methods are 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 use Adam [14] as the optimization method with the mini-batch size of 12, the learning rate is set as 0.001, and the training epochs are set as 50.

Evaluation Metrics. RMSE (Root Mean Square Error), MAPE (Mean Absolute Percent Error) and MARE (Mean Absolute Relative Error) are used. Let $\mathbf{y} = \{y^i\}$ be the ground truth and $\hat{\mathbf{y}} = \{\hat{y}^i\}$ be the predicted result. Then we have: $RMSE(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{i=1}^{N} \sqrt{\frac{|y^i - \hat{y}^i|^2}{N}}$, $MAPE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N} |\frac{y^i - \hat{y}^i}{y^i}|$, $MARE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_{i=1}^{N} |y^i - \hat{y}^i|}{\sum_{i=1}^{N} |y^i|}$.

B. Setting of Model's Hyper-parameters

We consider four hyper-parameters: (1) the embedding sizes (d_w, d_d) of time intervals; (2) the sizes $(d_g^1, d_g^2, d_c^1, d_c^2)$ of different layer's neural networks in $\mathcal{M}_c$; (3) the sizes (d_h, d_a) of different layer's neural networks in $\mathcal{M}_p$; (4) the sizes (d_e) of different layer's neural networks in $\mathcal{M}_e$. In particular, we set each hyper-parameter's values as different integers, and then evaluate DeepTP's performance on the validation of *CD* and *XA*. As shown in Figure 9, we plot the MAPE and the MARE 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 $d_w = 4$, $d_d = 8$, $d_g^1 = 8$, $d_g^2 = 16$, $d_c^1 = 32$, $d_c^2 = 16$, $d_h = 32$, $d_a = 4$, $d_e = 32$. (2) For *XA*, we have $d_w = 4$, $d_d = 8$, $d_g^1 = 8$, $d_g^2 = 8$, $d_c^1 = 16$, $d_c^2 = 16$, $d_h = 16$, $d_a = 8$, $d_e = 32$.

C. Effectiveness of Loss Weight

To fine-tune the loss weight λ, we vary it from 0.001 to 10 with a scale of 10 in training DeepTP. We compute the three metrics for the validation data. The result is plotted in Figure 10, from which we find: the performance first improves with the increase of λ, but is much worsened when exceeding a certain threshold. The reason is two-fold: (i) there is a trade-off between the accuracy of large λ and small λ; (ii) a bigger λ leads to a bigger loss, making it difficult to converge to a global optimum based on the given learning rate, so the performance is extremely worsened when λ is greater than 1. The best value of λ is 0.1, 0.01 and 0.01 on these three metrics respectively, and it holds for both *CD* and *XA*. Based on the majority voting rule, we set the default value of λ as 0.01.

D. Effectiveness Comparison

Apart from comparing DeepTP with the seven baseline methods, we replace our DeepTP by three variations, namely NF, NP and NC, to evaluate the effectiveness of different parts of encodings in DeepTP (see Figure 3). In NF, we remove the future spatial-temporal information encoding module. In NP, we remove the past traffic sequence encoding module. In NC, we remove the graph-based correlation encoding module.

Table I reports the evaluation results of all methods with the default settings of Δs, Δt and l, and we observe:

(1) ARIMA performs the worst and is not good enough to model traffic, due to its limited model expressiveness and incapability to fully leverage complex spatial-temporal features.

(2) Neural network based methods outperform other methods, because they can approximately fit any function.

(3) The MARE is consistently better than the MAPE across all methods. By definition of MAPE and MARE, if MAPE is greater than MARE, then $\sum_{i=1}^{N} |y^i - \hat{y}^i|(\frac{1}{y^i} - \frac{N}{\sum_{j=1}^{N} y^j}) > 0$, i.e., $|y^i - \hat{y}^i|$ is larger when $y^i < \sum_{j=1}^{N} y^j/N$. That means the difference between the ground truth and the prediction is larger when the ground truth traffic is small, and hence all methods suffer when the traffic data are too small.

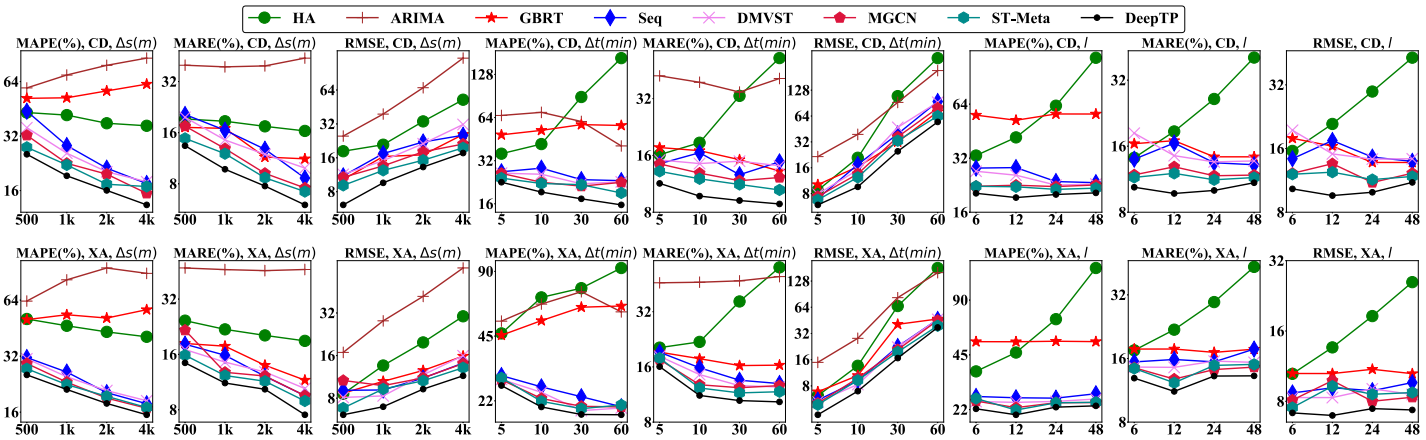


Fig. 11. MAPE & RMSE & MARE vs. Variables Δs & Δt & l. (The title of each subfigure is labelled with the format "A, B, C", where "A", "B" and "C" respectively refer to a metric, a dataset and one kind of variable.)

(4) When looking into the results of NF, NP, NC and DeepTP, we find that the graph-based correlation encoding is the most critical part of DeepTP, followed by future spatial-temporal information encoding and past traffic encoding. In other words, considering the correlation between different kinds of traffic data is the main reason that DeepTP outperforms.
(5) DeepTP performs the best on all metrics. For example, DeepTP outperforms the best existing method ST-Meta by more than 19% on MARE for the test data of *CD*.
(6) When comparing the relative error (a.k.a, MAPE and MARE), the performance of almost all methods on *CD* is better than that on *XA*. The reason is that the traffic data of *XA* is sparser than *CD*, and intuitively it is more difficult to learn good representations from sparse data.

Furthermore, we evaluate the robustness of different models by respectively varying the values of Δs, Δt and l in Figure 11. We have the following observations:
(1) With the increase of Δs or Δt, the absolute error RMSE increases for all methods, because the affiliated traffic data at each time interval for each region is larger.
(2) As for MAPE and MARE, different models show different changes. First, both MAPE and MARE decrease for all deep learning based methods. Larger Δs or Δt corresponds to denser traffic data, making it better to learn spatial-temporal representations. Second, with the same reason, MARE decreases for GBRT; however, its MAPE increases as GBRT cannot take effect on the prediction of small traffic data when training data contain more large traffic. Third, the performance of HA improves when Δs increases, but deteriorates when Δt increases. Actually, a bigger Δt means that longer past traffic data needs to be incorporated to compute the average value for HA, but they have less influence to the future traffic. Last, ARIMA heavily relies on the stationarity of the traffic data sequence, which unfortunately cannot be guaranteed by increasing or deceasing Δs and Δt. Thus, both MAPE and MARE fluctuate.
(3) With the increase of the size l of past sequence, all three metrics show the same trend for every method. The reason is similar to the increase of Δt.

TABLE II
EFFICIENCY OF TEST RESULT (Δs=1km, Δt=10min, l=12)

	dataset	HA	ARIMA	GBRT	Seq	DMVST	MGCN	ST-Meta	DeepTP
model size(MByte)	*CD*	-	0.78	1.6	0.31	0.47	0.29	0.11	0.56
	XA	-	0.78	1.6	0.31	0.47	0.24	0.11	0.48
training time (minutes/epoch)	*CD*	-	23.59	6.47	11.45	13.89	14.15	10.13	9.05
	XA	-	21.97	5.82	11.16	13.30	13.47	9.26	7.94
estimation time(seconds)	*CD*	2.65	7.33	6.77	24.44	24.79	170.79	32.05	24.50
	XA	2.70	6.99	6.58	20.65	20.21	130.02	27.39	20.10

(4) No matter how Δs, Δt and l change, our method DeepTP consistently has the best performance.

E. Efficiency Comparison

We use the model size, training time and estimation time for efficiency evaluation. The model size represents the size of required memory for applying the corresponding model, and is used to evaluate the efficiency of memory usage. The training time is used to evaluate the offline learning efficiency. In particular, we compute the average time of an epoch for each method. The estimation time can evaluate the online prediction efficiency. The results are reported in Table II (HA is not a learning method, so we ignore its model size and training time). We observe the following:
(1) Compared with state-of-the-art MGCN and ST-Meta, DeepTP is more efficient in both offline training and online prediction.
(2) GBRT requires the most memory while ST-Meta consumes the least memory.
(3) The model size is almost the same for most methods, except for MGCN and DeepTP. The reason is that both MGCN and DeepTP have to maintain region-based graphs, whose sizes vary from one dataset to another. In addition, our model DeepTP includes region embeddings, whose size also varies for different cities. In particular, the number of regions in *CD* is greater than *XA*, so the model size on *CD* is greater than that on *XA*.
(4) ARIMA needs more training time than other methods. That is because ARIMA has to consider the whole training data as a sequence and then input it at once for training parameters, while other methods are based on batch training.

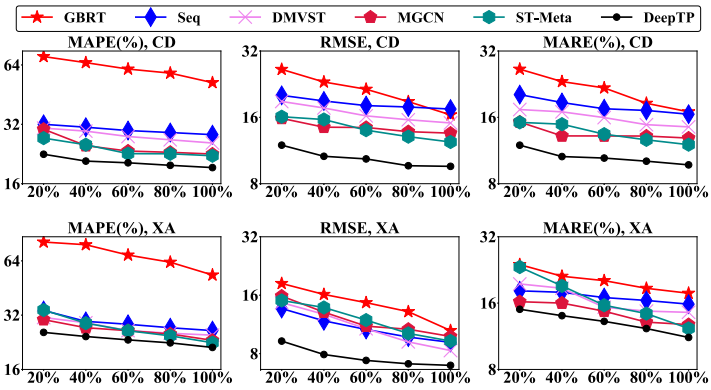


Fig. 12. MAPE & MARE & RMSE vs. the Scalability. (The title of each subfigure is labelled with the format "A, B", where "A" refers to the metric and "B" refers to the dataset.)

TABLE III
THE MAPE RESULT (%) ON TEST DATA (Δs=1km, Δt=10min, l=12)

	R-one	T-one	No-reg	No-time	No-ctx	No-con	No-ngb	No-sim	No-sd	No-pc
CD	20.32	20.07	20.53	20.47	20.44	20.87	20.75	20.35	20.36	20.46
	5.23%	3.94%	6.32%	6.01%	5.85%	8.08%	7.46%	5.39%	5.44%	5.96%
XA	23.13	22.84	23.97	23.88	23.03	23.04	22.97	22.87	22.96	22.99
	9.16%	7.79%	13.12%	12.69%	8.68%	8.73%	8.40%	7.93%	8.35%	8.49%

(5) GBRT consumes less training and estimation time than the deep learning methods (Seq, DMVST, MGCN, ST-Meta and DeepTP), because they contain the RNN module that regards the given input as a sequence and processes it sequentially.

F. Scalability Comparison

To compare the scalability of the learning based methods (i.e., GBRT, Seq, DMVST, MGCN, ST-Meta and DeepTP), we train different models by varying the training data size. In particular, we sample 20%, 40%, 60%, 80% and 100% from the training data, and collect the associated MAPE, RMSE and MARE of the online prediction over the test data. From Figure 12, we have the following observations:
(1) All methods perform better if we use more training data.
(2) Deep learning methods are more stable and effective than GBRT. E.g., the MARE of GBRT on *CD* is increased by $\frac{26.47-16.95}{16.95} = 56.17\%$ when we only use 20% training data, while the ratio is only $\frac{11.96-9.74}{9.74} = 22.79\%$ for DeepTP.
(3) Our DeepTP consistently has the best performance. In addition, in many cases (e.g., the MAPE and RMSE metrics on *CD* and *XA*), with the sampling rate decreased, the gap between our DeepTP and other methods becomes larger.

G. Ablation Test For DeepTP

Last, we evaluate the effectiveness of different parts of DeepTP by conducting an ablation test. First, we replace regions' embeddings with one-hot codes and denote the variation with R-one. Similarly, we replace the future time interval's embedding with one-hot codes and denote the variation with T-one. Second, for testing the module $\mathcal{M}_c$, we respectively remove the region embedding part, the time interval embedding part and the context encoding part, which are denoted with No-reg, No-time and No-ctx. Third, for evaluating the effectiveness of three spatial graphs (*connectivity-aware graph*, *neighbor-aware graph* and *similarity-aware graph*), we respectively remove them from DeepTP and denote them as No-con, No-ngb and No-sim. Last, we test *sd-Fusion* and *pc-Fusion* by respectively replacing them with direct concatenated operators, which are denoted as No-sd and No-pc. As shown in Table III (the percentage below the MAPE value shows the MAPE's increase percentage w.r.t. DeepTP.), we have the following observations:
(1) The region embedding model plays a more important role than the time interval embedding model in influencing the effectiveness of DeepTP, which can be deduced from the fact that T-one outperforms R-one.
(2) Based on the performance of No-reg, No-time and No-ctx, the region embedding is the most ciritcal part of $\mathcal{M}_c$, followed by the time interval embedding and the context encoding.
(3) All of the three spatial graphs are important to improving our model's effectiveness. The most significant one is *connectivity-aware graph*, followed by *neighbor-aware graph* and *similarity-aware graph*.
(4) Both *sd-Fusion* and *pc-Fusion* can impact the effectiveness of DeepTP, and *pc-Fusion* is more significant.
(5) The decreasing degree of *XA* is greater than that of *CD*. One possible reason is that the traffic data in *XA* is sparser than that in *CD*, and thereby is harder to predict.

VIII. Related Work

A. Cross-region Traffic Prediction

Cross-region traffic prediction problems regard the city as disjoint regions, and focus on the traffic data (e.g., travel demand and traffic flow) in each region. Most existing methods are proposed to solve the prediction problem for one specific type of traffic data, as listed below.

Travel Demand Prediction. The authors in [31], [2], [18] leverage similar techniques to solve the problem. Specifically, they first apply the CNN model to model region-level correlations among different regions at each historical interval and then apply the LSTM model to capture the sequential features of historical intervals. To better extract spatial-temporal features, the authors in [15] regard past traffic data as a video and then apply 3D CNN and 3D ResNet to capture the correlation between pick-up and drop-off. However, the above methods cannot well capture the semantic relationship among different regions. To address this issue, the authors in [29], [1] model the semantic relationship with graphs and then leverage GNN (Graph Neural Network) to capture them. The authors in [8] also use the GNN model but they further consider the constraint of road networks when building graphs.

Traffic Flow Prediction. The authors in [34] employ the CNN model and the Residual neural network model to encode historical traffic flow data, to predict traffic flows. To better capture the temporal features, the authors in [30], [12] leverage the LSTM model to encode traffic flow data that are modeled as sequences. Differently, the method in [30] further contains an attention mechanism for enhancing the sequential model.

B. Network-based Traffic Flow Prediction

Existing studies mainly exploit the topological structure of road networks to make traffic flow predictions, while differing in the model design. At first, some traditional machine learning algorithms, such as SVR (Support Vector Regression) [13], were proposed. Recently, deep learning become the best option due to its power of fitting any functions and the core task is how to effectively encode the spatial-temporal features in traffic flow. In particular, the authors in [16] propose to use the GCN (Graph Convolution Network) to model spatial dependency and use the RNN model to capture the temporal dynamics. However, it cannot capture the global correlations and the dynamic patterns of traffic data. To address this issue, the authors in [11] employ the attention mechanisms; the authors in [7] design a multi-resolution temporal module and a global correlated spatial module; the authors in [21] leverage the meta learning model to build meta GNN and meta RNN models to capture dynamic correlations.

Remark. Unfortunately, none of the above work, has considered the correlations among different types of traffic data, in predicting a single type of traffic data; however, as pointed out in Section I, it is critical to the prediction accuracy and it is not trivial to extend existing methods (primarily designed for a single type of traffic data prediction) to achieve a joint prediction, as various encodings need to be heavily redesigned to incorporate the inter-traffic correlations.

IX. Conclusions

In this paper, we studied the joint prediction of travel demands and traffic flows for the first time, and proposed a comprehensive neural network model `DeepTP` that seamlessly incorporated region-level correlations, temporal periodicity and inter-traffic correlations. First, we constructed three spatial graphs to respectively capture two types of region-level correlations, by which we can effectively generate regions' embeddings. Second, we built temporal graphs to initialize future time interval embeddings to capture temporal periodicity. Last, we designed an effective model to encode past traffic data with inter-traffic correlations. Extensive experiments on real datasets verified the effectiveness of our model.

References

[1] L. Bai, L. Yao, S. Kanhere, X. Wang, and Q. Sheng. Stg2seq: Spatial-temporal graph to sequence model for multi-step passenger demand forecasting. In *IJCAI*, pages 1981–1987, 2019.

[2] L. Bai, L. Yao, S. S. Kanhere, Z. Yang, J. Chu, and X. Wang. Passenger demand forecasting with multi-task convolutional recurrent neural networks. In *PAKDD*, pages 29–42, 2019.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *SIGKDD*, pages 785–794, 2016.

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734, 2014.

[5] K.-F. Chu, A. Y. Lam, and V. O. Li. Deep multi-scale convolutional lstm network for travel demand and origin-destination predictions. *TITS*, pages 1–14.

[6] J. D. Cryer. *Time series analysis*, volume 286. 1986.

[7] S. Fang, Q. Zhang, G. Meng, S. Xiang, and C. Pan. Gstnet: Global spatial-temporal network for traffic flow prediction. In *IJCAI*, pages 10–16, 2019.

[8] X. Geng, Y. Li, L. Wang, L. Zhang, Q. Yang, J. Ye, and Y. Liu. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In *AAAI*, volume 33, pages 3656–3663, 2019.

[9] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *AISTATS*, pages 315–323, 2011.

[10] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864, 2016.

[11] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI*, volume 33, pages 922–929, 2019.

[12] Z. He, C.-Y. Chow, and J.-D. Zhang. Stcnn: A spatio-temporal convolutional neural network for long-term traffic prediction. In *MDM*, pages 226–233, 2019.

[13] X. Jin, Y. Zhang, and D. Yao. Simultaneously prediction of network traffic flow based on PCA-SVR. In *ISNN*, pages 1022–1031, 2007.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR(Poster)*, 2015.

[15] L. Kuang, X. Yan, X. Tan, S. Li, and X. Yang. Predicting taxi demand based on 3d convolutional neural network and multi-task learning. *Remote Sensing*, 11(11):1265, 2019.

[16] Y. Li, R. Yu, C. Shahabi, and Y. Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *ICLR(Poster)*, 2018.

[17] B. Liao, J. Zhang, C. Wu, D. McIlwraith, T. Chen, S. Yang, Y. Guo, and F. Wu. Deep sequence learning with auxiliary information for traffic prediction. In *SIGKDD*, pages 537–546, 2018.

[18] L. Liu, Z. Qiu, G. Li, Q. Wang, W. Ouyang, and L. Lin. Contextualized spatial–temporal network for taxi origin-destination demand prediction. *TITS*, 20(10):3875–3887, 2019.

[19] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang. Traffic flow prediction with big data: a deep learning approach. *TITS*, 16(2):865–873, 2014.

[20] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *NeurIPS*, pages 2204–2212, 2014.

[21] Z. Pan, Y. Liang, W. Wang, Y. Yu, Y. Zheng, and J. Zhang. Urban traffic prediction from spatio-temporal data using deep meta learning. In *SIGKDD*, pages 1720–1730, 2019.

[22] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: online learning of social representations. In *SIGKDD*, pages 701–710, 2014.

[23] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.

[24] D. A. Tedjopurnomo, Z. Bao, B. Zheng, F. Choudhury, and A. K. Qin. A survey on modern deep neural network for traffic prediction: Trends, methods and challenges. *TKDE*, pages 1–1, 2020.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[26] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR(Poster)*, 2018.

[27] D. Wang, W. Cao, J. Li, and J. Ye. Deepsd: Supply-demand prediction for online car-hailing services using deep neural networks. In *ICDE*, pages 243–254, 2017.

[28] M. Wang, B. Lai, Z. Jin, Y. Lin, X. Gong, J. Huang, and X. Hua. Dynamic spatio-temporal graph-based cnns for traffic prediction. *arXiv:1812.02019*, 2018.

[29] Y. Xu and D. Li. Incorporating graph attention and recurrent architectures for city-wide taxi demand prediction. *Geo-Inf*, 8(9):414, 2019.

[30] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *AAAI*, volume 33, pages 5668–5675, 2019.

[31] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, and Z. Li. Deep multi-view spatial-temporal network for taxi demand prediction. In *AAAI*, pages 2588–2595, 2018.

[32] H. Yuan, G. Li, Z. Bao, and L. Feng. Effective travel time estimation: When historical trajectories over road networks matter. In *SIGMOD*, pages 2135–2149, 2020.

[33] J. Zhang, F. Wang, K. Wang, W. Lin, X. Xu, and C. Chen. Data-driven intelligent transportation systems: A survey. *TITS*, 12(4):1624–1639, 2011.

[34] J. Zhang, Y. Zheng, and D. Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *AAAI*, pages 1655–1661, 2017.

[35] Y. Zheng, Y. Liu, J. Yuan, and X. Xie. Urban computing with taxicabs. In *UbiComp*, pages 89–98, 2011.