Route Travel Time Estimation on A Road Network Revisited: Heterogeneity, Proximity, Periodicity and Dynamicity

Haitao Yuan
Tsinghua University, China
yht19@mails.tsinghua.edu.cn

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

Zhifeng Bao
RMIT University, Australia
zhifeng.bao@rmit.edu.au

ABSTRACT

In this paper, we revisit the problem of route travel time estimation on a road network and aim to boost its accuracy by capturing and utilizing spatio-temporal features from four significant aspects: heterogeneity, proximity, periodicity and dynamicity.

Spatial-wise, we consider two forms of heterogeneity at link level in a road network: the turning ways between different links are heterogeneous which can make the travel time of the same link various; different links contain heterogeneous attributes and thereby lead to different travel time. In addition, we take into account the proximity: neighboring links have similar traffic patterns and lead to similar travel speeds. To this end, we build a link-connection graph to capture such heterogeneity and proximity.

Temporal-wise, the weekly/daily periodicity of temporal background information (e.g., rush hours) and dynamic traffic conditions have significant impact on the travel time, which result in static and dynamic spatio-temporal features respectively. To capture such impacts, we regard the travel time/speed as a combination of static and dynamic parts, and extract many spatio-temporal relevant features for the prediction task.

Talking about the methodology, it remains an open problem to build a generic learning model to boost the estimation accuracy. Hence, we design a novel encoder-decoder framework – The encoder uses the sequence attention model to encode dynamic features from the temporal-wise perspective. The decoder first uses the heterogeneous graph attention model to decode the static part of travel speed based on static spatio-temporal features, and then leverages the sequence attention model to decode the estimated travel time from spatial-wise perspective. Extensive experiments on real datasets verify the superiority of our method as well as the importance of the four aspects outlined above.

PVLDB Reference Format:
Haitao Yuan, Guoliang Li, and Zhifeng Bao. Route Travel Time Estimation on A Road Network Revisited: Heterogeneity, Proximity, Periodicity and Dynamicity. PVLDB, 16(3): 393-405, 2022.
doi:10.14778/3570690.3570691

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

Proceedings of the VLDB Endowment, Vol. 16, No. 3 ISSN 2150-8097.
doi:10.14778/3570690.3570691

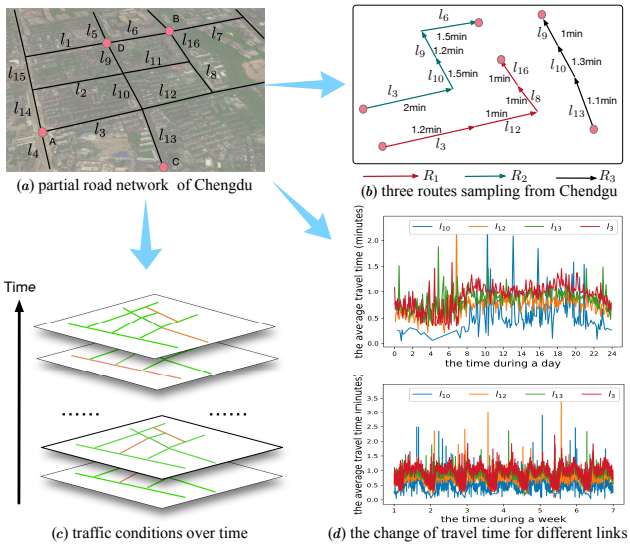


Figure 1: Some Aspects Related to Travel Time Estimation

1 INTRODUCTION

Estimating the travel time for a given route on a road network, as one of the most fundamental and frequently invoked operators in intelligent transport systems, has drawn tremendous attentions. When revisiting this problem, we find some critical yet missing aspects that contribute to its accuracy boost.

1. Spatial heterogeneity: It behaves in at least two aspects. First, the turning ways between different links in a road network are heterogeneous, which makes the travel time of the same link various (in this paper, we put the turning/waiting time between two links into the travel time of the former link). Taking the routes R_1 and R_2 in Fig. 1(b) as an example, R_1 keeps going straight while R_2 turns left after passing the link l_3, so the travel time of l_3 is absolutely different for these two routes. Second, different links contain heterogeneous attributes (e.g., different links' speed limits are different), and hence intrinsically correspond to different travel time.

2. Spatial proximity and temporal periodicity: First, neighboring links are under topological constraints of a road network, which indicates the influence of *spatial proximity*. For example, the top subfigure in Fig. 1(d) shows the travel time of four adjacent links ($l_{10}, l_{12}, l_{13}, l_3$) and we can find that they have similar patterns. Second, as shown in the bottom subfigure in Fig. 1(d), the travel time of each link exhibits periodic oscillation characteristics for a week, which results from the periodicity of temporal background (e.g., rush hours). In summary, the travel time should include a static part that is deduced based on the spatial proximity in a road network and the temporal periodicity.

3. Dynamic traffic conditions: They refer to the traffic status of a road network, where average travel speed is a natural indicator. As shown in Fig. 1(c), the traffic conditions on the road network vary over time, which reflects the influence from the so called dynamic context. Hence, it is important to capture the changing tendency of traffic condition via encoding the sequence of traffic conditions at past time slots.

Unfortunately, existing studies have not thoroughly captured or utilized most of the above aspects. For example, *statistics-based* approaches [3, 31, 32] directly estimate the travel time of a given route based on existing historical trajectories, while the influence of dynamic traffic conditions are totally ignored. In addition, these methods are not effective when historical trajectories are sparse: if there are only few trajectories passing a link, the link's travel time cannot be estimated accurately. To alleviate this issue, *learning-based* methods [16, 18, 30, 33, 41] have been proposed. They apply sequence or graph encoding models to capture the 'context' of a route or links in a road network. However, they ignore the spatial heterogeneity when encoding relevant features. Worse still, they cannot distinguish the influence of static (e.g. static proximity and periodicity) and dynamic features (e.g. dynamic traffic conditions), limiting their extension to fully exploit these features.

Actually, it remains an open problem to design a generic learning framework to incorporate all the above critical aspects, for three reasons. First, the spatial heterogeneity and proximity play opposite roles in the estimation task, which makes it difficult to learn different links' representations. Second, there are both connections and differences between the influences of static and dynamic features on the travel time, so it is non-trivial to distinguish them. Third, it remains a key challenge to design effective models to capture complex spatio-temporal correlations.

We aim to design an end-to-end model to solve this problem. First, we construct a heterogeneous graph, namely link-connection graph (**LCG**), to describe the correlation among links in a road network. To capture the spatial heterogeneity and proximity, we leverage a heterogeneous graph attention model, which can take all spatial contexts into account, on **LCG** to learn each link's representation. Second, we propose the concept of travel speed histogram to represent the distribution of travel speed/time, based on which each travel time/speed value can be divided into static and dynamic parts. Accordingly, we divide relevant spatio-temporal features into two parts, namely *sample-wise-general* and *sample-wise-special* features, based on which we can distinguish different influences to the travel time. Last, we design an encoder-decoder framework to fuse static and dynamic parts of the model in both spatial and temporal domains. In particular, we implement an encoder-decoder framework based on the attention mechanism [29], which can capture different spatio-temporal correlations. The encoder, namely *Temporal Encoder*, uses the sequence self-attention mechanism to encode dynamic temporal-wise features (e.g., traffic conditions related to the route and temporal background information) to get dynamic representations. In contrast, the decoder, namely *Spatial Decoder*, first generates static representations (i.e., static travel speed histograms) from static spatio-temporal features (e.g., link and time slot representations) and then fuses static and dynamic representations with an encode-decode attention in the spatial-wise perspective, based on which we can decode the estimated time.

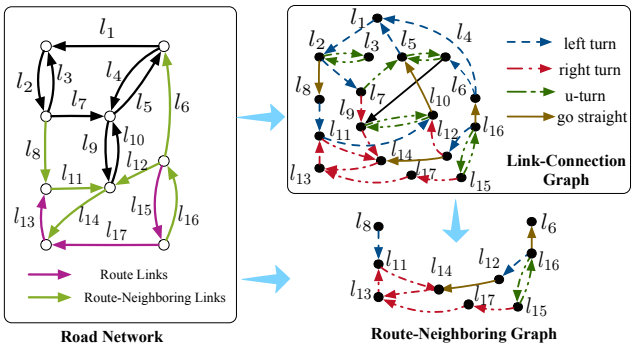


Figure 2: Road Network, Link-Connection Graph and Route-Neighboring Graph

In summary, we make the following technical contributions:

- We report four critical yet unexplored aspects towards accurate route travel time estimation, i.e. spatial heterogeneity, spatial proximity, temporal periodicity and dynamic traffic conditions, and conduct a comprehensive profiling of relevant features. (Sec. 2)
- We design a comprehensive neural network model, **STHR**, that can fully exploit different spatio-temporal features to achieve an accurate route travel time estimation. (Sec. 3)
- We leverage the self-attention mechanism to encode dynamic traffic conditions and temporal background information at each past time slot, by which we generate the dynamic part of travel time. (Sec. 4)
- We leverage the heterogeneous graph attention to learn each link's representation in a road network and generate the static part of travel time. In addition, we leverage an attention model to fuse static and dynamic parts, and decode the effective spatio-temporal representation. (Sec. 5 and Sec. 6)
- We conduct a comprehensive evaluation on two real-world datasets and find our method significantly outperforms state of the art in terms of accuracy and robustness. (Sec. 7)

2 PRELIMINARIES

We first introduce some key concepts and features useful in profiling the critical aspects towards an accurate estimation (outlined in Section 1). Then we formalize the travel time estimation problem and outline what a fine-grained estimation pipeline looks.

2.1 Basic Concepts

As shown in the left side of Fig. 2, 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. Accordingly, a route is represented as a sequence of links on a road network, which is denoted as $R = \langle l_1, l_2, \cdots, l_m\rangle$.

Link-Connection Graph (LCG) & Route-Neighboring Graph (RNG). As shown in the right side of Fig. 2, an **LCG** is built on the road network and is also modeled as a directed graph $G = \langle L, E\rangle$. L is the link set and E is the edge set; an edge $e = \langle l_i, l_j\rangle$ indicates people can directly drive from a link l_i to a link l_j. Notably, there are at most four ways (i.e., left turn, right turn, u-turn

and go straight) to turn from one link to another. Hence, the relationship between different links in G is heterogeneous. Given a route R, we call its passed links as *route links*; we call the links adjacent to those route links as *route-neighboring links*. The route-neighboring graph (**RNG**) is built on these two kinds of links, denoted as $G(R) = \langle L(R), E(R)\rangle$, where $L(R) \subseteq L$ and $E(R) \subseteq E$. Hence, **RNG** is a subgraph of **LCG**.

Travel Time & Travel Speed (Histogram). Given a link l, suppose its travel time is t_l, then its corresponding travel speed can be computed as $v = \frac{|l|}{t_l}$, where $|l|$ denotes the length of l. The reason of using travel speed is that each link's speed corresponds to a range while it is difficult to determine the range of travel time. Considering the speed range(e.g., $0 \sim 60km/h$), we split it into k disjoint parts (denoted as $p_1, \cdots, p_k$) to approximate the speed distribution $hist = [\alpha_1, \cdots, \alpha_k]$, where $0 \le \alpha_i \le 1$ and $\sum_{i=1}^{k} \alpha_i = 1$. For example, given five speed values (i.e., 5, 2, 12, 15, 20) and the two parts $p_1 = [0, 10)$ and $p_2 = [10, 20]$, the corresponding histogram would be $hist = [\frac{2}{5}, \frac{3}{5}]$. In addition, each speed value v can be computed with a speed histogram $hist = [\alpha_1, \cdots, \alpha_k]$ and a bias sequence $\Delta = [\delta_1, \cdots, \delta_k] (0 \le \delta_i \le 1)$ as follows:

$$v = \sum_{i=1}^{k} \alpha_i (p_i^{\triangleleft} + \delta_i d_i) \tag{1}$$

where $p_i^{\triangleleft}$ and $p_i^{\triangleright}$ denote the left and right value of p_i respectively, and $d_i = p_i^{\triangleright} - p_i^{\triangleleft}$ is the range of p_i.

Request & Its Travel Time Estimation (TTE). Similar to the definitions in [10], a request, $req = (R, s)$, indicates a route R that departs at time s. TTE is to estimate the travel time of R departing at time s for a given request $req = (R, s)$.

2.2 Key Features

Historical traffic conditions (Spatio-temporal features): We use the minimum, the maximum, the median and the mean travel speed (respectively denoted as $\grave{v}, \acute{v}, \tilde{v}, \bar{v}$) of links at past time slots to represent historical traffic conditions, where a time slot is set as Δt, whose default value is 5 minutes in this work. Moreover, the number of past time slots is set as p, whose default value is 12. For each route R, we consider its local traffic conditions, which correspond to the links in the **RNG** ($G(R) = \langle L(R), E(R)\rangle$), and the features are hence denoted as $C = \{(\grave{v}_l^j, \acute{v}_l^j, \tilde{v}_l^j, \bar{v}_l^j)\}_{l \in L(R)}^{j \in [s-p:s-1]}$.

Road network (Spatial-only features): We consider the graph structure of the road network by employing our link-connection graph **LCG** that incorporates heterogeneous relationships ($G = \langle L, E\rangle$). Also, we consider the properties f_l (e.g., the length, the width, the number of lanes, the type and the speed limit) for each link l in L, which is denoted as $F = \{f_l\}_{l \in L}$. In addition, for links in the **RNG** of a route R, we denote the features as $F(R) = \{f_l\}_{l \in L(R)}$.

Temporal background (Temporal-only features): The temporal background information (such as rush hours, weekdays, and other time-related information) is extracted as features. In particular, we denote such information at a time slot j as x^j. Notably, there exist $p + 1$ time slots (i.e., p past time slots and the time slot s falls in) in each sample. Hence, the temporal background information can be denoted as a sequence $X = \{x^j\}^{j \in [s-p:s]}$. Last, to capture the temporal periodicity, we consider all time slots of a week and then project s into one of these time slots [39, 40], where we use $T = \{T^j\}^{j \in [1, |T|]}$ and T^s to denote all time slots and the projected time slot of s, respectively.

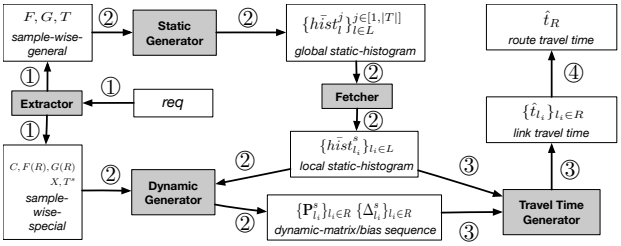


Figure 3: A Fine-grained Pipeline of Travel Time Estimation

Summary. The key features for a request include the spatio-temporal features ($C = \{(\grave{v}_l^j, \acute{v}_l^j, \tilde{v}_l^j, \bar{v}_l^j)\}_{l \in L(R)}^{j \in [s-p:s-1]}$), the spatial-only features ($F = \{f_l\}_{l \in L}$, $G = \langle L, E\rangle$, $F(R) = \{f_l\}_{l \in L(R)}$, $G(R) = \langle L(R), E(R)\rangle$), and the temporal-only features ($X = \{x^j\}^{j \in [s-p:s]}$, T, T^s). Intuitively, they can be further categorized into two parts: *sample-wise-general* and *sample-wise-special*. (1) Since F, G and T are independent of samples, they are called *sample-wise-general*. (2) Other features that rely on samples are thereby called *sample-wise-special*.

2.3 A Pipeline of Route Travel Time Estimation

Problem Definition. Given a request sample $req = \langle R, s\rangle$, assuming we can get the general features (F, G, T) and the special features $(C, F(R), G(R), X, T^s)$; the TTE problem is to estimate the travel time y of the route R based on these features by designing and learning a model $\mathcal{M}$, where $y = \mathcal{M}(C, F(R), G(R), X, T^s; F, G, T)$.

For designing the model, we propose the following pipeline of generating the travel time from a request:

$$\begin{aligned} req \xrightarrow{①} C, F(R), G(R), X, T^s, F, G, T \xrightarrow{②} hist_{l_1}^s \cdots hist_{l_m}^s, \Delta_{l_1}^s \cdots \Delta_{l_m}^s \\ \xrightarrow{③} \hat{t}_{l_1} \cdots \hat{t}_{l_m} \xrightarrow{④} \hat{t}_R = \sum_{i=1}^{m} \hat{t}_{l_i}. \end{aligned} \tag{2}$$

where we first estimate each passing link's speed histogram ($hist_{l_i}^s$) and bias sequence ($\Delta_{l_i}^s$), and then deduce its travel time ($\hat{t}_{l_i}$) based on Formula (1). Specifically, we further split each speed histogram $hist_{l_i}^s$ into two parts: the *static-histogram* ($\bar{hist}_{l_i}^s$) and the *dynamic-matrix* ($\boldsymbol{P}_{l_i}^s$), where $\bar{hist}_{l_i}^s \in \mathbb{R}^k$ is independent of samples and $\boldsymbol{P}_{l_i}^s \in \mathbb{R}^{k \times k}$ depends on each sample. Hence, we can compute $hist_{l_i}^s$ as follows:

$$hist_{l_i}^s = \bar{hist}_{l_i}^s \times Softmax(\boldsymbol{P}_{l_i}^s), \tag{3}$$

where $\times$ represents the matrix multiplication operator and the use of $Softmax(\cdot)$ helps keep $hist_{l_i}^s$ under the constraint of $\sum \alpha = 1$. Therefore, the four steps in Formula (2) can be refined as the pipeline in Fig. 3. Firstly, the *sample-wise-general* features correspond to the global information, which keeps static for different samples, so we use them to generate the *global static-histogram* $\{hist_l^j\}_{l \in L}^{j \in [1,|T|]}$ through the process "Static Generate". Secondly, for a given request with a special route R and a corresponding departure time slot T^s, we would fetch the *local static-histogram* $\{\bar{hist}_{l_i}^s\}_{l_i \in R}$ from the *global static-histogram*. Thirdly, we leverage the *sample-wise-special* features and the *local static-histogram* to

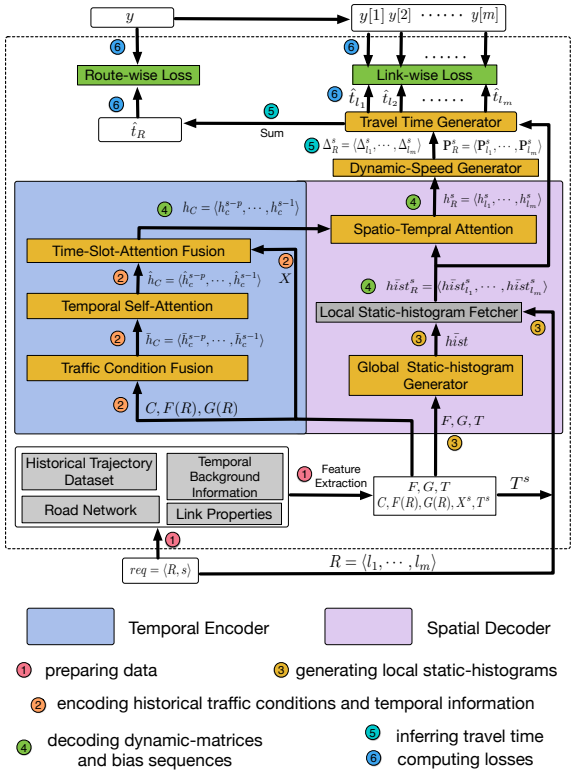


Figure 4: The Architecture and Pipeline of STHR

generate the dynamic-matrix $\{\boldsymbol{P}^s_{l_i}\}_{l_i \in R}$ and *bias sequence* $\{\Delta^s_{l_i}\}_{l_i \in R}$ through the process "Dynamic Generate". Finally, we use the *local static-histogram*, *dynamic-matrix* and *bias sequence* to infer the travel time through the process "Travel Time Generate".

3 MODEL OVERVIEW

In this section, we follow the proposed pipeline in Sec. 2.3 to describe the architecture of our system **STHR**, as well as how it incorporates the desired properties outlined in Section 1. As shown in Fig. 4, **STHR** is essentially a sequence encoder-decoder framework. In the sequence encoding part, we regard all spatio-temporal features as a temporal sequence whose length is the number of historical time slots, so we denote the encoding part as *Temporal Encoder*. As for the sequence decoding part, our goal is to generate the travel speed/time for each link in a given route. Hence, we regard the route as a sequence of links whose length is the number of links in the route, so we denote the decoding part as *Spatial Decoder*. Next, we will introduce them step by step.

1. Preparing data: As mentioned in Sec. 2.2, we extract related features for a given request $req = \langle R, s\rangle$. First, the *sample-wise-general* features (F, G, T) are extracted from road network and link properties. Second, the *sample-wise-special* features $(C, F(R), G(R), X, T^s)$ are generated based on the given request. In particular, C is extracted from historical trajectories, while X is extracted from temporal background information.

2. Encoding historical traffic conditions (C) and temporal information (X) : Given that the *Temporal Encoder* is designed to encode the temporal sequence, whose features mainly include C and X, we design some modules to encode and fuse them. Specifically, it consists of three phases:

phase 1: Since historical traffic conditions (C) are deduced based on **RNG**, it is necessary to fuse traffic conditions of all links in **RNG** at each historical time slot. Consequently, we design the *Traffic Condition Fusion* module to convert $C^j = \{(\check{v}^j_l, \acute{v}^j_l, \tilde{v}^j_l, \bar{v}^j_l)\}_{l \in L(R)}$ into the hidden representation $\bar{h}^j_c$. Accordingly, we can get the temporal sequence $\bar{h}_C = \langle \bar{h}^{s-p}_c, \cdots, \bar{h}^{s-1}_c\rangle$ because the historical time slots are from $s-p$ to $s-1$. Notably, the related spatial features (i.e., $F(R), G(R)$) are taken into account when fusing traffic conditions.

phase 2: To capture the temporal correlation, we apply the self-attention mechanism to further encode the representation sequence $\bar{h}_C$, where the module is called *Temporal Self-Attention*. In this module, each representation $\bar{h}^j_c$ is transformed into another representation $\hat{h}^j_c$, which incorporates the attention between $\bar{h}^j_c$ and each element (including $\bar{h}^j_c$) in $\bar{h}_C$. As a result, we get the new temporal sequence $\hat{h}_C = \langle \hat{h}^{s-p}_c, \cdots, \hat{h}^{s-1}_c\rangle$.

phase 3: The above encoding phases only consider historical time slots' traffic conditions and their correlations; however, the traffic conditions at different historical time slots would have different impacts on current time slot. Therefore, we design the *Time-Slot-Attention Fusion* module to further fuse the temporal sequence $\hat{h}_C$ with X, where we further convert the sequence $\bar{h}_C$ into the sequence $h_C = \langle h^{s-p}_c, \cdots, h^{s-1}_c\rangle$.

3. Generating local static-histograms: This corresponds to "Static Generate" and "Fetch" in Fig. 3, aiming to generate a static-histogram for every link of the given route at the given departure time slot. There are two phases:

phase 1: We design the *Global Static-histogram Generator* module to implement the "Static Generate" process. As a result, it generates the *global static-histogram* ($\bar{hist} = \{\bar{hist}^j_l\}^{j \in [1,|T|]}_{l \in L}$). Notably, $\bar{hist}$ is a set of histograms with the size of $|L| \times |T|$, where $|L|$ and $|T|$ respectively represent the number of all links and the number of all time slots in a week. For example, if we set the size of a time slot as 10 minutes, the value of $|T|$ would be $\frac{7\times24\times60}{10} = 1008$.

phase 2: From $\bar{hist}$, we look for the *local static-histograms* ($\bar{hist}^s_R = \langle \bar{hist}^s_{l_1}, \cdots, \bar{hist}^s_{l_m}\rangle$) via the route $R = \langle l_1, \cdots, l_m\rangle$ and the departure time slot T^s. Next, we can take the spatial sequence $\bar{hist}^s_R$ as the input of *Spatial Decoder*.

4. Decoding dynamic-matrices and bias sequences: It essentially corresponds to "Dynamic Generate" in Fig. 3, aiming to generate $\boldsymbol{P}^s_{l_i}$ and $\Delta^s_{l_i}$ for each link l_i of the given route via fusing the spatial sequence $\bar{hist}^s_R$ and the temporal encoding sequence h_C. Specifically, it is implemented in two phases:

phase 1: We design the *Spatio-Temporal Attention* module to fuse the spatial sequence and the temporal sequence. In this module, each spatial input element $\bar{hist}^s_{l_i}$ is converted into a code $h^s_{l_i}$ by using the attention mechanism to fuse it with all elements in the temporal sequence h_C. Then we get the final sequence $h^s_R = \langle h^s_{l_1}, \cdots, h^s_{l_m}\rangle$, which incorporates all spatio-temporal features.

phase 2: We design the *Dynamic-Speed Generator* module to generate $\boldsymbol{P}^s_{l_i}$ and $\Delta^s_{l_i}$ based on $h^s_{l_i}$. Specifically, we leverage a fully connected neural network to implement this module.

5. Inferring the travel time: This corresponds to the "Travel Time Generate" in Fig. 3, which is to estimate each link's travel time $\hat{t}_{l_1}$ for a given route $R = \langle l_1, \cdots, l_m \rangle$. In this module, we first use Formula (3) to compute the speed histogram $hist^s_{l_i}$ and then use Formula (1) to compute the speed value $\hat{v}_{l_i}$. Finally, we compute the travel time $\hat{t}_{l_i}$ with the formula $\hat{t}_{l_i} = \frac{|l_i|}{\hat{v}_{l_i}}$, where $|l_i|$ represents the length of l_i.

6. Computing the losses: To make our system work, it is necessary to use training data to learn the parameters in our system that needs to compute losses. On the one hand, we measure the loss for each link's travel time, which is denoted as *Link-wise Loss*. On the other hand, we measure the loss for the whole trip's travel time, which is denoted as *Route Loss*.

Roadmap. In what follows, we will elaborate the modules in *Temporal Encoder* (Sec. 4) and *Spatial Decoder* (Sec. 5), how to infer the travel time based on the encoded and decoded results (Sec. 6.1), and how to train parameters in all these modules (Sec. 6.2).

4 TEMPORAL ENCODER

In this section, we present three key components of *Temporal Encoder*: *Traffic Condition Fusion*; *Temporal Self-Attention*; *Time-Slot-Attention Fusion* (see Fig. 5). First, we explain how to incorporate the **RNG** $G(R)$ and its link features $F(R)$ to encode traffic conditions C^j at each historical time slot j in Sec. 4.1, where we fuse all links' traffic conditions for each time slot to get the fused results $\bar{h}_C = \{\bar{h}^{s-p}_c, \cdots, \bar{h}^{s-1}_c\}$. Second, to capture the temporal correlation, we explain how to leverage a self-attention mechanism to further encode $\bar{h}_C$ into the code $\hat{h}_C$ in Sec. 4.2. Last, we explain how to further encode $\hat{h}_C$ by fusing the temporal information X, which incorporates different impacts of historical time slots on the current time slot. Finally, we get the temporal encoding result h_C.

4.1 Traffic Condition Fusion With RNG

As shown in Fig. 5, the *Traffic Condition Fusion* module is designed to generate the hidden representation $\bar{h}_C = \{\bar{h}^{s-p}_c, \cdots, \bar{h}^{s-1}_c\}$ from the given historical traffic conditions $C = \{C^{s-p}, \cdots, C^{s-1}\}$. Specifically, this procedure contians two steps: encoding traffic conditions for each link in **RNG** ($G(R)$) and fusing links' representations.

Encoding Traffic Conditions. The goal of this step is to transform each link's traffic condition vector at each time slot into a dense vector. To capture the proximity among links in the heterogeneous graph $G(R) = \langle L(R), E(R) \rangle$, we use a graph attention network (GAT) model in which we apply two graph attention layers. Hence, at each time slot j ($s - q \le j \le s - 1$), we generate the corresponding representation $\bar{h}^j_c[l_i]$ for each link l_i ($1 \le i \le |L(R)|$) as follows:

$$h^j_{g_1}[i] = \boldsymbol{W}_{g_1} \cdot [\grave{v}^j_{l_i}, \acute{v}^j_{l_i}, \tilde{v}^j_{l_i}, \bar{v}^j_{l_i}] \tag{4}$$

$$e^{I(i_{1,2})}_{g_1}[i_1, i_2] = \boldsymbol{W}^{I(i_{1,2})}_{e_1} \cdot [F(l_{i_1}), F(l_{i_2})], \quad \langle l_{i_1}, l_{i_2} \rangle \in E(R) \tag{5}$$

$$\alpha^j_{g_1}[i_1, i_2] = \frac{exp(e^{I(i_{1,2})}_{g_1}[i_1, i_2] \cdot [h^j_{g_1}[i_1], h^j_{g_1}[i_2]])}{\sum_{i_3 \in \mathcal{N}(i_2)} exp(e^{I(i_{3,2})}_{g_1}[i_3, i_2] \cdot [h^j_{g_1}[i_3], h^j_{g_1}[i_2]])},$$

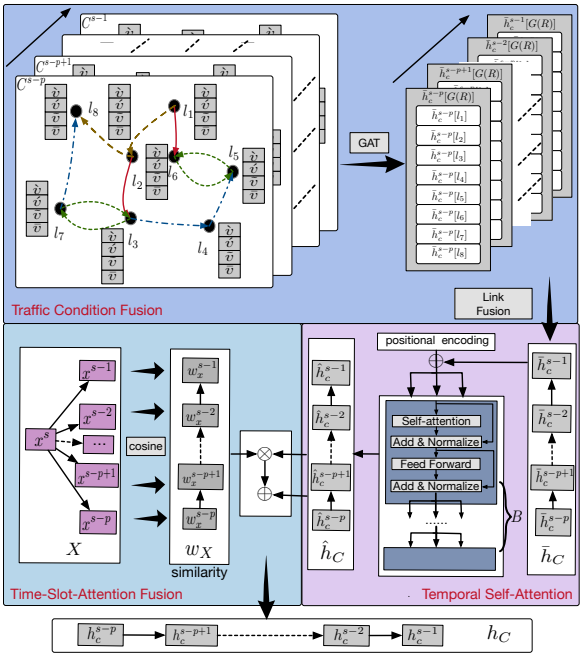


Figure 5: Elaboration of the Modules in Temporal Encoder

$$\langle l_{i_1}, l_{i_2} \rangle \in E(R) \tag{6}$$

$$h^j_{g_2}[i] = \boldsymbol{W}_{g_2}(h^j_{g_1}[i] + \sum_{i_1 \in \mathcal{N}(i)} \alpha_{g_1}[i_1, i] h^j_{g_1}[i_1]) \tag{7}$$

$$e^{I(i_{1,2})}_{g_2}[i_1, i_2] = \boldsymbol{W}^{I(i_{1,2})}_{e_2} \cdot [F(l_{i_1}), F(l_{i_2})], \quad \langle l_{i_1}, l_{i_2} \rangle \in E(R) \tag{8}$$

$$\alpha^j_{g_2}[i_1, i_2] = \frac{exp(e^{I(i_{1,2})}_{g_2}[i_1, i_2] \cdot [h^j_{g_2}[i_1], h^j_{g_2}[i_2]])}{\sum_{i_3 \in \mathcal{N}(i_2)} exp(e^{I(i_{3,2})}_{g_2}[i_3, i_2] \cdot [h^j_{g_2}[i_3], h^j_{g_2}[i_2]])}$$

$$\langle l_{i_1}, l_{i_2} \rangle \in E(R) \tag{9}$$

$$\bar{h}^j_c[l_i] = h^j_{g_2}[i] + \sum_{i_1 \in \mathcal{N}(i)} \alpha^j_{g_2}[i_1, i] h^j_{g_2}[i_1] \tag{10}$$

Here, $\boldsymbol{W}_{g_1} \in \mathbb{R}^{d_{g_1} \times 4}$ and $\boldsymbol{W}^{I(\cdot)}_{e_1} \in \mathbb{R}^{2d_{g_1} \times 2|F(\cdot)|}$ ($I(\cdot) \in \{1, 2, 3, 4\}$ is the edge type indicator) are parameters of the first graph attention layer, $\boldsymbol{W}_{g_2} \in \mathbb{R}^{d_{g_2} \times d_{g_1}}$ and $\boldsymbol{W}^{I(\cdot)}_{e_2} \in \mathbb{R}^{2d_{g_2} \times 2|F|}$ are parameters of the second graph attention layer, and $|F|$ denotes the number of link properties' dimension. More specifically, Formula (4) and (7) leverage the parameters $\boldsymbol{W}_{g_1}$ and $\boldsymbol{W}_{g_2}$ to respectively generate the two attention layers' inputs; the input of the second layer relies on the computed attentions in the first layer. Next, Formula (5) and (8) aim to generate the attention weight for each edge ($\langle l_{i_1} l_{i_2} \rangle$) in the graph $G(R)$. Considering the influence of links, we first concatenate two corresponding links' features into a vector and then multiply it with parameters. Since each edge in $G(R)$ belongs to one type, which is determined by the indicator $I(\cdot)$, we just compute one type of weight for each edge. However, there might be no edge between two nodes (links) in the graph $G(R)$, so we consider the constraint $\langle l_{i_1}, l_{i_2} \rangle \in E(R)$ when computing attention weights. Formula (6) and (9) correspond to computing the attentions in two layers, respectively. Here, $i_3 \in \mathcal{N}(i_2)$ indicates that the link l_{i_3} is adjacent to l_{i_2} and the edge $\langle l_{i_3}, l_{i_2} \rangle$ is in $E(R)$. In addition, we use an indicator $I(\cdot)$ to accurately choose the attention weight. At

last, we use neighbors' representations and the computed attentions to get each link's final representation $\bar{h}_c^j[l_i] \in \mathbb{R}^{d_{g_2}}$ in Formula (10). For the sake of simplicity, the formulas (4)-(10) are denoted as $\bar{h}_c^j[l_i] = GAT(l_i, G(R), F(R)|\boldsymbol{W}_{g_1}, \boldsymbol{W}_{e_1}, \boldsymbol{W}_{g_2}, \boldsymbol{W}_{e_2})$.

Link Fusion. At each time slot j, after getting the encoding set $\bar{h}_c^j[G(R)] = [\bar{h}_c^j[l_1], \cdots]$ for all links in the graph $G(R)$, we fuse the set into a vector $\bar{h}_c^j$ of fixed-length. One simple way is to use the average pooling technique but different links play different influence in the graph $G(R)$, so we need to measure each link's influence. Intuitively, the influence naturally depends on the link features $F(R)$. Therefore, we first compute the cosine similarity $cos(f_{l_i}, f_{l_{i'}})$ between any two links' features ($f_{l_i}, f_{l_{i'}} \in F(R)$), and then measure each link's influence score by averaging its similarity to other links, i.e., $score(l_i) = avg(\sum_{l_{i'} \in L(R)} cos(f_{l_i}, f_{l_{i'}}))$. Finally, we leverage a *Softmax* function to normalize each link's influence into a weight ($s\bar{c}ore(l_i) = \frac{score(l_i)}{\sum score(l_i')}$) and get the fused fixed-length vector $\bar{h}_c^j$ by a weighted sum, i.e., $\bar{h}_c^j = \sum_{l_i \in L(R)} s\bar{c}ore(l_i)\bar{h}_c^j[l_i]$.

4.2 Temporal Self-Attention

To capture correlations among different time slots, we apply the self-attention mechanism [29] to encode the sequence $\bar{h}_C$. At first, each element $\bar{h}_c^j$ in the sequence corresponds to a positional encoding $h_o^j \in \mathbb{R}^{d_{g_2}}$, which is computed as follows:

$$h_o^j[2d] = sin(\frac{j}{1000^{\frac{2d}{d_{g_2}}}}), \quad h_o^j[2d+1] = cos(\frac{j}{1000^{\frac{2d}{d_{g_2}}}}) \tag{11}$$

where $h_o^j[2d]$ and $h_o^j[2d+1]$ correspond to even and odd dimensions of h_o^j, respectively. Next, $\bar{h}_c^j$ would be updated by a point-wise plus operator, i.e., $\bar{h}_c^j = \bar{h}_c^j \oplus h_o^j$. Hence, the updated representation $\bar{h}_c^j$ contains the sequential order information. Next, we apply B self-attention blocks to convert the code $\bar{h}_c^j$ into the code $\hat{h}_c^j$ for each time slot j. In addition, each block includes the following steps:

1. Self-attention: For each code $\bar{h}_c^j$, we first generate three objects respectively: $query(Q_c^j = \boldsymbol{W}_q\bar{h}_c^j)$, $key(K_c^j = \boldsymbol{W}_k\bar{h}_c^j)$ and $value(V_c^j = \boldsymbol{W}_v\bar{h}_c^j)$, where $\boldsymbol{W}_q, \boldsymbol{W}_k$ and $\boldsymbol{W}_v$ are learned matrix parameters. Afterwards, we compute the score between each *query* Q_c^j and each *key* $K_c^{j'}$ with the formula $score(j, j') = softmax(\frac{Q_c^j K_c^{j'}}{\sqrt{d_{g_2}}})$. As a result, each code corresponds to a new representation $z_a^j = \sum_{j'} score(j, j')V_c^{j'}$ containing the temporal attentions. To summarize, the code $\bar{h}_c^j$ at each time slot j would be converted into a new representation z_a^j with a weighted summation of all codes, where each weight/attention measures the correlation between any two codes.

2. Add & Normalize: To relieve the gradient vanishing and explosion, we leverage the short-cut structure in ResNet [15] that uses a point-wise plus operator to update the code. In addition, we apply a normalization technique to accelerate the training process. Specifically, we leverage the layer-normalization [4] and denote the process with the function $LN(\cdot)$ for simplicity. In summary, the code would be updated with the following formula: $\bar{h}_c^j = LN(\bar{h}_c^j \oplus z_a^j)$.

3. Feed Forward: For each updated code, we further encode it with a fully connected neural network, i.e., $z_f^j = ReLU(\boldsymbol{W}_f\bar{h}_c^j + b_f)$, where $\boldsymbol{W}_f$ and b_f are learned parameters and $ReLU(\cdot)$ is the activation function.

4. Add & Normalize: Similar to the second step, we further update the code as below: $\bar{h}_c^j = LN(\bar{h}_c^j \oplus z_f^j)$.

For simplicity, we denote the above four steps in the i-th block as the function $SAB_i(\cdot)$, and denote the input and output of the i-th block as $h_{sab_{i-1}}$ and h_{sab_i}, respectively. Then we have $h_{sab_0} = \bar{h}_C$ and $h_{sab_B} = \hat{h}_C$. That is, the sequence $\hat{h}_C = \{\hat{h}_c^{s-p}, \cdots, \hat{h}_c^{s-1}\}$ is generated based on B self-attention blocks, which can be formulated as $\hat{h}_C = SAB_B(SAB_{B-1}(...SAB_1(\bar{h}_C)...))$.

4.3 Time-Slot-Attention Fusion

So far, we have encoded the traffic conditions and captured their spatio-temporal correlation at historical time slots. However, different historical time slots correspond to different impacts on the departure time slot. An ideal way is to measure traffic conditions' correlation between the departure time slot and others, but the traffic conditions at the departure time slot cannot be immediately counted. To address this issue, we exploit the temporal background information X. It can tell the similarity among different time slots, such that historical time slots' impacts on the departure time slot can be indirectly measured. Therefore, as shown in Fig. 5, we first compute the cosine similarity between the background information x^s of the departure time slot s and the background information x^j of each historical time slot j, which is formulated as $w_x^j = cos(x^s, x^j)$. Next, we regard each corresponding similarity as each historical time slot's impact, and then leverage the formula $h_c^j = \frac{(w_x^j\hat{h}_c^j) \oplus \hat{h}_c^j}{2}$ to convert each code $\hat{h}_c^j$ into a fused code h_c^j, where $w_x^j \in [-1, 1]$ helps distinguish different time slot's impact on the departure time slot by rescaling $\hat{h}_c^j$. Consequently, when considering the sequence of historical time slots, we have the fused result $h_C = [h_c^{s-p}, \cdots, h_c^{s-1}]$.

5 SPATIAL DECODER

In this section we describe how the *Spatial Encoder* works, which includes three modules (Fig. 6): *Global Static-histogram Generator*, *Local Static-histogram Fetcher* and *Spatio-Temporal Attention*. At first, in Sec. 5.1 we generate the *global static-histograms* $\bar{hist}$ based on the given **LCG** $G = \langle L, E \rangle$, the link features F and all time slots T. Since the number of $\bar{hist}$ is $|L| \times |T|$, we regard the $\bar{hist}$ as a tensor with the size $|L| \times |T| \times 4$. Next, we look up the *local static-histograms* from $\bar{hist}$. Here, given a request containing a route $R = \langle l_1, \cdots, l_m \rangle$ and a departure time s falling in the time slot T^s, the extracted static-histograms would respectively be $\bar{hist}_{l_1}^s$, $\cdots$, $\bar{hist}_{l_m}^s$. To capture the effect of *Temporal Encoder*, we apply the spatio-temporal attention model to fuse the temporal encoding code $h_C = [h_c^{s-p}, \cdots, h_c^{s-1}]$ in Sec. 5.2. As a result, we can get the fused spatio-temporal representation $h_{l_i}^s$ for each link l_i. When considering all links in the given route, we have the spatio-temporal sequence $h_R^s = [h_{l_1}^s, \cdots, h_{l_m}^s]$.

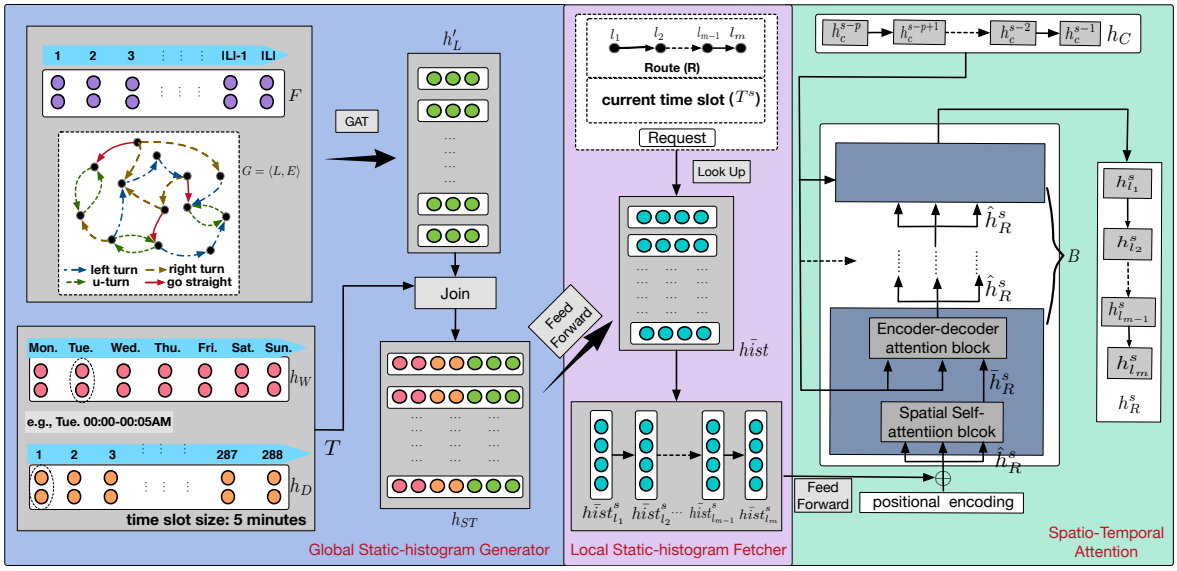


Figure 6: Elaboration of the Key Modules in Spatial Decoder

5.1 Generating Static-Histograms

As mentioned in Sec. 2.3, we first leverage the *sample-wise-general* features (G, F, T) to generate the *global static-histograms*, from which we fetch *local static-histograms* for each given request.

(1) Generating global static-histograms: Spatially, the "global" refers to all links in a road network. Hence, we consider encoding all links' features F, which can be regarded as a matrix of $|L| \times |f|$ size. One straightforward way is to design a function to transform each link's feature into a hidden representation; however, it cannot capture the correlation across different links. To address this issue, we leverage the graph neural network model GAT to encode each link's feature based on our proposed link-connection graph G. Similar to Formula (4)-(10), the GAT model employed here also contains two attention layers. For simplicity, we denote the encoded result $h'_L[l_i]$ of each each link $l_i \in L$ as $h'_L[l_i] = GAT(l_i, G, F | \boldsymbol{W}_{g'_1}, \boldsymbol{W}_{e'_1}, \boldsymbol{W}_{g'_2}, \boldsymbol{W}_{e'_2})$. where $\boldsymbol{W}_{g'_1} \in \mathbb{R}^{d_{g'_1} \times |F(\cdot)|}$, $\boldsymbol{W}_{e'_1}^{I(\cdot)} \in \mathbb{R}^{2d_{g'_1}}$, $\boldsymbol{W}_{g'_2} \in \mathbb{R}^{d_{g'_2} \times d_{g'_1}}$ and $\boldsymbol{W}_{e'_2}^{I(\cdot)} \in \mathbb{R}^{2d_{g'_2}}$ are learned parameters. Also, $I(\cdot) \in \{1, 2, 3, 4\}$ indicates the type of the edge between two links.

Temporally, the "global" refers to all time slots in a week. Inspired by [40], we split each time slot's embedding into two parts: "day-in-week" and "time-in-day", which are denoted as h_W and h_D, respectively. For example, when setting the time slot size as 5 minutes, the embeddings of the time slot (*Tue. 00:00-00:05AM*) in "day-in-week" and "time-in-day" are $h_W[2]$ and $h_D[1]$ because *Tue* is the second day in a week and *00:00-0:05AM* is the first time slot in a day. Notably, both $h_W \in \mathbb{R}^{7 \times d_w}$ and $h_D \in \mathbb{R}^{|h_D| \times d_d}$ are learned parameters, where the dimension's sizes d_w and d_d are set by users. In addition, the number of h_D is computed as $|h_D| = \frac{24 \times 60}{\Delta t}$, where Δt denotes the time slot's size and is also set by users.

Last, we join the spatial encodings $(\bar{h'}_L)$ and the two temporal embeddings (h_W and h_D) by a cartesian product, resulting in $|L| \times 7 \times |h_D|$ spatio-temporal embeddings, which are denoted as $h_{ST} \in \mathbb{R}^{(|L| \times 7 \times |h_D|) \times (d_{g'_2} + d_d + d_w)}$. To get the *global static-histograms*, we further leverage the *Feed Forward* model to decode each spatio-temporal embedding $h_{ST}[i]$ into a static-histogram, which is denoted as $\bar{hist}[i] = Softmax(\boldsymbol{W}_{st} h_{ST}[i] + b_{st})$. In particular, $\boldsymbol{W}_{st} \in \mathbb{R}^{k \times (d_{g'_2} + d_d + d_w)}$ and $b_{st} \in \mathbb{R}^k$ are learned parameters and $Softmax(\cdot)$ is the activation function. The reason of using $Softmax(\cdot)$ is that we need to guarantee $\sum_{\alpha_j \in \bar{hist}[i]} \alpha_j = 1$.

(2) Fetching local static-histograms. Given a route $R = \langle l_1, \cdots, l_m \rangle$ and a departure time s (its projected time slot is T^s), we can successively fetch the static-histogram $\bar{hist}^s_{l_i}$ for each link l_i in the route. Specifically, the order of $\bar{hist}^s_{l_i}$ in $\bar{hist}$ is computed as $(L(l_i) - 1) \times (7 \times |h_D|) + T^s[W] \times |h_D| + T^s[D]$, where $L(l_i)$ denotes the order of l_i in L, and $T^s[W]$ and $T^s[D]$ denote the order of T^s in "day-in-week" and "time-in-day" respectively. Consequently, the static-histogram sequence is $\bar{hist}^s_R = [\bar{hist}^s_{l_1}, \cdots, \bar{hist}^s_{l_m}]$.

5.2 Spatio-Temporal Attention

In this section, we first leverage the feed forward model to transform each $\bar{hist}^s_{l_i}$ into a representation $\hat{h}^s_{l_i}$, denoted as $\hat{h}^s_{l_i} = ReLU(\boldsymbol{W}_{hr} \bar{hist}^s_{l_i} + b_{hr})$. Here, $\boldsymbol{W}_{hr} \in \mathbb{R}^{d_{hr} \times k}$ and $b_{hr} \in \mathbb{R}^{d_{hr}}$ are learned parameters, where b_{br} is the dimension size of $\hat{h}^s_{l_i}$. Next, we use Formula (11) to compute the positional encoding h^i_o for $\bar{h}^s_{l_i}$. In particular, we update $\bar{h}^s_{l_i}$ with the formula $\bar{h}^s_{l_i} = \bar{h}^s_{l_i} \oplus h^i_o$. Then we get the updated sequence $\bar{h}^s_R$ when considering all links in the route. Afterwards, in order to capture the influence of *Temporal Encoder*, we exploit the attention mechanism to fuse the temporal sequence h_C and $\bar{h}^s_R$. In particular, we design B spatio-temporal attention blocks and each block contains two sub-blocks: spatial self-attention sub-block and encoder-decoder attention sub-block. Next, we will explain these two sub-blocks.

(1) Spatial self-attention: Similar to Sec.4.2, we also use the self-attention mechanism to capture the correlation of elements in the

Algorithm 1: Model Learning for **STHR**

Input: training inputs Z, training labels Y, three parts of the whole model ($\mathcal{M}_{enc}$, $\mathcal{M}_{dec}$, $\mathcal{M}_{est}$), the time interval size Δt, learning rate lr, training epochs ep, batch size bs, loss weight λ.

Output: parameters $\theta_{enc}, \theta_{dec}, \theta_{est}$ for the three parts $\mathcal{M}_{enc}$, $\mathcal{M}_{dec}$ and $\mathcal{M}_{est}$

1 $G, F \leftarrow$ generating **LCG** and link features based on road network;
2 $T \leftarrow$ generating time slots for a week based on Δt;
3 initialize parameters $\theta_{enc}, \theta_{dec}, \theta_{est}$ with normal distribution;
4 **for** $i \leftarrow 1 \cdots ep$ **do**
5 $\quad \theta_{enc}, \theta_{dec}, \theta_{est} \leftarrow$ $ModelTrain(Z, Y, \mathcal{M}_{enc}, \mathcal{M}_{dec}, \mathcal{M}_{enc}, lr, bs, \lambda, G, F, T)$; using $\theta_{enc}, \theta_{dec}, \theta_{est}$ to update $\mathcal{M}_{enc}$, $\mathcal{M}_{dec}$ and $\mathcal{M}_{est}$;

Function ModelTrain

Input: $Z, Y, \mathcal{M}_{enc}, \mathcal{M}_{dec}, \mathcal{M}_{enc}, lr, bs, \lambda, G, F, T$

1 training iterations $TI = \lfloor \frac{|X|}{bs} \rfloor$, $shuffle(Z, Y)$;
2 **for** $i \leftarrow 1 \cdots TI$ **do**
3 $\quad$ collect $Z_{(i-1)b_s+1:i\times b_s}, Y_{(i-1)b_s+1:i\times b_s}$;
4 $\quad [(R, s, C, X)] \leftarrow Z_{(i-1)b_s+1:i\times b_s}$;
5 $\quad [(F(R), G(R), T^s)] \leftarrow$ generating based on G, F, T, $[(R, s)]$;
6 $\quad [h_C] \leftarrow \mathcal{M}_{enc}([(C, F(R), G(R), X)])$;
7 $\quad [h_R^s] \leftarrow \mathcal{M}_{dec}([(R, T^s, h_C)]|G, F, T)$;
8 $\quad [(\hat{t}_{l_1}, \cdots, \hat{t}_{l_m})] \leftarrow \mathcal{M}_{est}([h_R^s])$, $[\hat{t}_R] = [\sum_j^m \hat{t}_{l_j}]$;
9 $\quad loss = \sum(\lambda \frac{\sum_j |\hat{t}_{l_j} - y[j]|}{m} + (1-\lambda)|\hat{t}_R - \sum_j y[j]|)$;
10 $\quad \theta_{enc}, \theta_{dec}, \theta_{est}, \leftarrow AdamOpt(loss, lr)$;
11 return $\theta_{enc}, \theta_{dec}, \theta_{est}$;

spatial sequence $\hat{h}_R^s$, which results in a representation sequence $\bar{h}_R^s$. This sub-block contains four steps: *Self-attention→Add&Normalize →Feed Forward→Add&Normalize*. For simplicity, the above process in the i-th block is denoted as $\bar{h}_R^s = SAB'_i(\hat{h}_R^s)$.

(2) Encoder-decoder attention: As shown in Fig. 6, we take the first sub-block's output $\bar{h}_R^s$ as the *query* object and take the temporal representation sequence h_C as both *key* and *value* objects. That is, we aim to capture the attention of h_C on each element in $\bar{h}_R^s$. This sub-block also contains four steps: *Encoder-encoder Attention→Add&Normalize→Feed Forward→Add&Normalize*. Notably, the difference with the self-attention is the first step. For simplicity, the above process in the i-th block is denoted as $\hat{h}_R^s = EDA'_i(\bar{h}_R^s, h_C)$.

To summarize, the process in the i-th spatio-temporal block can be formulated as $\hat{h}_R^s = EDA'_i(SAB'_i(\hat{h}_R^s), h_C)$. Considering that we have B blocks, the B-th block's output corresponds to the final spatio-temporal sequence h_R^s.

6 ROUTE TRAVEL TIME ESTIMATION

We first explain how to generate the travel time based on the spatio-temporal sequence h_R^s and the local static-histograms $\bar{hist}_R^s$ (Sec. 6.1), and then describe the training process (Sec. 6.2).

6.1 Inferring Travel Time

As shown in Fig. 4, we design two modules, "Dynamic-Speed Generator" and "Travel Time Generator", to infer the travel time based on h_R^s and $\bar{hist}_R^s$. The first module aims to generate the speed-dynamic factors (i.e., dynamic-matrices $\{\boldsymbol{P}_{l_i}^s\}_{i\in[1,m]}$ and the bias sequences $\{\Delta_{l_i}^s\}_{i\in[1,m]}$). The second module at first generates the travel speed for each link l_i based on its static-histogram $\bar{hist}_{l_i}^s$ and speed-dynamic factor ($\boldsymbol{P}_{l_i}^s$ and $\Delta_{l_i}^s$), and then infers its travel time based on the generated travel speed and its length. Details are presented below.

(1) Generating dynamic factors: In this part, we leverage two feed forward models to generate $\boldsymbol{P}_R^s = \{\boldsymbol{P}_{l_i}^s\}_{i\in[1,m]}$ and $\Delta_R^s = \{\Delta_{l_i}^s\}_{i\in[1,m]}$ respectively based on the spatio-temporal sequence h_R^s. That is, given a spatio-temporal code $h_{l_i}^s$ in the sequence h_R^s, the two corresponding dynamic factors are computed as follows:

$$\boldsymbol{P}_{l_i}^s = Reshape(\boldsymbol{W}_P h_{l_i}^s + b_P, k \times k) \tag{12}$$

$$\Delta_{l_i}^s = Sigmoid(\boldsymbol{W}_\Delta h_{l_i}^s + b_\Delta) \tag{13}$$

where $\boldsymbol{W}_P \in \mathbb{R}^{(k^2 \times d_{hr})}$, $b_P \in \mathbb{R}^{d_{hr}}$, $\boldsymbol{W}_\Delta \in \mathbb{R}^{(k \times d_{hr})}$, $b_\Delta \in \mathbb{R}^k$ are learned parameters. k denotes the dimension size of speed histogram, $Reshape(A, k \times k)$ is to convert the vector $A \in \mathbb{R}^{k^2}$ into a matrix of size $k \times k$, and the function $Sigmoid(z) = \frac{1}{1+e^{-z}}$ helps keep $\Delta_{l_i}^s$ under the constraint of $0 \le \delta_{l_i}^s \le 1$, $\forall \delta_{l_i}^s \in \Delta_{l_i}^s$.

(2) Generating travel time: At first, according to Formula (3), for each link l_i, we generate the dynamic-histogram $hist_{l_i}^s$ based on the static-histogram and the dynamic-matrix. Next, according to Formula (1), we generate its corresponding estimated speed $\hat{v}_{l_i}$ based on $hist_{l_i}^s$ and $\Delta_{l_i}^s$. At last, we can compute the estimated travel time with the formula $\hat{t}_{l_i} = \frac{|l_i|}{\hat{v}_{l_i}}$. By considering all links in the given route, we have the total travel time $\hat{t}_R = \sum_{i=1}^m \hat{t}_{l_i}$.

6.2 Learning Model

Offline training. Algorithm 1 outlines the training process. At first, we extract the global spatio-temporal features (G, F, T). Specifically, G and F are extracted from the whole road network while F is computed based on the given time slot size Δt. Next, we initialize all parameters θ_{enc}, θ_{dec} and θ_{est} for the whole model with a normal distribution (lines 1-3). Then we iteratively train the whole model with the given epochs ep (lines 4-5). *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 training data X, Y (line 1). In each iteration, we extract b_s training data from Z, Y. Each element of Z is composed of the request (the route R and the departure time s), the historical traffic conditions C, and the temporal background X. In addition, we build the local graph **RNG** ($G(R)$), and extract local link features ($F(R)$) and the current time slot (T^s) based on the global information G, F, T and each element's R and s (lines 2-5). In particular, we first use the part $\mathcal{M}_{enc}$ (see Sec. 4) to generate the code h_C to represent spatial encoding results, and then use the part $\mathcal{M}_{dec}$ (see Sec. 5) to decode the representation h_R^s. Afterwards, we use the part $\mathcal{M}_{est}$ (see Sec. 6.1) to generate each link's estimated travel time and then compute the overall travel time for the given route (line 6-8). At last, we design the following loss function: $loss = \sum(\lambda \frac{\sum_j |\hat{t}_{l_j} - y[j]|}{m} + (1-\lambda)|\hat{t}_R - \sum_j y[j]|)$. It consists of two types of mean absolute losses: link-wise and route-wise. The former can help the model learning in a fine-grained mode. We utilize Adam Optimizer [20] to optimize all parameters by minimizing the total loss $loss$ (lines 9-10).

7 EXPERIMENTS

In this section, we would like to evaluate the effectiveness, efficiency and scalability of our model **STHR**.

7.1 Experimental Setup

Datasets. (1) **Road Networks & Time Intervals.** We use two road networks: *Chengdu Road Network (CRN)* and *Xi'an Road Network (XRN)* extracted from OpenStreetMap [2]. *CRN* includes 3, 191 vertices and 9, 468 edges; *XRN* contains 4, 576 vertices and 12, 668 edges. To verify the scalability, we use a larger road network, *Nanjing Road Network (NRN)*, which includes 121, 271 vertives and 343, 276 edges. We set the size of a time slot (Δt) as 5, 10, 30 and 60 minutes for robustness evaluation, where the default is 10 minutes.

(2) **Requests & Traffic Conditions.** We extract the requests based on the taxi orders in *Chengdu (CD)* and *Xi'an (XA)*, collected from Didi Chuxing [1]. Each order corresponds to a trajectory and we can extract its request and departure time as the request. There are $5.8M$ and $3.4M$ orders in *CD* and *XA* [39, 40] respectively, both from 10/01/2016 to 11/30/2016. In addition, we collect $13M$ orders on *NRN* from 01/01/2011 to 31/01/2011, which is called *Nanjing (NJ)*. The total number of time slots can be calculated via $\frac{k\times 24\times 60min}{\Delta t}$, where k is set as 61 for *CD*/*XA* and 31 for *NJ*. Hence, we can compute each link's travel speed at each time slot and use it as the traffic conditions based on these trajectories. We set p, the number of past time slots, as 6, 12, 24, 48 for robustness evaluation, and the default is 12.

(3) **Temporal Background Information.** Two types of temporal information, namely holiday/weekend indicator (1 for holiday/weekend, 0 otherwise) and rush-hour indicator (1 for rush hour, 0 otherwise), are used.

(4) **Training, Validation and Test.** Since we can get $\frac{24\times 61\times 60min}{\Delta t}$ time slots 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%.

Baseline methods. We compare our models with six methods:

- **Avg**: We estimate the travel time by averaging the travel time of all historical trajectories falling in the same time slot for each link, whose default travel time is set as the average value of records during all time slots.
- **Sim** [31]: This is a nearest neighbor based approach, which estimates the travel time by averaging the travel time of all historical trajectories falling in the same departure time with similar origin and destination points, where the road distance between two similar points is less than 100 meters. For better estimation, we ignore those trajectories whose routes are different from the given route based on the similarity function LCRS ([37]), and the default similarity threshold is 0.5.
- **DTravel** [41]: This is an end-to-end method. It first extracts spatial and temporal features and then employs bidirectional LSTM and auxiliary tasks based on dual intervals.
- **STANN** [16]: This is a spatio-temporal graph neural network. It first encodes the spatial information by graph attention and then encodes the temporal information by LSTM and attention mechanism.
- **ConST** [10]: This is also a spatio-temporal graph neural network. It captures the local contextual spatio-temporal features for each link, and then applies the multi-task learning to capture the correlation between different links.
- **Ens** [18]: This method is an ensemble of traditional learning methods (e.g., GBDT) and deep learning methods (e.g., CNN and LSTM). It is worth mentioning that this method won the first place in the 2021 SIGSPATIAL Cup .

Notably, the last four methods are *learning-based* methods. For fair comparison, we make their parameter scale (a.k.a., model size) similar to ours, which is shown in Sec. 7.5.

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 [20] as the optimization method with the mini-batch size of 500. The initial learning rate was 0.001.

Evaluation metrics. We evaluate our proposed methods and baseline methods based on three metrics: MSE (Mean Square Error), MAE (Mean Absolute Error) and MAPE (Mean Absolute Percent Error), which are widely adopted by the baselines we compare with. Specifically, suppose the ground truth is represented as $\mathrm{y} = \{y^i\}$ and the predicted result is denoted as $\hat{\mathrm{y}} = \{\hat{y}^i\}$, where $1 \le i \le N$, these metrics are computed as follows: $MSE(\mathrm{y}, \hat{\mathrm{y}}) = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{y}^i)^2$; $MAE(\mathrm{y}, \hat{\mathrm{y}}) = \frac{1}{N}\sum_{i=1}^{N}|y^i - \hat{y}^i|$; $MAPE(\mathrm{y}, \hat{\mathrm{y}}) = \frac{1}{N}\sum_{i=1}^{N}|\frac{y^i - \hat{y}^i}{y^i}|$.

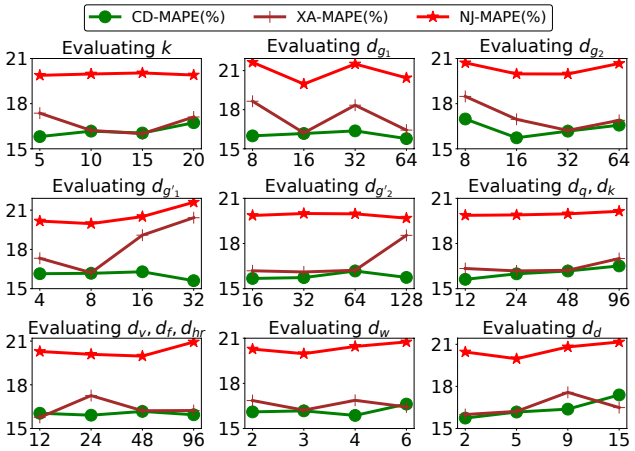


Figure 7: MAPE(%) vs. Hyper-parameters

7.2 Setting of Model's Hyper-parameters

we consider the following hyper-parameters: (1) the size (k) of each speed histogram; (2) the sizes ($d_{g_1}, d_{g_2}, d_{g'_1}, d_{g'_2}$) of different layers' neural networks in two GAT models; (3) the settable-dimension sizes (d_q, d_k, d_v, d_f) of parameters ($\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v, \boldsymbol{W}_f$) in sequence attention models, where $d_q = d_k$ and $d_f = d_v$; (4) the settable-dimension size (d_{hr}) of parameters ($\boldsymbol{W}_P, \boldsymbol{W}_\Delta$), where $d_{hr} = d_f$; (5) the embedding sizes (d_w, d_d) of time slots. 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*, *XA* and *NJ* to determine its optimal value. As shown in Fig. 7, we plot the MAPE for different hyper-parameters. In summary, we set each

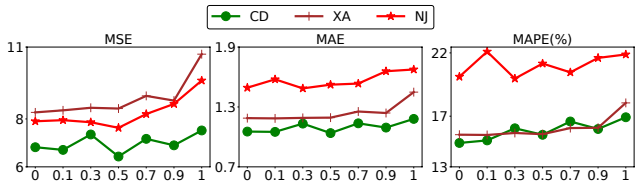


Figure 8: Loss on Validation Data vs. the Loss Weight λ

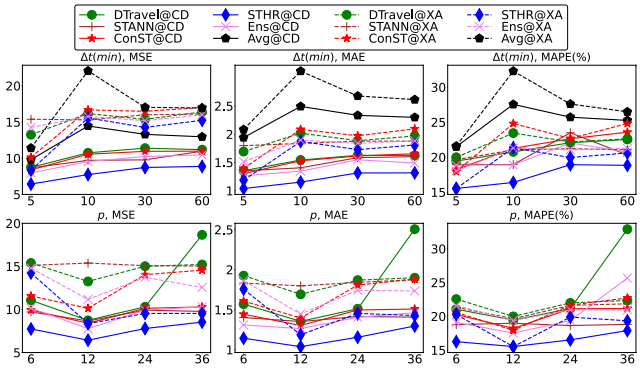


Figure 9: MSE & MAE & MAPE vs. Variables Δt & p. (The title of each subfigure is labelled in form of "A, B", where "A" and "B" respectively refer to one kind of variable and a metric. Each legend "C@D" means the method "C" evaluated on the dataset "D".)

hyper-parameter with the value corresponding to the optimal performance as follows: (1) For *CD*, we have $k = 5$, $d_{g_1} = 64$, $d_{g_2} = 16$, $d_{g'_1} = 32$, $d_{g'_2} = 32$, $d_q = d_k = 12$, $d_v = d_f = d_{hr} = 24$, $d_w = 4$, $d_d = 2$. (2) For *XA*, we have $k = 15$, $d_{g_1} = 64$, $d_{g_2} = 32$, $d_{g'_1} = 8$, $d_{g'_2} = 32$, $d_q = d_k = 24$, $d_v = d_f = d_{hr} = 12$, $d_w = 3$, $d_d = 2$. (3) For *NJ*, we have $k = 20$, $d_{g_1} = 16$, $d_{g_2} = 16$, $d_{g'_1} = 8$, $d_{g'_2} = 32$, $d_q = d_k = 24$, $d_v = d_f = d_{hr} = 48$, $d_w = 3$, $d_d = 5$.

7.3 Effectiveness of Loss Weight

To fine-tune the loss weight λ, we vary it from 0 to 1 when training **STHR**. We compute the three metrics for the validation data. The result is plotted in Figure 8 from which we find: 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 link-wise loss and the route-wise loss. Based on the majority voting rule, the best values of λ for *CD*, *XA* and *NJ* are respectively 0.5, 0.1 and 0.3, which are also set as the default values in subsequent experiments.

7.4 Effectiveness Comparison

Apart from comparing **STHR** with baseline methods, we replace our **STHR** by four variations, namely **NR**, **NL**, **NT** and **NS**, to evaluate the effectiveness of different parts of encodings in **STHR**. In **NR**, we leverage fully connected neural networks (FCN) to fuse traffic conditions (i.e., the GAT model on **RNG**). In **NL**, we use FCN to generate static-histograms (i.e., the GAT model on **LCG**). In **NT**, we use FCN to replace temporal self-attention networks. In **NS**, we use FCN to replace spatio-temporal attention networks.

Table 1: Effectiveness Results on Test Data($\Delta t = 5min$, $p = 12$)

	CD			XA		
Method	MSE	MAE	MAPE(%)	MSE	MAE	MAPE(%)
Avg	9.79	1.94	21.49	11.40	2.08	21.58
Sim	9.89	1.99	22.54	18.09	2.80	30.44
DTravel	8.74	1.35	19.45	13.25	1.69	19.99
STANN	8.66	1.35	18.98	15.39	1.80	19.65
ConST	8.39	1.30	19.22	10.13	1.40	17.94
Ens	7.96	1.26	18.85	14.25	1.51	18.03
NR	6.90	1.07	15.60	8.72	1.22	15.74
NL	6.95	1.09	15.48	8.44	1.20	15.65
NT	7.03	1.11	16.32	8.53	1.24	16.78
NS	7.09	1.10	16.45	8.98	1.28	16.72
STHR	**6.43**	**1.04**	**15.53**	**8.43**	**1.19**	**15.58**

Table 1 reports the evaluation results of all methods with the default settings of Δt and p, and we have the following observations:
(1) **Avg** and **Sim** are worse than deep learning based methods, because the former can approximately fit any function. In addition, it is not good enough to give accurate predictions when historical data are sparse for **Avg** and **Sim**.
(2) When looking into the results of **NR**, **NL**, **NT**, **NS** and **STHR**, we find that the spatial-temporal attention is the most critical part of **STHR**, followed by the temporal self-attention, and the GAT models on **RNG** and **LCG**.
(3) **STHR** performs the best on all metrics. For example, **STHR** outperforms the best existing methods (i.e., **ConST** and **Ens**) by 20% on MAE for the test data of *CD*. The reason is that our method can capture heterogeneity, proximity, periodicity and dynamicity.
(4) When comparing errors on *CD* and *XA*, the performance of all methods on *CD* is better than that on *XA*. The reason is two-fold: first, most routes of *XA* are longer than those of *CD*, and intuitively it is more difficult to accurately estimate longer travel time; second, the size of *CD* is larger than that of *XA* and more data often lead to better training results for neural networks.

Furthermore, we evaluate the robustness of different models by respectively varying the values of Δt and p in Figure 9. We have the following observations:
(1) With the increase of Δt, the errors of almost all methods increase, because the traffic status at bigger time slots would be more dynamic, which could improve the prediction difficulty. However, the method **Avg**'s performance would be better when the size of Δt exceeds 10; the reason is that larger time slots correspond to more trips for each link and hence make it more credible to use the average of historical travel time.
(2) With the increase of the size p, all three metrics show the same trend for each method: the performance first improves and then is much worsened when exceeding a certain value. This is because there is a trade-off between the accuracy of large p and small p. A bigger p leads to more data, making it more accurate to predict the travel time. However, the longest past traffic data have less influence to future traffics, so the bigger p would lead to more noisy inputs for the prediction.
(3) No matter how Δt and l change, our method **STHR** consistently has the best performance.

Table 2: Efficiency of Test Result (Δt=5min, p=12)

	dataset	Avg	Sim	DTravel	STANN	ConST	Ens	STHR
memory	*CD*	147M	147M	501K	1.5M	1.4M	1.4M	1.5M
(MByte)	*XA*	98M	73M	501K	1.4M	1.3M	1.3M	1.3M
training	*CD*	-	-	8.88	6.17	12.98	10.34	11.13
(minutes/ep)	*XA*	-	-	4.47	3.17	6.76	5.13	5.16
estimation	*CD*	0.03	0.22	0.20	0.58	0.67	0.35	0.39
(seconds/K)	*XA*	0.01	0.03	0.23	0.27	0.37	0.25	0.31

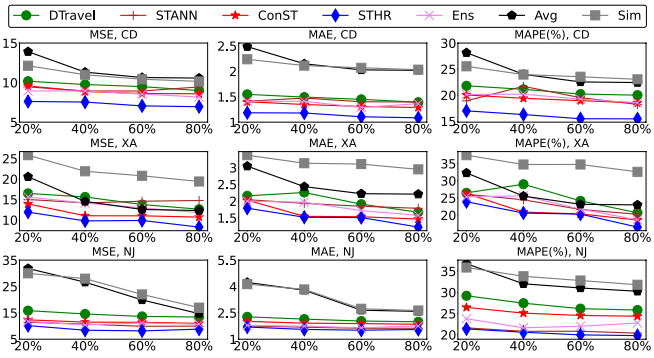


Figure 10: MAPE & MARE & RMSE vs. the Scalability. (The title of each subfigure is labelled in form of "A, B", where "A" refers to the metric and "B" refers to the dataset.)

7.5 Efficiency Comparison

We use the memory usage, training time and estimation time for efficiency evaluation. The memory usage represents the size of required memory for applying corresponding methods, and it is used to evaluate the memory efficiency. The training time is used to evaluate the offline learning efficiency for neural network based methods. In particular, we compute the average time of an epoch for each method. The estimation time can evaluate the online prediction efficiency. Specifically, we use different methods to estimate the results for 1,000 routes and record the latency respectively. The results are reported in Table 2. We observe the following:

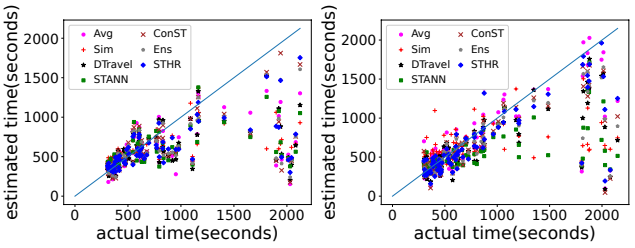
(1) **Avg** and **Sim** require more memory than others. The reason is that they need to load data info whose size is proportional to the size of historical trajectories, while other learning methods only need to load model parameters whose size is smaller.

(2) **DTravel** has the same model size for all datasets, but other learning methods are different. The reason is that other methods consider the size of road network, which varies from one city to another. As a result, the model size of **DTravel** is much less than that of other learning methods.

(3) Due to the computation complexity of graph neural network, **ConST** and **STHR** consume more training and estimation time than **DTravel**, **STANN** and **Ens**.

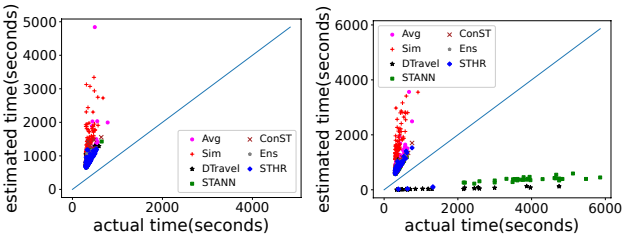
7.6 Scalability Comparison

We evaluate the scalability of all methods by varying the training data size. In particular, we sample 20%, 40%, 60% and 80% from the training data, and collect the associated MAPE, RMSE and MARE of the online prediction over the test data. From Figure 10, we have the following observations:



(a) CD (b) XA

Figure 11: Estimated time vs. actual time



(a) CD (b) XA

Figure 12: Estimated time vs. actual time on MAPE

(1) All methods perform better if more training data is used. The reason is that more data indicates more situations, making the model learned better.

(2) Deep learning methods are more stable and effective than **Avg** and **Sim**. For example, the MAPE of **Avg** on *CD* is increased by $\frac{28.15-21.49}{21.49} = 30.99\%$ when we only use 20% training data, while the ratio is only $\frac{17.00-15.53}{15.53} = 9.47\%$ for **STHR**.

(3) Our **STHR** consistently has the best performance. In addition, in many cases (e.g., the MAE and MAPE metrics on *CD* and *XA*), with the sampling rate decreased, the gap between our **STHR** and other methods becomes larger.

7.7 Case Study

We first randomly sample 50 test data from *CD* and *XA* respectively, and then use different methods to generate the estimated travel time. After that, we get 50 pairs of the actual time and the estimated time for each method. We illustrate all pairs with scatter points, as plotted in Figure 11. In each figure, we draw an auxiliary line $y = x$ as reference. We can find:

(1) Most of our **STHR** model's points are closer to the reference line than other methods. (2) With the increase of actual time duration, the errors of estimated time also increase for all methods, but **STHR** has the smallest degree of increase.

To study the performance of each method in the worst case, we select 50 worst-performing cases for each method. Similarly, we draw them in Figure 12. Specifically, we compare them based on the MAPE loss. According to the definition of MAPE, shorter actual travel time and longer estimated travel time would cause a bigger MAPE loss. Therefore, almost all selected cases were located in the upper left corner of the corresponding figures. We can find that our method **STHR** is closer to the reference line than other methods in most cases.

8 RELATED WORK

8.1 Travel Time Estimation

There are two broad categories of work – travel time estimation for OD (Origin-Destination) and travel time estimation for routes.

Travel time estimation for OD inputs. The problem of travel time estimation for OD inputs [5, 9, 17, 19, 22, 23, 39] aims to estimate the arrival time for a given origin and destination points, and the only available data are the two points and the departure time. In particular, the authors in [19] propose a multi-layer neural network called `STNN`. They first predict the travel distance based on a given origin and a given destination, and then they combine the predicted distance with the given temporal information to predict the travel time. However, they neglect the information about road network. Hence, Li et al. [23] leverage road topological structure and spatio-temporal information of road network to predict travel time. However, they directly embed the longitude and latitude of the origin and the destination, which cannot accurately capture the spatial features on the road network. In addition, they ignore historical trajectories, which are useful for travel time estimation. Therefore, Yuan et al. [39] propose a novel neural network based prediction model, which can be trained with historical trajectories and can be used to estimate without trajectories.

Travel time estimation for routes. The method of estimating travel time for paths can be divided into two groups: *statistics-based* and *learning-based*. The *statistics-free* approaches estimate the travel time of a route based on historical recorded trajectories. For example, Wang et al. [30] and Amirian et al. [3] first leverage historical trajectories to estimate the travel time of each link, and then deduce a route's travel time with the summation of its covered links' travel time. However, they cannot capture the correlation between adjacent links, which plays an important role on the travel time. For instance, the traffic lights in link intersections determine the waiting time on the corresponding road segments. To address this issue, Wang et al. [31] use similar historical trajectories to estimate the travel time for a given route, but this method is not effective when there are only few similar historical trajectories. Therefore, some *learning-based* methods [10, 16, 30, 33, 41] are proposed to solve the problem. In particular, they design different models, which are trained based on historical trajectories, to generate travel time for given routes. At first, some people [30, 33, 41] regard the route as a sequence of links, and apply some sequence encoding model (e.g., LSTM) to achieve the goal. In addition, considering the constraint of road network, He et al. [16] first apply the graph model to encode spatial information, and then apply the sequence model to generate travel time. However, sequence models are inefficient due to its recurrent-based structure, especially for the route containing hundreds of links. To address this issue, Fang et al. [10] leverage the graph attention model to independently capture each link's contextual features for a given route. In addition, Huang et al. [18] design an ensemble framework to integrate traditional and deep learning methods. However, existing methods cannot fully exploit spatio-temporal features from the following aspects: heterogeneity, proximity, periodicity and dynamicity. These inspire us to design our efficient models. Some deep learning approaches [24, 40] are designed to forecast traffic, but they need to fix the number of future time intervals, such as one hour or 10 minutes, which is essentially different from the ETA problem.

8.2 Deep Learning for Spatio-temporal Data

With the development of artificial intelligence, there is an increasing growth of deep learning applications in spatio-temporal data management or mining. First, Recurrent Neural Network (RNN) is recently applied to trajectory modeling. For example, Wu et al. [34] predict next movement through modeling trajectory with RNN and outperforms existing shallow models. The authors in [13] represent and identify the semantics of user mobility patterns by embedding trajectories with RNN model. In addition, Dong et al. [8] design a stacked RNN model to characterize the driving style of different drivers. Second, many studies focus on Convolutional Neural Network (CNN). Song et al. [28] propose an intelligent transportation system to simulate the human mobility and transportation mode. The authors in [42] regard the crowd density on road network as pictures and then propose a deep spatio-temporal residual network to predict the crowd flows. Third, considering the graph structure of road network, some people try to apply graph neural networks to solve traffic prediction problems, such as travel demand prediction [35] and traffic flow prediction [26]. Fourth, the attention mechanism is also applied to capture complex spatio-temporal correlations among different features. For example, Yuan et al. [40] jointly predict travel demands and traffic flows by capturing different spatio-temporal correlations with the attention mechanism. Last but not least, Multi-Layer Perception (MLP) is also used to design learned models to replace spatio-temporal data structures. For example, Kraska et al. [21] replace the index structure with a learned recursive model. To further support data updating, many recent studies [7, 11, 12, 14, 38] leverage different methods to design adaptive learned index. However, they cannot be extended to replace multi-dimensional index which is more useful for spatio-temporal data management, so some work [6, 25, 27, 36] further study how to build learned index for multi-dimensional data.

9 CONCLUSIONS

We studied the ETA problem on a road network. We proposed a comprehensive and novel neural network based approach that is able to fully exploit spatio-temporal features extracted from four significant aspects: heterogeneity, proximity, periodicity and dynamicity. We built a link-connection graph to capture each route's static contexts (i.e., spatial proximity and temporal periodicity), and we collect historical traffic conditions as its dynamic contexts. We applied different attention mechanisms (i.e., sequence attention and graph attention) in a encoder-decoder framework to encode all spatio-temporal features and decode estimated travel time. Extensive experiments on real datasets verified effectiveness, efficiency and scalability of our model. In future, we will study how to transfer the learned ETA model from one city to another, and how to learn an effective model when being given few training data.

ACKNOWLEDGMENTS

The corresponding author is Guoliang Li. This work was supported by NSF of China (61925205, 61632016, 62072261), Huawei, and TAL education. Zhifeng Bao is supported in part by ARC DP220101434 and DP200102611.

REFERENCES

[1] 2021. GAIA. https://outreach.didichuxing.com/research/opendata/.

[2] 2021. OpenStreetMap. https://www.openstreetmap.org.

[3] Pouria Amirian, Anahid Basiri, and Jeremy Morley. 2017. Predictive analytics for enhancing travel time estimation in navigation apps of Apple, Google, and Microsoft. In *SIGSPATIAL*, Stephan Winter, Gautam S. Thakur, and Nicole Ronald (Eds.). 31–36.

[4] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR* abs/1607.06450 (2016).

[5] Shaojie Dai, Yanwei Yu, Hao Fan, and Junyu Dong. 2022. Spatio-Temporal Representation Learning with Social Tie for Personalized POI Recommendation. *Data Sci. Eng.* 7, 1 (2022), 44–56. https://doi.org/10.1007/s41019-022-00180-w

[6] Angjela Davitkova, Evica Milchevski, and Sebastian Michel. 2020. The ML-Index: A Multidimensional, Learned Index for Point, Range, and Nearest-Neighbor Queries. In *EDBT*, Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang (Eds.). 407–410.

[7] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David B. Lomet, and Tim Kraska. 2020. ALEX: An Updatable Adaptive Learned Index. In *SIGMOD*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). 969–984.

[8] Weishan Dong, Jian Li, Renjie Yao, Changsheng Li, Ting Yuan, and Lanjun Wang. 2016. Characterizing driving styles with deep learning. *CoRR* (2016).

[9] Ju Fan, Guoliang Li, Lizhu Zhou, Shanshan Chen, and Jun Hu. 2012. SEAL: Spatio-Textual Similarity Search. *Proc. VLDB Endow.* 5, 9 (2012), 824–835. https://doi.org/10.14778/2311906.2311910

[10] Xiaomin Fang, Jizhou Huang, Fan Wang, Lingke Zeng, Haijin Liang, and Haifeng Wang. 2020. ConSTGAT: Contextual Spatial-Temporal Graph Attention Network for Travel Time Estimation at Baidu Maps. In *KDD*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). 2697–2705.

[11] Paolo Ferragina and Giorgio Vinciguerra. 2020. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *VLDB* 13, 8 (2020), 1162–1175.

[12] Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. 2019. FITing-Tree: A Data-aware Index Structure. In *SIGMOD*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). 1189–1206.

[13] Qiang Gao, Fan Zhou, Kunpeng Zhang, Goce Trajcevski, Xucheng Luo, and Fengli Zhang. 2017. Identifying Human Mobility via Trajectory Embeddings.. In *IJCAI*. 1689–1695.

[14] Ali Hadian and Thomas Heinis. 2020. MADEX: Learning-augmented Algorithmic Index Structures. In *AIDB@VLDB 2020*, Bingsheng He, Berthold Reinwald, and Yingjun Wu (Eds.).

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity Mappings in Deep Residual Networks. In *ECCV*. 630–645.

[16] Zhixiang He, Chi-Yin Chow, and Jia-Dong Zhang. 2019. STANN: A Spatio-Temporal Attentive Neural Network for Traffic Prediction. *IEEE Access* 7 (2019), 4795–4806.

[17] Huiqi Hu, Guoliang Li, Zhifeng Bao, Jianhua Feng, Yongwei Wu, Zhiguo Gong, and Yaoqiang Xu. 2016. Top-k Spatio-Textual Similarity Join. *IEEE Trans. Knowl. Data Eng.* 28, 2 (2016), 551–565. https://doi.org/10.1109/TKDE.2015.2485213

[18] YuRui Huang, Jie Zhang, HengDa Bao, Yang Yang, and Jian Yang. 2021. Complementary Fusion of Deep Network and Tree Model for ETA Prediction. In *SIGSPATIAL*. 638–641.

[19] Ishan Jindal, Xuewen Chen, Matthew Nokleby, Jieping Ye, et al. 2017. A unified neural network approach for estimating travel time and distance for a taxi trip. *CoRR* (2017).

[20] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR(Poster)*.

[21] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). 489–504.

[22] Guoliang Li, Jianhua Feng, and Jing Xu. 2012. DESKS: Direction-Aware Spatial Keyword Search. In *ICDE*. 474–485. https://doi.org/10.1109/ICDE.2012.93

[23] Yaguang Li, Kun Fu, Zheng Wang, Cyrus Shahabi, Jieping Ye, and Yan Liu. 2018. Multi-task representation learning for travel time estimation. In *SIGKDD*. 1695–1704.

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

[25] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning Multi-Dimensional Indexes. In *SIGMOD*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). 985–1000.

[26] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban traffic prediction from spatio-temporal data using deep meta learning. In *SIGKDD*. 1720–1730.

[27] Varun Pandey, Alexander van Renen, Andreas Kipf, Jialin Ding, Ibrahim Sabek, and Alfons Kemper. 2020. The Case for Learned Spatial Indexes. In *AIDB@VLDB 2020*, Bingsheng He, Berthold Reinwald, and Yingjun Wu (Eds.).

[28] 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.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*. 5998–6008.

[30] Dong Wang, Junbo Zhang, Wei Cao, Jian Li, and Yu Zheng. 2018. When Will You Arrive? Estimating Travel Time Based on Deep Neural Networks. In *AAAI*, Sheila A. McIlraith and Kilian Q. Weinberger (Eds.). 2500–2507.

[31] Hongjian Wang, Xianfeng Tang, Yu-Hsuan Kuo, Daniel Kifer, and Zhenhui Li. 2019. A Simple Baseline for Travel Time Estimation using Large-scale Trip Data. *TITS* 10, 2 (2019), 19:1–19:22.

[32] Yilun Wang, Yu Zheng, and Yexiang Xue. 2014. Travel time estimation of a path using sparse trajectories. In *SIGKDD*, Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani (Eds.). 25–34.

[33] Zheng Wang, Kun Fu, and Jieping Ye. 2018. Learning to Estimate the Travel Time. In *SIGKDD*, Yike Guo and Faisal Farooq (Eds.). 858–866.

[34] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. 2017. Modeling Trajectories with Recurrent Neural Networks. In *IJCAI*. 3083–3090.

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

[36] Zongheng Yang, Badrish Chandramouli, Chi Wang, Johannes Gehrke, Yinan Li, Umar Farooq Minhas, Per-Åke Larson, Donald Kossmann, and Rajeev Acharya. 2020. Qd-tree: Learning Data Layouts for Big Data Analytics. In *SIGMOD*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). 193–208.

[37] Haitao Yuan and Guoliang Li. 2019. Distributed in-memory trajectory similarity search and join on road network. In *ICDE*. 1262–1273.

[38] 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.

[39] 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.

[40] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2021. An Effective Joint Prediction Model for Travel Demands and Traffic Flows. In *ICDE*.

[41] Hanyuan Zhang, Hao Wu, Weiwei Sun, and Baihua Zheng. 2018. DeepTravel: a Neural Network Based Travel Time Estimation Model with Auxiliary Supervision. In *IJCAI*, Jérôme Lang (Ed.). ijcai.org, 3655–3661.

[42] Junbo Zhang, Yu Zheng, Dekang Qi, Ruiyuan Li, and Xiuwen Yi. 2016. DNN-based prediction model for spatio-temporal data. In *SIGSPATIAL*. 1–4.