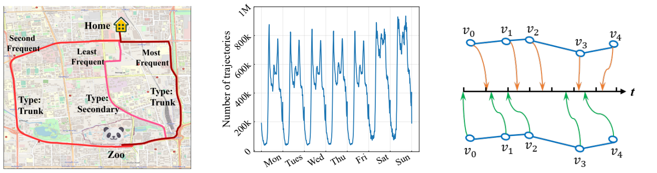
Self-supervised Trajectory Representation Learning with Temporal Regularities and Travel Semantics

Jiawei Jiang[1,2], Dayan Pan[1,2], Houxing Ren[1,2], Xiaohan Jiang[1], Chao Li[1,2], Jingyuan Wang[1,3,4,*]
[1]School of Computer Science and Engineering, Beihang University, Beijing, China
[2]Z-park Strategic Alliance of Smart City Industrial Technology Innovation, Beijing, China
[3]Pengcheng Laboratory, Shenzhen, China
[4]School of Economics and Management, Beihang University, Beijing, China
{jwjiang, dayan, renhouxing, jxh199, licc, jywang}@buaa.edu.cn

***Abstract*—Trajectory Representation Learning (TRL) is a powerful tool for spatial-temporal data analysis and management. TRL aims to convert complicated raw trajectories into low-dimensional representation vectors, which can be applied to various downstream tasks, such as trajectory classification, clustering, and similarity computation. Existing TRL works usually treat trajectories as ordinary sequence data, while some important spatial-temporal characteristics, such as temporal regularities and travel semantics, are not fully exploited. To fill this gap, we propose a novel Self-supervised trajectory representation learning framework with TemporAl Regularities and Travel semantics, namely START. The proposed method consists of two stages. The first stage is a Trajectory Pattern-Enhanced Graph Attention Network (TPE-GAT), which converts the road network features and travel semantics into representation vectors of road segments. The second stage is a Time-Aware Trajectory Encoder (TAT-Enc), which encodes representation vectors of road segments in the same trajectory as a trajectory representation vector, meanwhile incorporating temporal regularities with the trajectory representation. Moreover, we also design two self-supervised tasks, *i.e.,* span-masked trajectory recovery and trajectory contrastive learning, to introduce spatial-temporal characteristics of trajectories into the training process of our START framework. The effectiveness of the proposed method is verified by extensive experiments on two large-scale real-world datasets for three downstream tasks. The experiments also demonstrate that our method can be transferred across different cities to adapt heterogeneous trajectory datasets.**

I. Introduction

With the rapid development of GPS-enabled devices, a large amount of trajectory data can be collected in cities. Trajectory data analysis and management, such as trajectory-based prediction [33], [42], traffic prediction [37], [39], urban dangerous goods management [38], and trajectory similarity computation [34], have become a hot topic in the data engineering community. Traditional research on trajectory data analysis requires manual feature engineering and unique models for specific tasks, making them difficult to transfer to different applications [7]. To improve the generality of tools for analyzing trajectory data analysis tools, *Trajectory Representation Learning (TRL)* has emerged in recent years [8], [9]. TRL aims to transform raw trajectories into generic low-dimensional representation vectors that can be applied in various downstream tasks rather than being limited to a specific task.



(a) Trajectory Frequencies. (b) Periodic Patterns of Urban Traffic. (c) Time Interval Distribution.

Fig. 1. Temporal Regularities and Travel Semantics in Trajectories. (Map data © OpenStreetMap contributors, CC BY-SA.)

In the literature, earlier TRL studies directly use general sequence-to-sequence models (such as LSTM [35] and Transformers [11]) with reconstruction tasks to generate trajectory representation vectors [7]–[9]. Such models consider trajectories as ordinary sequence data and thus cannot fully capture spatial-temporal semantic information of trajectories in the representation vectors. After this, many trajectory representation learning methods are proposed for specific downstream tasks, such as for approximate trajectory similarity computation [19], [20], trajectory clustering [18], anomalous trajectory detection [32] and path ranking [36].

In recent years, some two-stage methods have been proposed to learn generic trajectory representations for multiple downstream tasks [5], [6]. These methods first adopt a graph representation learning to convert road segments of a static road network into representation vectors and then use sequential deep learning models with self-supervised tasks to convert the road representation vectors in the same trajectory into a trajectory representation vector. For example, Toast [5] and PIM [6] use node2vec [17] to learn road representations and respectively use Transformer with masked prediction and RNN with mutual information maximization as self-supervised tasks to generate generic trajectory representations. These two-stage methods incorporate the static road network as spatial semantic information in the trajectory representations so they can improve downstream tasks. However, trajectory data contains rather complicated spatial-temporal semantic information. Many critical spatial-temporal characteristics and semantic information are helpful for downstream tasks but are still not fully utilized by existing works.

[*] Corresponding Author: Jingyuan Wang

The first characteristic that should be considered in TRL is travel semantics. As shown in Figure 1(a), road segments traversed by different trajectories with the same origin and destination (OD) have different road types and visit frequencies, *i.e.,* the human mobility patterns. Both of these travel-related semantics are useful for downstream tasks and should be incorporated into trajectory representations. However, previous works such as Toast [5] and PIM [6] only model the static road network information in their representation learning but fail to incorporate the travel semantic information. The second characteristic that should be considered is temporal regularities. From a macro perspective, the trajectories generated by vehicles in the city are influenced by the periodic temporal patterns of urban traffic. As shown in Figure 1(b), the number of urban trajectories exhibits an apparent periodic pattern, *i.e.,* the number of trajectories during the morning and evening rush hours is much larger than usual. A large number of trajectories means congested road conditions, which naturally affects the generation of trajectories. From a micro perspective, irregular time intervals are another temporal regularity of trajectories. As shown in Figure 1(c), for two trajectories with the same shape, the sample points, *i.e.,* road segments, can be distributed quite differently on the time axis. It is because the travel time of a road segment is dynamic, which can also reflect the congestion level of roads. Both temporal regularities of periodic patterns and irregular time intervals are useful for downstream tasks. However, most previous works consider trajectories only as sequences of locations [5], [6], [8] and do not consider the temporal information in their methods. In addition, the self-supervised tasks in existing TRL methods do not sufficiently consider the spatial-temporal characteristics of the trajectories. Most methods use general sequence reconstruction [8], [9] or masked prediction tasks (MLM) [5] as their self-supervised tasks, which treat trajectories as general sequence data and fail to capture temporal and travel semantics. This problem limits the performance of the learned representation vectors on downstream tasks.

In this paper, we propose a novel **S**elf-supervised trajectory representation learning framework with **T**empor**A**l **R**egularities and **T**ravel semantics, abbreviated as START. The framework integrates temporal regularities and travel semantics into TRL using a two-stage learning method. The first stage is a Trajectory Pattern-Enhanced Graph Attention Network (TPE-GAT), which converts a road network into road segment representation vectors. Travel semantics information is also incorporated at this stage. Specifically, the TPE-GAT module takes rich road features as input and extends Graph Attention Network [4] using a road segment transfer probability matrix to model road visit frequencies. In this way, both the travel semantics of road features and visit frequencies are integrated into the road representations. The second stage converts road representation sequences into trajectory representations and incorporates temporal regularity information. Here we propose a Time-Aware Trajectory Encoder Layer (TAT-Enc) to incorporate temporal regularities. Specifically, the TAT-Enc fuses minute and day-of-week indexes with road segment representations to capture the periodic temporal patterns of urban traffic and adopts a Time Interval-Aware Self-Attention to process irregular time interval information.

We also design two self-supervised tasks to train our START. The first is *span-masked trajectory recovery*, which masks consecutive subsequences in trajectories to capture the local features and order information. The second is *trajectory contrastive learning*, which employs four data augmentation methods that consider the spatial-temporal characteristics of trajectories to train the contrastive learning loss. Compared to the traditional self-supervised tasks such as sequence reconstruction and MLM, the proposed tasks fully consider the spatial-temporal characteristics of the trajectories. The effectiveness of the proposed method is verified by extensive experiments on two large-scale datasets for three downstream tasks. The results show that START significantly outperforms the state-of-the-art models.

In summary, the main contributions of this paper are summarized as follows:

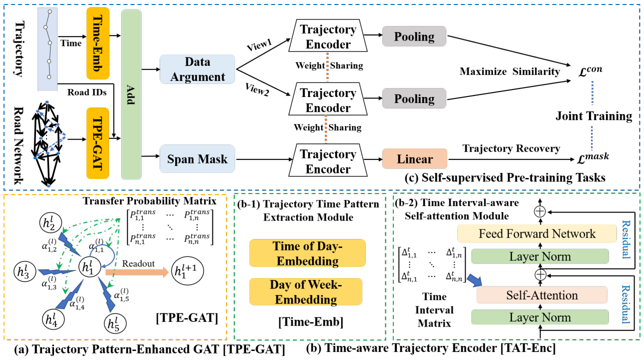
- We propose a two-stage TRL method incorporating temporal regularities and travel semantics into trajectory representations. Compared to previous TRL research, the proposed method can utilize more spatial-temporal characteristics of trajectories for downstream tasks.
- We design two self-supervised tasks to train our START. Compared to traditional self-supervised tasks for general sequence representation learning, such as sequence reconstruction and MLM, the proposed tasks are more suitable for TRL since they account for the spatial-temporal characteristics of trajectories. We believe these tasks can be applied to other TRL model training.
- In addition to superior performance, the experiments also demonstrate that the proposed self-supervised tasks can use fewer data to outperform the supervised model. Moreover, our methods can be transferred across heterogeneous road network datasets. To the best of our knowledge, this is the first TRL method with this feature, which is very useful for solving the problem of insufficient data in many real-world applications.

II. PRELIMINARIES

In this section, we first introduce basic notations and preliminaries used in this paper. Then we formalize the problem of trajectory representation learning.

A. Notations and Definitions

Definition 1 (Road Network). *We represent road network as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{F}_{\mathcal{V}}, \boldsymbol{A})$, where $\mathcal{V} = \{v_1, \cdots, v_{|\mathcal{V}|}\}$ is a set of $|\mathcal{V}|$ vertices, each vertex v_i representing a road segment, $\mathcal{N}_i$ is the neighborhood of road segment v_i, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, each $e_{i,j} = (v_i, v_j)$ representing the intersection between road segments v_i and v_j, $\boldsymbol{F}_{\mathcal{V}} \in \mathbb{R}^{|\mathcal{V}| \times d_{in}}$ is the features of road segments, and $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ is a binary value adjacency matrix of network $\mathcal{G}$ indicating whether there exists a directed link between roads.*



$$\begin{bmatrix} P_{1,1}^{trans} & \cdots & P_{1,n}^{trans} \\ \vdots & \ddots & \vdots \\ P_{n,1}^{trans} & \cdots & P_{n,n}^{trans} \end{bmatrix} \quad \begin{bmatrix} \Delta_{1,1}^{t} & \cdots & \Delta_{1,n}^{t} \\ \vdots & \ddots & \vdots \\ \Delta_{n,1}^{t} & \cdots & \Delta_{n,n}^{t} \end{bmatrix}$$

Fig. 2. Overall Framework of START.

Definition 2 (GPS-based Trajectory)**.** *A GPS-based trajectory (or a raw trajectory) $\mathcal{T}^{raw}$ is a sequence of spatial-temporal sample points recorded by GPS-enabled devices, a sample point $sp = \langle lat_i, lon_i, t_i \rangle$ is a triplet consisting of latitude, longitude, and a visit timestamp.*

Definition 3 (Road-network Constrained Trajectory)**.** *A road-network constrained trajectory $\mathcal{T}$ is a time-ordered sequence of m adjacent road segments generated by a user, i.e., $\mathcal{T} = [\langle v_i, t_i \rangle]_{i=1}^m$, where $v_i \in \mathcal{V}$ presents the i-th road segment and t_i is the visit timestamp for v_i. For simplicity, we also use roads to refer to road segments in the following.*

In this study, we mainly focus on the road-network constrained trajectories. Therefore, given a raw trajectory $\mathcal{T}^{raw}$ and the road network $\mathcal{G}$, we perform the *map matching* [3] procedure to align trajectory points with road segments and get the road-network constrained trajectory $\mathcal{T}$.

B. Problem Statement

Given a trajectory dataset $\mathcal{D} = \{\mathcal{T}_i\}_{i=1}^{|\mathcal{D}|}$ and a road network $\mathcal{G}$, the *Trajectory Representation Learning* (TRL) task aims to learn a generic low-dimensional representation $\boldsymbol{p}_i \in \mathbb{R}^d$ for each trajectory $\mathcal{T}_i \in \mathcal{D}$. Specifically, in this study, we aim to develop a self-supervised framework that encodes each trajectory $\mathcal{T}_i$ into a generic d-dimensional representation vector $\boldsymbol{p}_i$, which can be applied in various downstream tasks, such as travel time estimation, trajectory classification, and trajectory similarity computation.

III. Method

In this section, we introduce the proposed START framework. Figure 2 provides an overview of it. We start with the framework structure, including a trajectory pattern-enhanced graph attention layer (TPE-GAT) and a time-aware trajectory encoder layer (TAT-Enc). Then, we present two self-supervised tasks to train START. Finally, we display how to adapt the learned representations to specific downstream tasks.

A. Trajectory Pattern-Enhanced Graph Attention Layer

The Trajectory Pattern-Enhanced Graph Attention Network (TPE-GAT) is the first stage of START, which converts a road network into road representation vectors and incorporates the travel semantics of the trajectories. As mentioned in Section II-A, the roads in the trajectory have some important inherent properties, and they are constrained by the connectivity of the road network. Therefore, we learn the road-level representation vector from both the road features and network structure. Previous works often use random walk-based models such as node2vec [17] to encode the static road network as spatial semantic information used in the trajectory representations [5], [8]. However, such learning methods fail to incorporate road features and travel semantics in the trajectories, such as visit frequencies.

Therefore, we propose using graph neural networks to capture both the road features and network structure. Considering that the road network is a directed graph, we choose the graph attention network (GAT) [4] because it can dynamically assign weights to the neighborhood nodes by computing the attention weights between pairs of nodes. However, the standard GAT cannot capture the travel patterns in the trajectories. To solve this problem, we propose a Trajectory Pattern-Enhanced Graph Attention Network, namely TPE-GAT, which extends the computation of attention weights of GAT by introducing the transfer probability matrix between roads computed from the historical data to model visiting frequencies of roads.

The TPE-GAT consists of L_1 layers in total. First, we take rich road features $\boldsymbol{F}_{\mathcal{V}}$ as input to the first layer. Specifically, given a road v_i, we consider six types of features, namely road type, road length, number of lanes, maximum travel speed, in-degrees, and out-degrees in the road network. We concatenate these features to create the initialized road representation $\boldsymbol{h}_i^{(0)} \in \mathbb{R}^{d_0}$ for the road v_i. Then, the attention weight α_{ij} between road v_i and v_j in the l-th layer are computed as (l is ignored here for simplicity):

$$\begin{aligned} e_{ij} &= (\boldsymbol{h}_i \boldsymbol{W}_1 + \boldsymbol{h}_j \boldsymbol{W}_2 + p_{ij}^{trans} \boldsymbol{W}_3) \boldsymbol{W}_4^T, \\ \alpha_{ij} &= \frac{\exp(\mathrm{LeakyReLU}(e_{ij}))}{\sum_{k \in \mathcal{N}_i} \exp(\mathrm{LeakyReLU}(e_{ik}))}, \end{aligned} \tag{1}$$

where $\boldsymbol{h}_i, \boldsymbol{h}_j \in \mathbb{R}^{d_l}$ are road representations of v_i and v_j, $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{d_l \times d_{l+1}}, \boldsymbol{W}_3, \boldsymbol{W}_4 \in \mathbb{R}^{1 \times d_{l+1}}$ are learnable parameters, LeakyReLU is the activation function whose negative input slope is 0.2 [4], and p_{ij}^{trans} is the transfer probability between v_i and v_j, which can be calculated as:

$$p_{ij}^{trans} = \mathrm{count}(v_i \to v_j) / \mathrm{count}(v_i), \tag{2}$$

where $\mathrm{count}(v_i \to v_j)$ and $\mathrm{count}(v_i)$ is the frequency of edges (v_i, v_j) and road v_i appeared in the trajectory dataset $\mathcal{D}$, respectively.

Then we obtain the output feature $\tilde{\boldsymbol{h}}_i$ of the i-th road v_i through combining the features of its neighborhoods using the attention weights as:

$$\tilde{\boldsymbol{h}}_i^{(l+1)} = \mathrm{ELU}\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \boldsymbol{h}_j^{(l)} \boldsymbol{W}_5 \right), \tag{3}$$

where $\boldsymbol{W}_5$ are the learnable parameters and ELU is the Exponential Linear Unit activation function [4].

We use multi-head attention to stabilize the learning process and incorporate various types of information. Specifically, H_1

denotes the number of independent attention mechanisms that are computed as Equations (1) and (3), then we concatenate the outputs of these independent attention mechanisms as:

$$\boldsymbol{h}_i^{(l+1)} = \overset{H_1}{\underset{k=1}{||}} \mathrm{ELU}\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^{(k)} \boldsymbol{h}_j^{(l)} \boldsymbol{W}_5^{(k)}\right), \tag{4}$$

where $||$ represents concatenation, $\alpha_{ij}^{(k)}$ are the attention scores computed by the k-th attention head, $\boldsymbol{W}_5^{(k)}$ is the weight matrix of the corresponding linear transformation in layer l.

The TPE-GAT layer considers the connectivity between roads due to both static road network structure and human mobility. The output of the last layer is defined as $\boldsymbol{r}_i \in \mathbb{R}^d$ and represents the representation of the road v_i, which contains road network contextual information and trajectory travel semantics. Moreover, the TPE-GAT layer is trained together with the trajectory encoder layer described below. We use sparse matrix operations following [4] to enable the model for large-scale road networks.

B. Time-Aware Trajectory Encoder Layer

After obtaining the road representations from the TPE-GAT layer, we need to convert road representation sequences into trajectory representations and incorporate temporal regularity information in the second stage. To model the co-occurrence relationship between roads in the trajectory, we use the Transformer encoder [11] because it can capture the contextual information of the trajectory from the left and right sides of the road to realize the full interaction between roads. In addition, we extend the Transformer encoder and propose a Time-Aware Trajectory Encoder Layer (TAT-Enc) to incorporate temporal regularities in urban trajectories, which consist of two modules. The first is a Trajectory Time Pattern Extraction module that uses two temporal embeddings to capture the periodic patterns of urban traffic. The second is a Time Interval-Aware Self-Attention module to explicitly model the irregular time intervals between roads in the trajectory.

1) Trajectory Time Pattern Extraction Module: To capture the cyclical patterns of urban traffic, we use two temporal embedding vectors to extract the periodicity of weeks and days, respectively. For each visit timestamp t_i of the road v_i, we use embedding vectors $\boldsymbol{t}_{mi(t_i)} \in \mathbb{R}^d$ and $\boldsymbol{t}_{di(t_i)} \in \mathbb{R}^d$ to embed the two periodic patterns, where $mi(t_i)$ and $di(t_i)$ are functions of transforming t_i into its minutes index (1 to 1440) and day-of-week index (1 to 7).

Then we obtain the fused embeddings $\boldsymbol{x}_i \in \mathbb{R}^d$ of road v_i by summing several representations as follows:

$$\boldsymbol{x}_i = \boldsymbol{r}_i + \boldsymbol{t}_{mi(t_i)} + \boldsymbol{t}_{di(t_i)} + \boldsymbol{pe}_i, \tag{5}$$

where $\boldsymbol{r}_i$ denotes the road representations, $\boldsymbol{t}_{mi(t_i)}$ and $\boldsymbol{t}_{di(t_i)}$ are corresponding temporal representations, and $\boldsymbol{pe}_i$ denotes the position encoding used in Transformer to introduce position information of the input trajectory. Finally, the initial representation of the trajectory $\mathcal{T}$ is obtained by concatenating the embeddings of roads in it as $\boldsymbol{X} = \boldsymbol{x}_1 \| \dots \| \boldsymbol{x}_{|\mathcal{T}|} \in \mathbb{R}^{|\mathcal{T}|\times d}$.

2) Time Interval-aware Self-attention Module: In the standard multi-head self-attention of Transformer encoder, given the input trajectory representation $\boldsymbol{X}$, the H_2 attention heads transform $\boldsymbol{X}$ into the H_2 query matrixes $\boldsymbol{Q}_h = \boldsymbol{X}\boldsymbol{W}_h^Q$, key matrixes $\boldsymbol{K}_h = \boldsymbol{X}\boldsymbol{W}_h^K$, and value matrixes $\boldsymbol{V}_h = \boldsymbol{X}\boldsymbol{W}_h^V$ synchronously, where $\boldsymbol{W}_h^Q, \boldsymbol{W}_h^K, \boldsymbol{W}_h^V \in \mathbb{R}^{d\times d'}$ are learnable parameters and $d' = d/H_2$. Then the self-attention of the h-th attention head is calculated as:

$$A_h(\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}_h \boldsymbol{K}_h^T}{\sqrt{d'}}\right)\boldsymbol{V}_h. \tag{6}$$

To consider the irregular time intervals between road segments, which can reflect the congestion level of the road, we propose a *Time Interval-Aware Self-Attention* to replace the standard self-attention of the Transformer encoder as:

$$TA_h(\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h) = \mathrm{softmax}\left(\frac{\boldsymbol{Q}_h \boldsymbol{K}_h^T}{\sqrt{d'}} + \tilde{\boldsymbol{\Delta}}\right)\boldsymbol{V}_h, \tag{7}$$

where $\tilde{\boldsymbol{\Delta}} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{T}|}$ is an adaptive time interval matrix and each element in it measures the impact among road segments in a trajectory. Given two roads v_i and v_j, the element $\delta_{ij} \in \tilde{\boldsymbol{\Delta}}$ should have a large value when the time interval between v_i and v_j is short, *i.e.,* the two roads have strong impacts in the self-attention, vice versa. In this way, the irregular time intervals could be incorporated into the Transformer encoder.

The calculation process of $\tilde{\boldsymbol{\Delta}}$ is as follows. Given the timestamp t_i of the road v_i, we calculate the relative time interval $\delta_{i,j} = |t_i - t_j|$ for any two roads to obtain the original time interval matrix $\boldsymbol{\Delta}$ as:

$$\boldsymbol{\Delta} = \begin{bmatrix} \delta_{1,1} & \delta_{1,2} & \cdots & \delta_{1,|\mathcal{T}|} \\ \delta_{2,1} & \delta_{2,2} & \cdots & \delta_{2,|\mathcal{T}|} \\ \cdots & \cdots & \cdots & \cdots \\ \delta_{|\mathcal{T}|,1} & \delta_{|\mathcal{T}|,2} & \cdots & \delta_{|\mathcal{T}|,|\mathcal{T}|} \end{bmatrix}. \tag{8}$$

In the matrix $\boldsymbol{\Delta}$, the shorter the time interval between v_i and v_j, the smaller the value of $\delta_{i,j}$. Since the impact between roads should become smaller with the time interval increasing, *i.e.,* the greater the time interval, the smaller the impact, we introduce a decay function to process the raw value in $\boldsymbol{\Delta}$. Specifically, we set $\delta'_{i,j} = 1/\log(\mathrm{e} + \delta_{i,j})$, where $e \approx 2.718$. In this way, $\delta'_{i,j}$ decreases with increasing time intervals.

Furthermore, we adopt a two-linear-transformation to process $\delta'_{i,j}$ as:

$$\tilde{\delta}_{i,j} = (\mathrm{LeakyReLU}(\delta'_{i,j}\,\boldsymbol{\omega}_1))\,\boldsymbol{\omega}_2^T, \tag{9}$$

where $\boldsymbol{\omega}_1, \boldsymbol{\omega}_2$ are learnable parameters and $\mathrm{LeakyReLU}$ is a activation function whose negative input slope is 0.2. By this method, $\tilde{\delta}_{i,j}$ becomes learnable and can capture the irregular time interval information. Finally, we plug $\tilde{\delta}_{i,j}$ into Eq. (7) to get the Time Interval-Aware Self-Attention.

Then we concatenate the output of the H_2 attention heads and project it through $\boldsymbol{W}^O \in \mathbb{R}^{d\times d}$ to obtain the outputs $\boldsymbol{X}' \in \mathbb{R}^{|\mathcal{T}|\times d}$ as:

$$\boldsymbol{X}' = \mathrm{MultiAtt}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = (TA_1 \| \dots \| TA_{H_2})\boldsymbol{W}^O. \tag{10}$$

After the multi-head attention, we employ layer normalization and residual connection following Transformer [11].

Finally, a position-wise feed-forward network (noted as FFN) consists of two layers of linear transformations, and ReLU activation is used to get the output representation $\boldsymbol{Z}$ of trajectory $\mathcal{T}$ as:

$$\boldsymbol{Z} = (\mathrm{ReLU}(\boldsymbol{X}'\boldsymbol{W}_F^1 + \boldsymbol{b}_F^1))\boldsymbol{W}_F^2 + \boldsymbol{b}_F^2, \tag{11}$$

where $\boldsymbol{W}_F^1, \boldsymbol{W}_F^2 \in \mathbb{R}^{d \times d}, \boldsymbol{b}_F^2, \boldsymbol{b}_F^2 \in \mathbb{R}^d$ are learnable parameters and ReLU is the activation function. The layer normalization and residual connection are also used here.

3) Trajectory Representation Pooling: After stacking L_2 layers of the self-attention module, we obtain the final output representation $\boldsymbol{Z} \in \mathbb{R}^{|\mathcal{T}| \times d}$, which has been fully interacted between the road segments. Furthermore, following [10], we extract the whole trajectory representation $\boldsymbol{p}_i \in \mathbb{R}^d$ by inserting a placeholder in the first position throughout training tasks and take it as the trajectory representation.

C. Self-supervised Pre-training Tasks

This work aims to learn trajectory representations self-supervised to support multiple downstream tasks. Therefore, considering the spatial-temporal characteristics of the trajectories, we design two self-supervised tasks which do not target specific downstream tasks to learn generic representations.

1) Span-Masked Trajectory Recovery: Masked language modeling (MLM) has proven its superiority in learning sequence data representations in many studies [5], [10]. Each word in the sequence is masked independently with a probability in the previous MLM task, and the model is used to predict the masked words. However, this task is not fully applicable to our task because the trajectory is a sequence of *adjacent* roads. If we mask the road independently, the model can easily infer the masked road based on its upstream and downstream roads in the road network. Therefore, we propose the span-masked method, where we select several consecutive subsequences of length l_m in the trajectory for masking, whose total length is p_m percent of the trajectory length. When masking the trajectory, we replace the selected road v_i with a special token [MASK] and set the corresponding minute index $mi(t_i)$ and day-of-week index $di(t_i)$ to a special token [MASKT]. After obtaining the representation $\boldsymbol{Z}$ of the masked trajectory $\mathcal{T}$, we use a linear layer with parameters $\boldsymbol{W}_m \in \mathbb{R}^{d \times |\mathcal{V}|}, \boldsymbol{b}_m \in \mathbb{R}^{|\mathcal{V}|}$ to predict the masked roads as:

$$\hat{\boldsymbol{Z}} = \boldsymbol{Z}\boldsymbol{W}_m + \boldsymbol{b}_m \in \mathbb{R}^{|\mathcal{T}| \times |\mathcal{V}|}, \tag{12}$$

Then we use the cross-entropy loss between masked roads and predicted values as the optimized target:

$$\boldsymbol{\mathcal{L}}_{\mathcal{T}}^{mask} = -\frac{1}{|\mathcal{M}|} \sum_{v_i \in \mathcal{M}} \log \frac{\exp(\hat{\boldsymbol{Z}}_{v_i})}{\sum_{v_j \in \mathcal{V}} \exp(\hat{\boldsymbol{Z}}_{v_j})}, \tag{13}$$

where $\mathcal{M}$ is the set of masked roads. We average all losses of N_b trajectories in a mini-batch to obtain the loss $\boldsymbol{\mathcal{L}}^{mask}$.

2) Trajectory Contrastive Learning: Mask prediction focuses on capturing co-occurrence relationships between roads and contextual information of the road network. To improve the modeling of the spatial-temporal characteristics and travel semantics, we introduce a contrastive learning method.

Trajectory Data Augmentation Strategies. Contrastive learning aims to learn representations to bring semantically similar positive samples closer and make negative samples farther apart. Thus, the crucial question is how to construct different views in contrastive learning. Considering the spatial-temporal characteristics of the trajectories, we explore four data augmentation strategies to generate views for contrastive learning.

- *Trajectory Trimming*: We obtain the enhanced trajectory by randomly removing a continuous subsequence from the trajectory. In order not to destroy the continuity and travel semantics of the trajectory, we trim only at the origin or destination of the trajectory, and the trimming ratio r_1 is a random sample of $0.05 - 0.15$. This data augmentation method is applied since the semantics of trajectories with close origins or destinations are similar.
- *Temporal Shifting*: Influenced by the urban traffic patterns, the road travel time is dynamic. Given a trajectory, we randomly select a subset of roads (scale $r_2 = 0.15$) and perform a random perturbation by $t_{aug} = t_{cur} - (t_{cur} - t_{his}) * r_3$, where r_3 is a random sample of $0.15 - 0.30$, t_{cur} and t_{his} are the current and historical average travel time of that road, respectively. Using this augmentation method helps to capture the travel semantics of the trajectory in the temporal dimension.
- *Road Segments Mask*: In the span-masked trajectory recovery task, some roads and the corresponding timestamps of the trajectory are randomly selected and masked. The masked trajectory can be considered as the trajectory with missing values to learn the travel semantics of the trajectories in both temporal and spatial dimensions.
- *Dropout*: Dropout is a widely used method to avoid overfitting. Here we use it as a data augmentation method to randomly drop some tokens with a certain probability from the data embedding layer and set them to zero [21].

Contrastive Trajectory Learning. Following [15], we adopt the normalized temperature-scaled cross-entropy loss with in-batch negatives as the contrastive objective. We randomly select N_b trajectories from the dataset $\mathcal{D}$ and then obtain $2N_b$ trajectories after data augmentation. Each trajectory (also called the anchor) is trained to find out the corresponding data-augmentation trajectory (the positive sample) among $2(N_b-1)$ negative samples in the batch. Formally, the contrastive training objective for a positive pair (i, j) is defined as:

$$\boldsymbol{\mathcal{L}}_{i,j}^{con} = -\log \frac{\exp(\mathrm{sim}(\boldsymbol{p}_i, \boldsymbol{p}_j)/\tau)}{\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i]} \exp(\mathrm{sim}(\boldsymbol{p}_i, \boldsymbol{p}_k)/\tau)}, \tag{14}$$

where τ is the temperature hyperparameter, $\mathrm{sim}(\boldsymbol{p}_i, \boldsymbol{p}_j)$ is the cosine similarity $\frac{\boldsymbol{p}_i \cdot \boldsymbol{p}_j}{\|\boldsymbol{p}_i\| \|\boldsymbol{p}_j\|}$ between $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ ($\cdot$ is the inner product operation), $\mathbf{1}$ is the indicator equal to one if the condition is satisfied, otherwise it is zero. We average all $2N_b$ in-batch losses to obtain the contrastive loss $\boldsymbol{\mathcal{L}}^{con}$.

We pre-train the proposed START with the two self-supervised tasks above. The pre-training loss is defined as:

$$\boldsymbol{\mathcal{L}}^{pre} = \lambda \boldsymbol{\mathcal{L}}^{mask} + (1 - \lambda)\boldsymbol{\mathcal{L}}^{con}, \tag{15}$$

where λ is the hyperparameter to balance the two tasks.

D. Model Fine-tuning and Downstream Tasks

In this section, we aim to adapt the learned representations to specific downstream tasks, either directly or with the necessary fine-tuning.

1) Trajectory Travel Time Estimation: This task aims to estimate the travel time from the origin to the destination with a given road sequence and the departure time. We build a regression model using a single fully connected layer to obtain the predicted value as $\hat{y}_i = FC(\boldsymbol{p}_i)$. Then we use the mean square error (MSE) as the optimization objective:

$$\mathcal{L}^{regress} = \frac{1}{N}\sum_{i=1}^{N} \|y_i - \hat{y}_i\|^2, \tag{16}$$

where y_i is the ground truth and N is the total number of trajectories in the test dataset.

2) Trajectory Classification: This task aims to classify trajectories based on a specific label, such as carrying passengers or not, the driver ID, the transportation, etc. We employ a simple fully connected layer with the $\mathrm{softmax}$ activation to obtain the predicted value as $\hat{\boldsymbol{y}}_i = \mathrm{softmax}(FC(\boldsymbol{p}_i))$. Then we optimize the model with the cross-entropy loss:

$$\mathcal{L}^{classify} = \frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} -\boldsymbol{y}_i(c)\log(\hat{\boldsymbol{y}}_i(c)), \tag{17}$$

where $\boldsymbol{y}_i$ is the ground truth, N is the total number of trajectories in the test dataset, and C is the number of categories.

3) Trajectory Similarity Computation and Search: In this task, we design two sub-tasks: the most similar trajectory search and the k-nearest trajectory search. Here we directly use the representation $\boldsymbol{p}_i$ obtained from the pre-training task without fine-tuning. The most similar trajectory search task is to find out the most similar trajectory from a large database given a query. In the k-nearest trajectory search task, given a trajectory, models need to find top-k similar trajectories from candidates ignoring the rank. The detailed settings for these two tasks can be found in Section IV-D4.

IV. Experiments

In this section, we conduct extensive experiments to evaluate the performance of the START framework. The experiments include five parts:

- *Performance Comparison*. We compare the performance of START with eight baselines on two large-scale datasets for three downstream tasks. The experiment results show the superior overall performance of START.
- *Pre-training Effect Study*. We demonstrate the effectiveness of the self-supervised pre-training tasks over small-size datasets and across datasets. The results show that the self-supervised tasks can effectively reduce the usage of training data, and the model can be transferred across heterogeneous datasets. This nature is beneficial for solving the problem of insufficient training data.
- *Ablation Experiment*. We use ablation studies to verify the effectiveness of each sub-module of START.
- *Parameter Sensitivity Experiment*. This experiment verifies the stability of our method over key parameters.
- *Efficiency and Scalability Study*. This experiment clarifies that our proposed framework is efficient and can scale for large datasets.

TABLE I
Statistics of the two datasets after preprocessing.

Dataset	BJ	Porto
Time span	2015/11/01-2015/11/30	2013/07/01-2014/07/01
#Trajectory	1018312	695085
#Usr	1677	435
#Road Segment	38479	10903
train/eval/test	656221/174478/187613	417040/139020/139025

A. Datasets and Preprocessing

We use two real-world, large-scale trajectory datasets in the experiments, *i.e.,* BJ and Porto. BJ was collected by taxis in Beijing in November 2015. Porto is an open-source dataset released for a Kaggle competition[1] and is sampled every 15 seconds. We download map data of Beijing and Porto from OpenStreetMap (OSM) [1] to construct the directed graph (road network) $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{F}_{\mathcal{V}}, \boldsymbol{A})$ defined in Definition 1. The OSM data contains three parts: i) All road segments in the cities. We use these roads to form the vertex set $\mathcal{V}$ of $\mathcal{G}$, where each road represents a vertex. ii) The connection relationships between all roads. If two roads have a connection relation, we define that the corresponding vertexes have an edge. We use these connection relationships to form the binary value adjacency matrix $\boldsymbol{A}$ and the edge set $\mathcal{E}$ of $\mathcal{G}$. iii) The features of roads. We select four important road features, *i.e.,* road type, length, number of lanes, and maximum travel speed, and calculate the in-degree and out-degree of each road in the adjacency matrix $\boldsymbol{A}$ to construct the road features $\boldsymbol{F}_{\mathcal{V}}$. Finally, we use the directed graph $\mathcal{G}$ of Beijing and Porto as the input of our proposed START. Furthermore, we perform *map matching* [3] to obtain the road-network constrained trajectories. Details of the two datasets are given in Table I.

We ignore the roads that are not covered by the trajectories. Besides, we also remove loop trajectories, trajectories with lengths less than six, and users with less than 20 trajectories and set the maximum trajectory length to 128. We split BJ into the training, validation, and test datasets in chronological order. The three datasets cover 18/5/7 days because there is less data on November 25. For Porto, we split each month's data in chronological order with a ratio of 6:2:2 and combine the data per month in the training, validation, and test datasets, considering the effects of seasons. We use the same data partitioning method in the pre-training and fine-tuning phases. The codes and processed datasets are available here [2].

B. Baselines

We select the trajectory representation learning methods that *adopt self-supervised training methods* and are *suitable for multiple downstream tasks*, *i.e.,* non-task-specific methods, as

[1] https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i

[2] https://github.com/aptx1231/START

TABLE II
THREE DOWNSTREAM TASKS OVERALL PERFORMANCE ON BJ AND PORTO.

		Travel Time Estimation			Trajectory Classification			Most Similar Trajectory Search		
BJ	Models	MAE ↓	MAPE(%) ↓	RMSE ↓	ACC ↑	F1 ↑	AUC ↑	MR ↓	HR@1 ↑	HR@5 ↑
	traj2vec	10.13±0.12	37.95±0.92	56.83±0.47	0.811±6e-4	0.852±1e-3	0.873±2e-5	7.186±0.03	0.607±1e-4	0.766±7e-5
	t2vec	10.03±0.10	36.42±1.31	56.65±0.12	0.814±1e-3	0.863±1e-2	0.879±9e-4	5.948±0.01	0.788±3e-4	0.935±8e-5
	Trembr	9.997±0.11	34.20±0.88	36.97±0.38	0.818±1e-3	0.871±2e-3	0.880±2e-3	2.509±2e-3	0.884±5e-4	0.952±6e-5
	Transformer	10.74±0.48	39.61±1.52	57.16±0.56	0.794±2e-3	0.845±1e-3	0.846±1e-3	40.60±0.19	0.515±1e-4	0.649±1e-4
	BERT	10.21±0.14	37.31±0.17	37.09±0.36	0.804±3e-3	0.862±2e-3	0.864±3e-3	27.10±0.11	0.587±3e-3	0.712±4e-4
	PIM	10.19±0.09	39.04±0.58	57.73±0.26	0.803±1e-3	0.861±1e-3	0.862±9e-4	23.51±0.12	0.760±4e-3	0.898±2e-4
	PIM-TF	12.05±0.03	43.14±0.66	61.15±0.33	0.789±1e-3	0.849±2e-3	0.842±6e-3	86.45±0.32	0.296±2e-4	0.340±2e-4
	Toast	10.69±0.22	35.37±1.14	57.41±0.41	0.810±2e-3	0.870±2e-3	0.871±2e-3	29.53±0.15	0.611±2e-4	0.746±3e-4
	START	**9.134**±0.03	**30.92**±0.35	**35.40**±0.09	**0.853**±2e-3	**0.896**±1e-3	**0.916**±4e-4	**1.295**±1e-3	**0.969**±4e-4	**0.997**±4e-5
	Improve	8.63%	9.59%	4.24%	4.28%	2.87%	4.09%	48.39%	9.62%	4.73%
Porto	Models	MAE ↓	MAPE ↓	RMSE ↓	Micro-F1 ↑	Macro-F1 ↑	Recall@5 ↑	MR ↓	HR@1 ↑	HR@5 ↑
	traj2vec	1.552±6e-3	23.70±0.35	2.351±4e-3	0.063±3e-3	0.038±3e-3	0.183±5e-3	30.52±0.13	0.552±3e-4	0.732±5e-4
	t2vec	1.539±5e-3	23.65±0.12	2.324±5e-3	0.068±2e-4	0.048±3e-4	0.187±3e-4	12.70±0.08	0.746±3e-4	0.856±8e-4
	Trembr	1.480±2e-3	22.64±0.37	2.164±0.01	0.071±9e-4	0.049±1e-3	0.192±2e-3	4.635±1e-3	0.846±4e-4	0.929±8e-5
	Transformer	1.738±3e-3	25.72±0.26	2.637±2e-3	0.028±7e-3	0.018±5e-3	0.075±8e-3	68.58±0.21	0.447±2e-4	0.664±5e-5
	BERT	1.593±7e-3	24.63±0.57	2.291±3e-3	0.065±3e-4	0.044±1e-3	0.184±1e-3	39.12±0.15	0.511±4e-3	0.714±5e-4
	PIM	1.559±3e-3	24.68±0.25	2.339±0.01	0.061±4e-4	0.037±3e-4	0.153±5e-4	19.53±0.10	0.653±3e-4	0.774±7e-4
	PIM-TF	1.945±2e-3	28.82±0.15	2.841±3e-4	0.025±4e-3	0.016±5e-3	0.069±7e-3	78.78±0.24	0.384±2e-5	0.547±3e-5
	Toast	1.624±8e-3	24.63±0.33	2.445±5e-3	0.062±1e-3	0.035±4e-4	0.181±1e-3	22.61±0.12	0.684±2e-5	0.789±2e-5
	START	**1.334**±3e-3	**20.66**±0.14	**2.001**±1e-3	**0.089**±4e-4	**0.067**±2e-3	**0.244**±1e-3	**1.897**±1e-3	**0.921**±3e-4	**0.973**±6e-5
	Improve	9.86%	8.75%	7.53%	25.35%	36.73%	27.08%	59.07%	8.87%	4.74%

[*] All experiments are repeated ten times, and we report both the mean and standard deviation. The bold results are the best, and the underlined results are the second best. The metric with ”↑” means that a larger result is better, and the metric ”↓” means that a smaller result is better.

our baselines. The baselines meet the criteria include three categories:

(1) *Encoder-decoder with reconstruction*: This category uses an RNN-based encoder-decoder model to convert raw trajectories as representation vectors and adopts the reconstruction self-supervised task to train the encoder-decoder model. We select the following representative methods as the baselines.

- Traj2vec [9] converts trajectories to feature sequences and uses a sequence-to-sequence (seq2seq) model to learn representations.
- T2vec [8] is the state-of-the-art seq2seq trajectory representation method with negative sampling and spatial proximity aware loss. We use the decoder of t2vec to recover the input trajectory without downsampling since the data are road-network constrained trajectories.
- Trembr [7] is a seq2seq model whose decoder reconstructs both roads and timestamps of the input trajectory.

(2) *Two-stage representation models*: This category first converts road segments as representation vectors and then generates trajectory representation vectors from the road representation vectors in the same trajectory. We select the following methods of this category as the baselines.

- PIM [6] uses node2vec to generate road representations of the static road network and uses a mutual information maximization method to train a LSTM encoder for trajectory representation generation.
- PIM-TF replaces the LSTM encoder in PIM with a transformer encoder.
- Toast [5] uses the context-aware node2vec to generate road representations and uses the MLM and trajectory discrimination task to train a Transformer encoder for trajectory representation generation.

Besides, we also adopt classical self-supervised sequence representation learning models as the baselines.

(3) *Self-supervised sequence representation models*: We adopt Transformer and BERT that input with road-network constrained trajectories as the baselines.

- Transformer [11] is a self-attention model of the encoder-decoder architecture. We use MLM as the pre-training self-supervised task.
- BERT [10] is a self-attention model. We train the model with a MLM task and a classification task, splitting a trajectory $\mathcal{T}$ as two parts, *i.e.,* $\mathcal{T}_1$ and $\mathcal{T}_2$, and treating $(\mathcal{T}_1, \mathcal{T}_2)$ as positive samples and $(\mathcal{T}_2, \mathcal{T}_1)$ as negative samples.

The models mentioned in the related work section but requiring supervised labels, such as NEUTRAJ [19], Traj2SimVec [26] and T3S [20], and the methods that are not suitable for multiple downstream tasks, such as DETECT [27] and E2dtc [18], are not chosen as the baselines.

C. Experimental Settings

1) Model and Baseline Settings: All experiments are conducted on Ubuntu 18.04 with an NVIDIA GeForce 3090 GPU. We implement START and all baselines based on the PyTorch 1.7.1 [2]. We set the embedding size d to 256, the TPE-GAT layers L_1 to 3, and the TAT-Enc layers L_2 to 6. The attention heads H_1 are [8, 16, 1] for TPE-GAT and H_2 is 8 for TAT-Enc. The mask length l_m is 2 and the mask ratio p_m is 15%. The dropout ratio is 0.1, and the temperature parameter τ is 0.05. The default data augmentation methods are *Trajectory Trimming* and *Temporal Shifting*. Finally, $\lambda = 0.6$ to balance the pre-training losses. The baselines have the same settings as START, with 256 hidden dimensions and six layers (or six encoders and six decoders for the encoder-decoder model), and the other settings follow their defaults.

2) Training Settings: We pre-train and fine-tune our model using the optimizer AdamW [16]. The batch size is 64, and

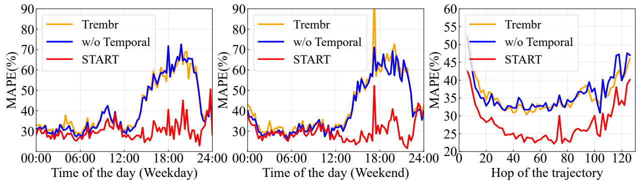


Fig. 3. MAPE on BJ Under Different Scenarios.

the training epoch is 30. The learning rate lr is 0.0002, and we use the warm-up policy corresponding to increase lr linearly for the first five epochs and decrease it after using a cosine annealing schedule.

3) Evaluation Metrics: For the travel time estimation task, we adopt three metrics, including mean absolute error (MAE), mean absolute percentage error (MAPE), and root mean square error (RMSE). For the trajectory classification task, we use Accuracy (ACC), F1-score (F1), and Area Under ROC (AUC) to evaluate binary classification tasks, and Micro-F1, Macro-F1, and Recall@5 to evaluate multi-classification tasks. For the most similar search task, we use Mean Rank(MR) and Hit Ratio(HR@1, HR@5) to evaluate whether the model can find the truth. For the k-nearest search task, we use Precision to measure the coverage of the top-k results.

D. Performance Comparision

1) Overall Performance: Table II reports the overall results on the three downstream tasks. Based on the table, we can make the following observations.

- Our START achieves the best performance in terms of all metrics on these three tasks for the two real-world datasets. It confirms the superior performance of our framework in learning trajectory representations by introducing temporal regularities and travel semantics in the pre-training phase.
- The encoder-decoder models with reconstruction outperform the sequence representation models (Transformer, BERT). It could be because the pre-training methods of these two models from the natural language processing domain are not suitable for trajectory data, ignoring the spatial-temporal characteristics.
- Trembr performs best among all baselines because it considers the visit timestamp of each road in the decoding process, highlighting the importance of temporal information in the trajectories.
- The performance of the two-stage models, *i.e.,* PIM and Toast, is unsatisfactory due to two factors. First, they consider trajectories as ordinary road sequences and ignore the temporal information. Second, their road representation learning method does not adequately consider the travel semantics, such as road visit frequencies.

2) Performance of Trajectory Travel Time Estimation: We fine-tune all models with the objective function (16). Note that no time information is fed into the model during fine-tuning, except for the *departure time* to avoid information leakage. In addition to the overall performance in Table II, to investigate the performance of the model under different scenarios and verify the role of pre-training with temporal regularities, we present the MAPE results on different departure times, whether it is a weekend or not, and the hops of the trajectory on BJ in Figure 3. Here we compare three models, including START, a variant without temporal (noted as *w/o Temporal*) where the time embeddings and the time interval matrix are removed, and the best baseline Trembr. We can observe the following phenomena: (1) START consistently outperforms others, regardless of the weekday or weekend or the trajectory hop size. Moreover, START shows excellent performance, especially in the late peak periods (16:00-21:00) and when the trajectory is between 20 to 100 hops. (2) The no temporal variant cannot capture temporal regularities and therefore has worse performance than START, highlighting the importance of temporal regularities when pre-training.

3) Performance of Trajectory Classification: We fine-tune all models with the objective function (17). We use whether the taxi carries passengers as a binary classification label in BJ and the driver ID as the label in Porto for the multi-classification (435 classes). As shown in Table II, our model consistently outperforms all baselines because it can capture the underlying travel semantics and achieves accurate performance.

4) Performance of Trajectory Similarity Search: The similarity measure is a fundamental problem with various applications, such as identifying popular routes and similar drivers in trajectory analysis. A recent study [8] proposes to use the most similar trajectory search and the k-nearest trajectory search to evaluate the effectiveness of different methods. We adopt it in the experiments as it is currently the best evaluation method that proves the effectiveness of the model from multiple perspectives. Here we directly use the trajectory representations obtained from the pre-training without fine-tuning and use the Euclidean distance of the representations to represent the similarity between the trajectories, *i.e.,* the smaller the distance, the greater the similarity.

(a) Most Similar Trajectory Search: The most similar trajectory search task is to find out the most similar trajectory $\mathcal{T}'_a$ from a large trajectory database $\mathcal{D}_\mathcal{D}$ given a query trajectory $\mathcal{T}_a$ in the query dataset $\mathcal{D}_\mathcal{Q}$. However, the lack of ground truth makes it difficult to evaluate the accuracy of trajectory similarity. Li *et al.* [8] use downsampling in various proportions to construct the query and ground truth from the GPS-based trajectories. However, the influence of downsampling can be eliminated after the *map matching*. Chen *et al.* [5] propose a detour method to generate ground truth for road-network constrained trajectories. Based on this, we propose a ground truth generation method based on the top-k detour. Specifically, we randomly select N_q trajectories from the test dataset, denoted as the query dataset $\mathcal{D}_\mathcal{Q}$. For each trajectory $\mathcal{T}_a \in \mathcal{D}_\mathcal{Q}$, we select a section of consecutive sub-trajectories $\mathcal{S}_a$ whose length does not exceed p_d (*e.g.,* 0.2) of the original trajectory length. Then we perform a top-k search [24] on the road network between the origin and destination of $\mathcal{S}_a$. If the travel time of the searched trajectory exceeds a certain threshold t_d with respect to the original trajectory, this trajectory is defined as $\mathcal{S}'_a$. The detour trajectory

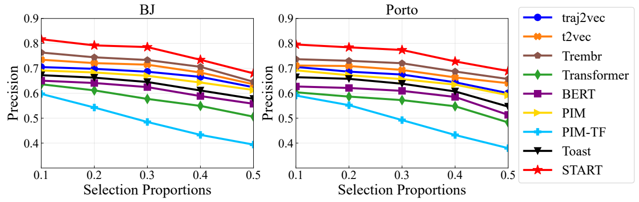


Fig. 4. Performance of k-nearest Trajectory Search Task When Selection Proportions p_d Vary.

$\mathcal{T}_a'$ of $\mathcal{T}_a$ is obtained by replacing $\mathcal{S}_a$ by $\mathcal{S}_a'$. In this way, we can construct the detour dataset $\mathcal{D}_\mathcal{Q}' = \{\mathcal{T}_a'\}$. Furthermore, we extract other N_{neg} trajectories from the test dataset that do not overlap with $\mathcal{D}_\mathcal{Q}$, defined as the set $\mathcal{D}_\mathcal{N}$, and use the same method to obtain the corresponding detour dataset $\mathcal{D}_\mathcal{N}'$. Together, $\mathcal{D}_\mathcal{N}'$ and $\mathcal{D}_\mathcal{Q}'$ form the database $\mathcal{D}_\mathcal{D} = \mathcal{D}_\mathcal{N}' \cup \mathcal{D}_\mathcal{Q}'$. When using $\mathcal{T}_a$ to query the most similar trajectory in $\mathcal{D}_\mathcal{D}$, $\mathcal{T}_a'$ will ideally rank first since it is generated from $\mathcal{T}_a$, *i.e.,* the ground truth of $\mathcal{T}_a$.

In the experiments we set $N_q = 10,000$, $N_{neg} = 100,000$, select proportion $p_d = 0.2$, time threshold $t_d = 0.2$. Table II shows detailed results, and our model outperforms all baselines, especially in the mean rank (MR) metric. We attribute this to the fact that the representations learned by the model capture the travel semantics of the trajectories. In this way, the model can find the shape and semantically similar trajectories, an advantage that sequence-to-sequence models do not have.

(b) k-nearest Trajectory Search: In the k-nearest search task, given a query trajectory, models need to find top-k similar trajectories from the target database, ignoring the rank. Here, we use each query trajectory $\mathcal{T}_a$ in the query dataset $\mathcal{D}_\mathcal{Q}$ to find the k-nearest-neighbors from the database $\mathcal{D}_\mathcal{D}$ as ground truth. Then we construct the transformed detour dataset $\mathcal{D}_\mathcal{Q}'$ from $\mathcal{D}_\mathcal{Q}$ using the same method as above. For each transformed query $\mathcal{T}_a' \in \mathcal{D}_\mathcal{Q}'$, we find the k-nearest-neighbors from database $\mathcal{D}_\mathcal{D}$ and compare them to the ground truth. Since different selection proportions p_d significantly change the generated trajectories, we vary p_d from 0.1 to 0.5 to generate multi-data to evaluate the models. Figure 4 shows the Precision of different models when the p_d is varied and k is fixed at 5. The Precision of all methods decreases as the selection proportion p_d increases. START always stays ahead and decreases more slowly, while Transformer, BERT, PIM-TF, and Toast perform less well. This is likely because the representations learned by the self-attention model are anisotropic [25] and difficult to adapt to downstream tasks without fine-tuning. If we change the time threshold t_d, we obtain similar results not reported here.

(c) Comparision of Top-3 Similar Trajectories: To intuitively examine the search results of our proposed START, we randomly select two trajectories and retrieve the top-3 similar trajectories using START and Trembr, respectively, as shown in Figure 5. The results show that START can find diverse trajectories that are not exactly consistent with the query, but their overall trends (shape, OD, etc.) are similar. Compared with Trembr, the trajectory found by START is closer to the query, while the result of Trembr deviates more from the query, especially the top-3 of query 8379. This illustrates the effectiveness of our proposed START in capturing the global features and travel semantics of the trajectory.

TABLE III
PERFORMANCE COMPARISON WHEN TRANSFER MODEL ACROSS DATASETS.

	Travel Time Estimation			Trajectory Classification		
Models	MAE	MAPE(%)	RMSE	Micro-F1	Macro-F1	Recall@2
No Pre-train Geolife	12.325	78.547	19.584	0.519	0.498	0.790
Pre-train Geolife	11.980	73.489	18.613	0.568	0.571	0.814
Porto-START	10.455	65.371	18.024	0.623	0.619	0.832
BJ-START	**9.995**	**64.331**	**17.183**	**0.669**	**0.665**	**0.887**
Porto-Trembr	15.200	80.294	23.223	0.507	0.468	0.728
BJ-Trembr	14.851	79.239	23.109	0.512	0.486	0.741

E. Effect of Pre-training

In this section, we verify the effectiveness of the two pre-training tasks we designed in two ways. One is to explore whether the training data size can be reduced by pre-training. The other is to investigate whether the pre-trained model can be transferred to other small datasets, even with a heterogeneous road network, to solve the problem of insufficient training data in many real-world applications.

1) Performance Over Small Size Datasets: One of the advantages of pre-training is that it can reduce the use of training data. We reduce the training data size for pre-training and fine-tuning and compare the proposed START with the variant without pre-training (noted as *No Pre-train*), *i.e.,* trained in a supervised manner. Figure 6 shows performance on the entire test dataset of travel time estimation (ETA) and trajectory classification. We vary the size of the training data from 100k to 400k and train both *No Pre-train* and START. We find that the performance of both models improves with more labeled data, and START consistently outperforms the *No Pre-train* variant regardless of the training data size. Besides, as more data is used for pre-training, the performance of the model improves more significantly. These experiments show that pre-training can effectively reduce the use of training data.

2) Transfer Model Across Datasets: We transfer the model that pre-trained on a large dataset to another small dataset for fine-tuning, with the expectation that the knowledge learned from the large dataset will be transferred to the small one to solve the problem of insufficient training data. The small dataset is Geolife [3], a public dataset consisting of trajectories from 2007 to 2012 in Beijing. Since we need to perform *map matching*, we only keep the trajectories with four transportation modes, including Car/Taxi, Walk, Bike, and Bus. In this way, we obtain 5,760 trajectories after data processing. In Table III, we compare the performance of the following models: (1) START trained directly or pre-trained with fine-tuned on Geolife (noted as *No Pre-train Geolife*, *Pre-train Geolife*), (2) START pre-trained on BJ and Porto and fine-tuned on Geolife (noted as *BJ-START*, *Porto-START*), and (3) the best baseline Trembr pre-trained on BJ and Porto and fine-tuned on Geolife (noted as *BJ-Trembr*, *Porto-Trembr*). Since the source datasets are cab datasets, we use only the 882 Car/Taxi mode trajectories from the Geolife data for travel

[3] https://research.microsoft.com/en-us/projects

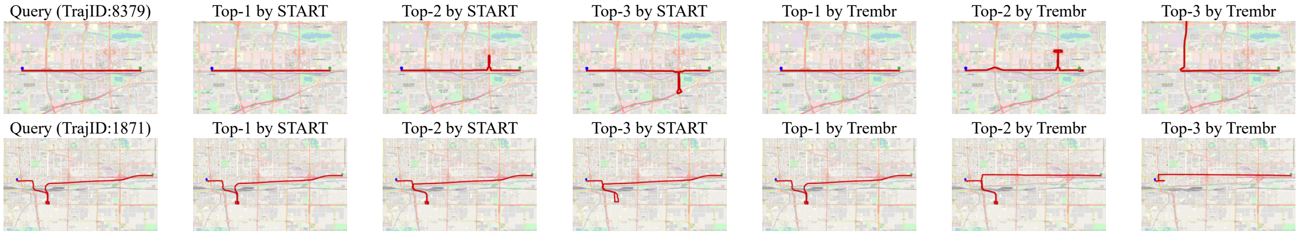
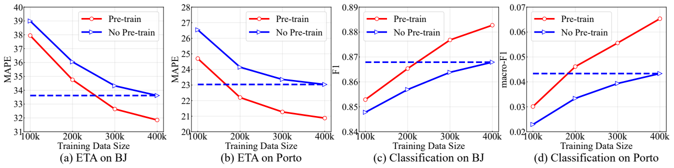


Fig. 5. Comparision of Top-3 Similar Trajectories Retrieved by START and Trembr. (Map data © OpenStreetMap contributors, CC BY-SA.)



(a) ETA on BJ
(b) ETA on Porto
(c) Classification on BJ
(d) Classification on Porto

Fig. 6. Performance When Train Size Vary.

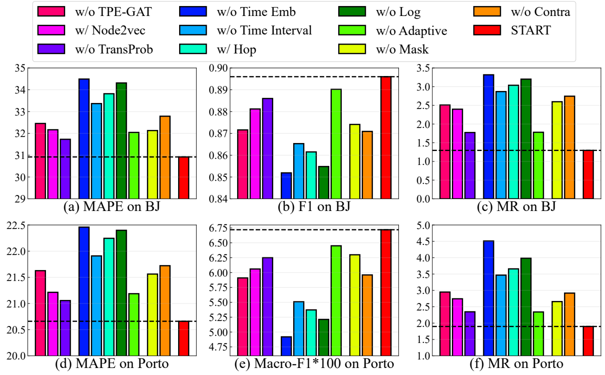
time prediction. The label for the trajectory classification is the four transportation modes.

We can conclude the following: (1) Direct pre-training on small datasets can also improve performance compared to non-pre-training. (2) Our proposed START, whether pre-trained on BJ or Porto, outperforms the model pre-trained on Geolife, showing that pre-training can significantly improve performance on small datasets through knowledge transfer. The model pre-trained on BJ performs better than the model pre-trained on Porto because BJ and Geolife have the same road network. The parameters of our TPE-GAT layer are independent of the number of roads so that it can learn the road representations as long as the road network and the road features are given. Therefore, we can transfer START to a heterogeneous road network dataset. In terms of performance improvement, we argue that the model learns the deep travel semantics of trajectories that are similar between different cities, thus enabling improvements. (3) When the pre-trained Trembr model is transferred to the Geolife dataset, performance is even worse. This confirms that the sequence-to-sequence model is unsuitable for transferring between datasets. Instead, our proposed START is suitable for pre-training and transfer learning to solve the insufficient data problem.

F. Ablation Study

To further investigate the effectiveness of each sub-module in START, we conduct the following ablation experiments on both datasets. All experiments are repeated ten times and report the average results in Figure 7. Due to space limitations, we show only one metric for each task.

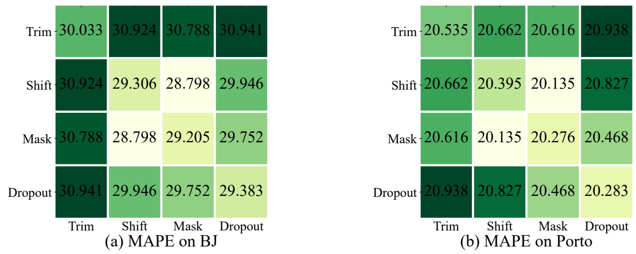
1) Impact of Trajectory Pattern-Enhanced Graph Attention Layer: (a) *w/o TPE-GAT*: this variant replaces the TPE-GAT with randomly initialized learnable road embeddings. (b) *w/ Node2vec*: the variant replaces the TPE-GAT with learnable road embeddings initialized by node2vec [17] algorithms. (c) *w/o TransProb*: this variant removes the transfer probability matrix in the TPE-GAT. We can see that performance drops significantly without the TPE-GAT. The variant *w/o TransProb* removes the transfer probability matrix so that the TPE-GAT degenerates to a standard GAT. This variant performs better than the variant *w/ Node2vec* because compared with a standard GAT, the node2vec only focuses on the road network structure but ignores the road features. Moreover, this variant performs worse than the original model, reflecting that with transfer probabilities, the learned travel pattern enhanced-road representations are more valuable than the simple aggregation of the neighbors' features by a standard GAT.



(a) MAPE on BJ
(b) F1 on BJ
(c) MR on BJ
(d) MAPE on Porto
(e) Macro-F1*100 on Porto
(f) MR on Porto

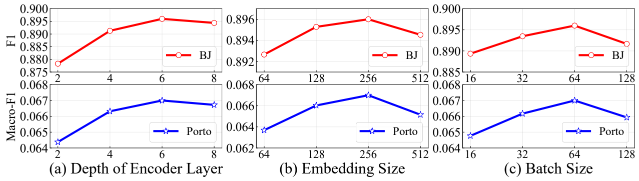
Fig. 7. Ablation Study on BJ and Porto.

2) Impact of Time-Aware Trajectory Encoder Layer: (a) *w/o Time Emb*: this variant drops the temporal embeddings $(\boldsymbol{t}_{mi}, \boldsymbol{t}_{di})$ to ignore the periodic temporal patterns. (b) *w/o Time interval*: this variant drops the time interval matrix $\tilde{\boldsymbol{\Delta}}$. (c) *w/ Hop*: this variant uses the number of hops between roads instead of the time interval to obtain the relative distance, *i.e.,* using $\delta_{i,j} = |i - j|$ instead of $\delta_{i,j} = |t_i - t_j|$. (d) *w/o Log*: this variant replaces the logarithmic function that processes the time interval, *i.e.,* using $\delta'_{i,j} = 1/\delta_{i,j}$ instead of $\delta'_{i,j} = 1/\log(\mathrm{e} + \delta_{i,j})$. (e) *w/o Adaptive*: this variant drops the Eq. (9), *i.e.,* $\tilde{\delta}_{i,j} = \delta'_{i,j} = 1/\log(\mathrm{e} + \delta_{i,j})$. In this way, the time interval matrix remains constant during the training process. We can see that the performance decreases significantly after neglecting the periodic temporal patterns (*i.e., w/o Time Emb*). It confirms the necessity of introducing periodic urban patterns. Besides, removing the time interval matrix leads to significant performance degradation. The variant *w/ Hop* performs worse than *w/o Time interval*, illustrating the importance of using the time interval between roads to measure the impacts among roads rather than using the hop distance. Similarly, the variant *w/o Log* performs worse than *w/o Time interval* because the inverse function changes too little at larger time intervals. The model performance also decreases if the matrix remains constant during the training process (*i.e., w/o Adaptive*), indicating the value of making the time interval matrix adaptive in model learning.



(a) MAPE on BJ (b) MAPE on Porto

Fig. 8. MAPE for Different Data Augmentations.



(a) Depth of Encoder Layer (b) Embedding Size (c) Batch Size

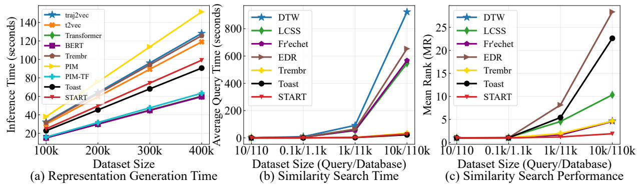
Fig. 9. Parameter Sensitivity Analysis.

3) Impact of Self-Supervised Tasks: (a) *w/o Mask*: this variant removes the span-masked loss and trains only with $\mathcal{L}^{con}$. (b) *w/o Contra*: this variant removes the contrastive loss and trains only with $\mathcal{L}^{mask}$. Results demonstrate that both self-supervised pre-training tasks significantly affect the performance of downstream tasks.

4) Impact of Data Augmentation Strategies: Data augmentation strategies are central in contrastive learning to capture the spatial-temporal characteristics and travel semantics. We show performance with different pairs of methods to explore our proposed four trajectory data augmentation strategies. Due to space constraints, we only show the performance of travel time prediction. As shown in Figure 8, we use a 4*4 grid to show the performance of different pairs of augmentation methods, where each row and column of the grid represents a data augmentation method. Note that the smaller the MAPE, *i.e.,* the lighter the color, the better the performance. We find that *Temporal Shifting* and *Road Segments Mask* perform best in this task. This shows that temporal regularities matter since both methods exhibit a change in the temporal dimension. Besides, *Dropout* is a simple but efficient strategy that does not break the semantics of the trajectory.

G. Parameter Sensitivity

We further conduct the parameter sensitivity analysis for critical hyperparameters, *e.g.,* encoder layers L_2, embedding size d, and batch size N_b on both datasets. We report only the results of trajectory classification, and the results of the other tasks are similar. From Figure 9, we can see that the model performance initially improves with d and L_2 increasing, but when they are too large, the performance deteriorates due to overfitting. Although previous studies have generally recommended larger batch sizes for contrastive learning [15], experiments have shown that model performance drops when batch sizes are too large. This may be due to a large batch introducing too many "hard" negative samples that differ minimally from the given anchor, *e.g.,* trajectories between the same ODs departing simultaneously. It is inappropriate to set these two semantically similar samples as a negative pair.



(a) Representation Generation Time (b) Similarity Search Time (c) Similarity Search Performance

Fig. 10. Efficiency And Scalability Analysis.

H. Model Efficiency And Scalability

Since START is a pre-trained method for learning representations that can be trained offline, in practicality, we are more concerned with the time cost of encoding the trajectories into representations and applying them to downstream tasks. Here we use the default settings described in Section IV-C. We report only the results for Porto due to space limitations, and the results for BJ are similar.

First, we report the inference time of START and baselines, *i.e.,* the time cost of embedding 100k-400k trajectories in Figure 10(a). The results show that self-attention models are faster than RNN models. This is because RNN model need $O(L)$ sequential operations to process the trajectory while self-attention model only need $O(1)$. Here L is the length of the trajectory. Besides, START is slightly slower than other self-attention models because it introduces the TPE-GAT layer and the time interval matrix, a tradeoff between performance and efficiency. Even so, it takes only 25.8 seconds to encode 100,000 trajectories, and it can get even faster as the batch size gets larger during inference.

In addition, we also compare the time cost of similarity search. Figure 10(b) shows the average time cost of a query for performing the most similar search with different query sizes and database sizes. The size of the query and detour dataset N_q varies from 10 to 10,000, and the size of the negative samples N_{neg} is ten times N_q. In addition to the two representative deep models Toast and Trembr, we also compare some traditional algorithms including Dynamic Time Warping (DTW) [30], Longest Common SubSequence (LCSS) [28], Fr'echet Distance [31], and Edit Distance on Real Sequence (EDR) [29]. For the deep models, we sum the time cost of obtaining the representations and computing the similarities using the representations.

We can see deep models are at least one order of magnitude faster than the traditional algorithms because the complexity of the traditional algorithms for computing the similarity is generally $O(L^2)$, while the deep models require only $O(d)$ complexity for computing the distance between the representations. Here L is the length of the trajectory, and d is the embedding size. Moreover, the deep models will be more efficient if we generate representations offline. The linear complexity makes START scale well on large datasets. Moreover, as shown in Figure 10 (c), START outperforms traditional algorithms on mean rank (MR) for search. This shows that START is not only efficient but also can be used directly as a powerful metric for computing trajectory similarity without fine-tuning.

Finally, from the two experiments above, it appears that both the time for inference and the time for similarity search of START increases linearly with the amount of data, which means START can be scaled for large datasets.

V. Related Work

A. Trajectory Representation Learning

Trajectory Representation Learning (TRL) is a powerful tool for spatial-temporal data analysis and management. TRL aims to convert raw trajectories into generic low-dimensional representation vectors that can be applied to various downstream tasks. Two of the earliest works that introduce the concept of trajectory representation learning into trajectory data management are t2vec [8] and traj2vec [9]. T2vec [8] is trained by reconstructing high-sampling trajectories from low-sampling trajectories. Traj2vec [9] transforms trajectories into feature sequences and trains a sequence-to-sequence (seq2seq) model based on reconstruction loss. Since then, many trajectory representation learning methods have been proposed for specific downstream tasks. For example, NEUTRAJ [19], Traj2SimVec [26], and T3S [20] aim to learn trajectory representations for approximate trajectory similarity computation. In addition, DETECT [27] and E2dtc [18] build a seq2seq model trained with a reconstruction loss and a cluster-oriented loss to learn representations for trajectory clustering. GM-VSAE [32] and D-TkDI [36] learn trajectory representations for anomalous trajectory detection and path ranking, respectively.

Most previous TRL works consider trajectories as sequences of locations, such as road segments, GPS sample points, or POI points while ignoring the corresponding temporal information. To the best of our knowledge, before our work, Trembr [7] was the only work that considered temporal information in self-supervised trajectory representation learning. Trembr is an RNN-based encoder-decoder model that considers the timestamps of each location in the decoding process. However, Trembr does not capture the periodic patterns of urban traffic or the irregular time intervals between trajectory samples. Our model explicitly incorporates the two temporal regularities into the trajectory representations, so it outperforms Trembr in the experiments. In addition to GPS trajectories, some studies focus on other types of trajectories. TRED [14], CTLTR [13], and SelfTrip [12] are semi- or self-supervised representation methods for trip recommendations based on sparse POI check-in trajectories.

In recent years, some two-stage methods have been proposed to learn generic trajectory representations for multiple downstream tasks [5], [6]. These methods first adopt a graph representation learning model to learn the road representation vectors and then use sequence learning models with self-supervised tasks to convert the road representation vectors in the same trajectory into the trajectory representations. For example, Toast [5] and PIM [6] use node2vec [17] to learn road representations and respectively use Transformer with masked prediction and RNN with mutual information maximization as self-supervised tasks to generate trajectory representations. Compared to our work, these two-stage methods consider trajectories as ordinary sequence data and thus ignore the temporal information. Besides, they only incorporate the static road network as spatial semantic information while ignoring the travel semantics, such as road visit frequencies.

B. Self-supervised Learning

Self-supervised learning is a technique that enables learning with unlabeled data and has recently achieved remarkable success in various fields, such as computer vision [15], natural language processing [10], [22], and data engineering [40], [41]. Self-supervised methods primarily include generative, predictive, and contrastive methods [23]. The generative methods learn representations based on reconstruction losses, such as some seq2seq models mentioned in Section V-A. The predictive methods construct labels based on the input data, such as BERT [10], using the mask language prediction for training. The contrastive methods construct positive and negative samples and train the models to close the distance between positive pairs and push the distance between negative pairs. SimCLR [15] is a contrastive learning method for visual representations using normalized temperature-scaled cross-entropy loss (NT-Xent) as training loss. SimCSE [21] uses standard dropout as noise to construct positive instances for sentence embeddings. ConSERT [22] proposes four different types of contrastive learning data augmentation methods for learning sentence embeddings. Although several works have focused on the self-supervised learning of trajectories, our framework is the first to use both predictive and contrastive methods to capture the temporal regularities and travel semantics of the road-network constrained trajectory.

VI. Conclusion and Future work

In this paper, we proposed a two-stage trajectory representation learning method, START, which incorporated temporal regularities and travel semantics into generic trajectory representation learning. Furthermore, we designed two self-supervised tasks to train our START, which fully considered the spatial-temporal characteristics of trajectories. Extensive experiments on two large-scale datasets for three downstream tasks confirmed the superior performance of our proposed framework compared with the state-of-the-art baselines. The experiment results also demonstrated that our methods could be transferred across heterogeneous trajectory datasets, which was very useful for solving the problem of insufficient data.

In the future, we plan to explore more data augmentation techniques for contrastive learning according to the specific downstream tasks and extend the proposed framework to other categories of trajectory data, such as POI check-in trajectories, to support more applications.

Acknowledgment

This work was supported by the National Key R&D Program of China (2019YFB2103203). Prof. Wang's work was supported by the National Natural Science Foundation of China (No. 82161148011, 72222022, 72171013), and the DiDi Gaia Collaborative Research Funds.

References

[1] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2017.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[3] C. Yang and G. Gidofalvi, "Fast map matching, an algorithm integrating hidden markov model with precomputation," *International Journal of Geographical Information Science*, vol. 32, no. 3, pp. 547–570, 2018.

[4] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *CoRR*, vol. abs/1710.10903, 2017.

[5] Y. Chen, X. Li, G. Cong, Z. Bao, C. Long, Y. Liu, A. K. Chandran, and R. Ellison, "Robust road network representation learning: When traffic patterns meet traveling semantics," in *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management*. ACM, 2021, pp. 211–220.

[6] S. B. Yang, C. Guo, J. Hu, J. Tang, and B. Yang, "Unsupervised path representation learning with curriculum negative sampling," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021*. ijcai.org, 2021, pp. 3286–3292.

[7] T.-Y. Fu and W.-C. Lee, "Trembr: Exploring road networks for trajectory representation learning," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 1, pp. 1–25, 2020.

[8] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *34th IEEE International Conference on Data Engineering, (ICDE), Paris, France, April 16-19, 2018*. IEEE Computer Society, 2018, pp. 617–628.

[9] D. Yao, C. Zhang, Z. Zhu, J. Huang, and J. Bi, "Trajectory clustering via deep representation learning," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 3880–3887.

[10] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT (1)*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[12] Q. Gao, W. Wang, K. Zhang, X. Yang, C. Miao, and T. Li, "Self-supervised representation learning for trip recommendation," *Knowledge-Based Systems*, vol. 247, p. 108791, 2022.

[13] F. Zhou, P. Wang, X. Xu, W. Tai, and G. Trajcevski, "Contrastive trajectory learning for tour recommendation," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 1, pp. 1–25, 2021.

[14] F. Zhou, H. Wu, G. Trajcevski, A. Khokhar, and K. Zhang, "Semi-supervised trajectory understanding with poi attention for end-to-end trip recommendation," *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, vol. 6, no. 2, pp. 1–25, 2020.

[15] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[16] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[17] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*. ACM, 2016, pp. 855–864.

[18] Z. Fang, Y. Du, L. Chen, Y. Hu, Y. Gao, and G. Chen, "E^{2}dtc: An end to end deep trajectory clustering framework via self-training," in *37th IEEE International Conference on Data Engineering, (ICDE), Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 696–707.

[19] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," in *35th IEEE International Conference on Data Engineering, (ICDE), Macao, China, April 8-11, 2019*. IEEE, 2019, pp. 1358–1369.

[20] P. Yang, H. Wang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "T3S: effective representation learning for trajectory similarity computation," in *37th IEEE International Conference on Data Engineering, (ICDE), Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 2183–2188.

[21] T. Gao, X. Yao, and D. Chen, "Simcse: Simple contrastive learning of sentence embeddings," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021*. Association for Computational Linguistics, 2021, pp. 6894–6910.

[22] Y. Yan, R. Li, S. Wang, F. Zhang, W. Wu, and W. Xu, "Consert: A contrastive framework for self-supervised sentence representation transfer," in *ACL/IJCNLP (1)*. Association for Computational Linguistics, 2021, pp. 5065–5075.

[23] J. Yu, H. Yin, X. Xia, T. Chen, J. Li, and Z. Huang, "Self-supervised learning for recommender systems: A survey," *CoRR*, vol. abs/2203.15876, 2022.

[24] J. Y. Yen, "Finding the k shortest loopless paths in a network," *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[25] J. Gao, D. He, X. Tan, T. Qin, L. Wang, and T. Liu, "Representation degeneration problem in training natural language generation models," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[26] H. Zhang, X. Zhang, Q. Jiang, B. Zheng, Z. Sun, W. Sun, and C. Wang, "Trajectory similarity learning with auxiliary supervision and optimal matching," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. ijcai.org, 2020, pp. 3209–3215.

[27] M. Yue, Y. Li, H. Yang, R. Ahuja, Y. Chiang, and C. Shahabi, "DETECT: deep trajectory clustering for mobility-behavior analysis," in *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*. IEEE, 2019, pp. 988–997.

[28] M. Vlachos, D. Gunopulos, and G. Kollios, "Discovering similar multidimensional trajectories," in *Proceedings of the 18th International Conference on Data Engineering (ICDE), San Jose, CA, USA, February 26 - March 1, 2002*. IEEE Computer Society, 2002, pp. 673–684.

[29] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 491–502.

[30] B. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *Proceedings of the Fourteenth International Conference on Data Engineering (ICDE), Orlando, Florida, USA, February 23-27, 1998*. IEEE Computer Society, 1998, pp. 201–208.

[31] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *International Journal of Computational Geometry & Applications*, vol. 5, no. 01n02, pp. 75–91, 1995.

[32] Y. Liu, K. Zhao, G. Cong, and Z. Bao, "Online anomalous trajectory detection with deep generative sequence modeling," in *36th IEEE International Conference on Data Engineering, (ICDE), Dallas, TX, USA, April 20-24, 2020*. IEEE, 2020, pp. 949–960.

[33] Z. Fang, L. Pan, L. Chen, Y. Du, and Y. Gao, "MDTP: A multi-source deep traffic prediction framework over spatio-temporal trajectory data," *Proc. VLDB Endow.*, vol. 14, no. 8, pp. 1289–1297, 2021.

[34] G. Li, C. Hung, M. Liu, L. Pan, W. Peng, and S. G. Chan, "Spatial-temporal similarity for trajectories with location noise and sporadic sampling," in *37th IEEE International Conference on Data Engineering, (ICDE), Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 1224–1235.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[36] S. B. Yang and B. Yang, "Learning to rank paths in spatial networks," in *36th IEEE International Conference on Data Engineering, (ICDE), Dallas, TX, USA, April 20-24, 2020*. IEEE, 2020, pp. 2006–2009.

[37] J. Wang, J. Jiang, W. Jiang, C. Li, and W. X. Zhao, "Libcity: An open library for traffic prediction," in *SIGSPATIAL/GIS*. ACM, 2021, pp. 145–148.

[38] J. Wang, X. Lin, Y. Zuo, and J. Wu, "Dgeye: Probabilistic risk perception and prediction for urban dangerous goods management," *ACM Trans. Inf. Syst.*, vol. 39, no. 3, pp. 28:1–28:30, 2021.

[39] J. Ji, J. Wang, Z. Jiang, J. Jiang, and H. Zhang, "STDEN: towards physics-guided neural networks for traffic flow prediction," in *AAAI*. AAAI Press, 2022, pp. 4048–4056.

[40] H. Ren, J. Wang, W. X. Zhao, and N. Wu, "RAPT: pre-training of time-aware transformer for learning robust healthcare representation," in *KDD*. ACM, 2021, pp. 3503–3511.

[41] N. Wu, W. X. Zhao, J. Wang, and D. Pan, "Learning effective road network representation with hierarchical graph neural networks," in *KDD*. ACM, 2020, pp. 6–14.

[42] J. Wang, N. Wu, W. X. Zhao, F. Peng, and X. Lin, "Empowering a* search algorithms with neural networks for personalized route recommendation," in *KDD*. ACM, 2019, pp. 539–547.