# SimRN: Trajectory Similarity Learning in Road Networks based on Distributed Deep Reinforcement Learning

Danlei Hu
Zhejiang University
dlhu@zju.edu.cn

Yilin Li
Zhejiang University
yilin.23@intl.zju.edu.cn

Lu Chen
Zhejiang University
luchen@zju.edu.cn

Ziquan Fang
Zhejiang University
zqfang@zju.edu.cn

Yushuai Li
Aalborg University
yusli@cs.aau.dk

Yunjun Gao
Zhejiang University
gaoyj@zju.edu.cn

Tianyi Li
Aalborg University
tianyi@cs.aau.dk

## ABSTRACT

Trajectory similarity computation in road networks is crucial for data analytics. However, both non-learning-based and learning-based methods face challenges. First, they suffer from low accuracy due to manual parameter selection for model training and the omission of key spatio-temporal features in road networks. Second, they have low efficiency, stemming from the high time complexity of similarity computation and the time-consuming training process. Third, learning-based methods struggle with poor model generality due to the small size of available training samples.

To address these challenges, we propose an effective and efficient trajectory similarity learning framework for road networks, called SimRN. To our knowledge, SimRN is the first deep reinforcement learning (DRL) approach for trajectory similarity computation. Specifically, SimRN consists of three key modules: the spatio-temporal prompt information extraction (STP) module, the trajectory representation based on DRL (TrajRL) module, and the graph contrastive learning (GCL) module. The STP module captures spatio-temporal features from road networks to improve the training of the trajectory representation. The TrajRL module automatically selects optimal parameters and enables parallel training, improving both trajectory representation and the efficiency of similarity computations. The GCL module employs a self-supervised contrastive learning paradigm to generate sufficient samples while preserving spatial constraints and temporal dependencies of trajectories. Extensive experiments on two real-world datasets, compared with three state-of-the-art methods, show that SimRN: (i) improves accuracy by 20%–40%, (ii) achieves speedups of 2–4x, and (iii) demonstrates strong generality, enabling effective similarity learning with very small sample sizes.

**PVLDB Reference Format:**
Danlei Hu, Yilin Li, Lu Chen, Ziquan Fang, Yushuai Li, Yunjun Gao, and Tianyi Li. SimRN: Trajectory Similarity Learning in Road Networks based on Distributed Deep Reinforcement Learning. PVLDB, 18(7): 2057 - 2069, 2025.
doi:10.14778/3734839.3734844

Proceedings of the VLDB Endowment, Vol. 18, No. 7 ISSN 2150-8097.
doi:10.14778/3734839.3734844

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/ZJU-DAILY/SimRN.

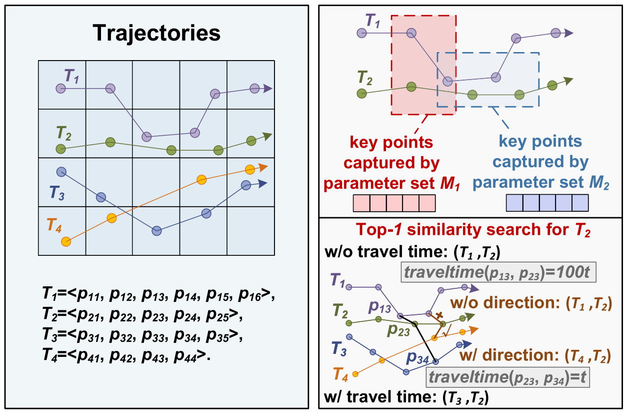
## 1 INTRODUCTION

With the rapid development of location acquisition technologies in the urban Internet of Things (IoT), massive amounts of vehicle trajectory data are being collected. For instance, in 2022, Uber, the world's largest ridesharing company, captured up to 131 million vehicle trajectories daily [42]. This data volume enables advanced trajectory analysis and applications. **Trajectory Similarity Computation** (TSC), which measures the similarity or distance between two trajectories, is a key function in trajectory analysis such as trajectory retrieval [36, 41], clustering [21, 29], classification [20, 25], route planning [31, 57], and outlier detection [27, 28].

Various trajectory similarity computation methods have been proposed, such as Dynamic Time Warping Distance (DTW) [26, 34, 52], Longest Common Subsequence (LCSS) [37, 44], Edit Distance on Real Sequence (EDR) [7, 11], Frechet distance [1, 46], and deep learning based NEUTRAJ [51]. While these methods effectively compute similarities for trajectories in free space, they struggle with handling vehicle trajectories constrained by urban road networks.

Recently, researchers have proposed solutions for road network-constrained trajectories, which can be categorized into non-learning-based and learning-based methods. Non-learning-based methods compute point-to-point distances between two trajectories manually [24, 39, 45, 56], aligning points from one trajectory with consecutive points from another [19]. This results in quadratic time complexity, limiting scalability. Although distributed methods [40, 47] have been introduced to improve efficiency through pruning techniques and parallel processing, their time complexity remains quadratic, which is still not efficient enough. To enhance efficiency, many studies [13, 18, 30, 51, 58] leverage neural networks to measure trajectory similarities under road network constraints. These learning-based methods typically use Graph Convolutional Networks (GCNs) and Recurrent Neural Networks (RNNs) to transform high-dimensional trajectory data into low-dimensional representations. The similarity between these representations can then be efficiently calculated using Euclidean distance [30], reducing time complexity from quadratic to linear. However, these road network oriented learning-based methods still face three key challenges.

***Challenge I: Low accuracy of similarity computation.*** Existing learning-based methods suffer from accuracy. Table 1 reports


Trajectories
$T_1$
$T_2$
$T_3$
$T_4$
$T_1$=<$p_{11}$, $p_{12}$, $p_{13}$, $p_{14}$, $p_{15}$, $p_{16}$>,
$T_2$=<$p_{21}$, $p_{22}$, $p_{23}$, $p_{24}$, $p_{25}$>,
$T_3$=<$p_{31}$, $p_{32}$, $p_{33}$, $p_{34}$, $p_{35}$>,
$T_4$=<$p_{41}$, $p_{42}$, $p_{43}$, $p_{44}$>.
key points captured by parameter set $M_1$
key points captured by parameter set $M_2$
Top-1 similarity search for $T_2$
w/o travel time: ($T_1$ ,$T_2$)
traveltime($p_{13}$, $p_{23}$)=100t
$p_{13}$
$p_{23}$
$p_{34}$
w/o direction: ($T_1$ ,$T_2$)
w/ direction: ($T_4$ ,$T_2$)
traveltime($p_{23}$, $p_{34}$)=t
w/ travel time: ($T_3$ ,$T_2$)

(a) Example trajectories (b) An example of low accuracy

**Figure 1: A motivating example**

the accuracy and efficiency of three state-of-the-art similarity computation methods: GTS [18], ST2Vec [13], and GRLSTM [60]. As observed, *HR@50* of the three methods are suboptimal, with values below 0.6 (a higher *HR@50* indicates better performance [19]). The underlying reasons include two key aspects.

First, similarity learning models rely on various network parameters (e.g., number of layers and learning rates), which directly impact their ability to approximate trajectory similarities. For example, in the upper part of Figure 1(b), two different parameter sets, $\mathcal{M}_1$ and $\mathcal{M}_2$, capture different key points from the same trajectories $T_1$ and $T_2$, leading to different representation vectors. Existing studies [50] rely on manual parameter tuning, which makes it difficult to optimize model performance. However, finding the proper parameters for training is challenging because of the many variables that need to be set in advance. Only a limited number of parameter combinations are typically tested due to the high time costs of tuning, which leads to missed opportunities for optimal tuning.

Second, existing methods focus solely on trajectory features for similarity computation, ignoring essential spatio-temporal features of road networks, such as road direction and travel time. These features are vital for accurate similarity measurement. As shown in the lower part of Figure 1(b), when performing a Top-1 similarity search for $T_2$, $T_1$ appears as the result without considering road direction and travel time. However, the road segments between them are impassable, and the travel time is too long (i.e., *traveltime* = $100t$). In reality, $T_3$ or $T_4$ are much more similar to $T_2$ compared to $T_1$. Unfortunately, extracting these features presents two challenges: (i) the direction and travel time of each road are discovered from trajectory datasets, which may not fully cover all roads, leaving some features unknown; and (ii) travel times are dynamic, varying throughout the day due to fluctuating traffic conditions.

***Challenge II: Low efficiency of model training.*** Although learning-based methods offer linear time complexity for similarity computation, the training process remains time-intensive, especially for large datasets or complex models. For example, as shown in Table 1, GRLSTM's training time is 4580.16 seconds, hundreds of times longer than its similarity computation time, emphasizing the critical need for enhanced training efficiency. To deal with this issue, a natural idea is to utilize distributed techniques to train the

**Table 1: Accuracy and efficiency of similarity computation using three state-of-the-art learning-based methods on the T-Drive dataset, where (i) TP [39] is the similarity measure; (ii) *HR@50* represents the hitting ratio for Top-50 similarity search; (iii) "Training" represents training time; and (iv) "Computing" represents the similarity computation time via trained model.**

| Methods | *HR@50* | Training | Computing |
|---|---|---|---|
| GTS [18] | 0.5609 | 1782.00s | 28.92ms |
| ST2Vec [13] | 0.5767 | 2418.32s | 20.11ms |
| GRLSTM [60] | 0.4379 | 4580.16s | 29.77ms |

model in parallel for improving efficiency. Nevertheless, it is non-trivial to keep load balance of multiple computing GPUs because of the complex model parameters. Therefore, effective distributed partition strategies are needed to ensure balanced load distribution in the similarity learning framework.

***Challenge III: Poor model generality caused by insufficient samples.*** Learning-based methods typically require large amounts of training data, which is often unavailable in real-world road networks with fewer acquisition devices, leading to reduced accuracy. A straightforward solution is to generate more data using augmentation techniques such as random flipping and rotation. However, it is challenging because trajectories have both spatial constraints (road network constraints) and temporal dependencies (timestamps are sequential), making it difficult to generate trajectory samples while preserving these relationships. Therefore, it is crucial to design methods that can achieve good performance with limited samples.

Overall, inspired by the strong decision-making ability of deep reinforcement learning (DRL) [10, 38, 53], we propose SimRN, the first distributed DRL-based trajectory similarity learning framework for road networks. To tackle ***Challenge I***, we leverage DRL to automatically select optimal parameters for similarity learning models by transforming trajectory similarity computation into a decision-making problem, thereby enhancing accuracy. We also extract spatio-temporal features from road networks using data completion and period partitioning, then augment the network to include these features. The augmented network is subsequently used as input for the GCN model during training.

To tackle ***Challenge II***, we develop an actor-learner scheme that processes model training in parallel by assigning decision-making steps to multiple actors and learners, respectively. This supports distributed computing with varying parameters. To further improve SimRN's efficiency, we incorporate an experience replay pool to guide parameter selection and prune invalid parameters. To tackle ***Challenge III***, we present a self-supervised contrastive learning paradigm. Specifically, we propose a positive and negative sample generation method using sub-graph enhancement to create sufficient training samples while preserving the spatial constraints and temporal dependencies of trajectories. Moreover, we extend the standard GCN layer into a multi-channel GCN model to capture spatial constraints, and combine it with an LSTM model to capture temporal correlations, improving similarity learning.

Overall, this paper makes the following contributions.

- We propose SimRN, a novel trajectory similarity learning framework for road networks. To our knowledge, it is the first distributed DRL-based model for trajectory similarity computation.

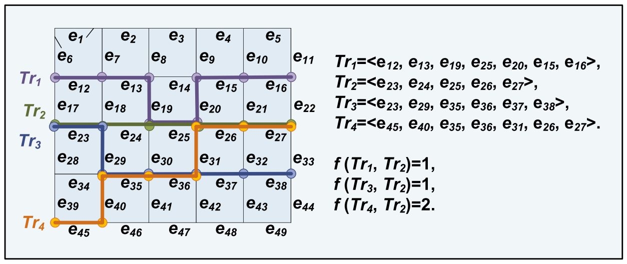

$e_1$ $e_2$ $e_3$ $e_4$ $e_5$
$e_6$ $e_7$ $e_8$ $e_9$ $e_{10}$ $e_{11}$
$Tr_1$
$e_{12}$ $e_{13}$ $e_{14}$ $e_{15}$ $e_{16}$
$e_{17}$ $e_{18}$ $e_{19}$ $e_{20}$ $e_{21}$ $e_{22}$
$Tr_2$
$Tr_3$
$e_{23}$ $e_{24}$ $e_{25}$ $e_{26}$ $e_{27}$
$e_{28}$ $e_{29}$ $e_{30}$ $e_{31}$ $e_{32}$ $e_{33}$
$e_{34}$ $e_{35}$ $e_{36}$ $e_{37}$ $e_{38}$
$e_{39}$ $e_{40}$ $e_{41}$ $e_{42}$ $e_{43}$ $e_{44}$
$Tr_4$
$e_{45}$ $e_{46}$ $e_{47}$ $e_{48}$ $e_{49}$
$Tr_1$=<$e_{12}$, $e_{13}$, $e_{19}$, $e_{25}$, $e_{20}$, $e_{15}$, $e_{16}$>,
$Tr_2$=<$e_{23}$, $e_{24}$, $e_{25}$, $e_{26}$, $e_{27}$>,
$Tr_3$=<$e_{23}$, $e_{29}$, $e_{35}$, $e_{36}$, $e_{37}$, $e_{38}$>,
$Tr_4$=<$e_{45}$, $e_{40}$, $e_{35}$, $e_{36}$, $e_{31}$, $e_{26}$, $e_{27}$>.
$f(Tr_1, Tr_2)=1$,
$f(Tr_3, Tr_2)=1$,
$f(Tr_4, Tr_2)=2$.

**Figure 2: Road network constrained trajectories**

- To enhance similarity computation accuracy, we transform trajectory similarity computation into a decision-making problem, enabling automatic selection of optimal parameters. Moreover, we extract spatio-temporal features from the road network to improve SimRN's training.
- To enhance the training efficiency, we develop an actor-learner scheme for parallel and distributed model training. Moreover, we employ an experience replay pool to store training experiences, guiding parameter selection and pruning invalid parameters.
- To deal with insufficient training samples, we present a self-supervised contrastive learning paradigm to generate sufficient samples while preserving spatial constraints and temporal dependencies of trajectories. In addition, we design a multi-channel GCN combined with an LSTM to capture spatio-temporal constraints for similarity learning.
- We conduct extensive experiments on two real-life datasets, comparing with three state-of-the-arts. The results show that SimRN: (i) improves accuracy by 20%–40%; (ii) achieves speedups of 2–4x; and (iii) demonstrates high model generality, enabling effective similarity learning with very small sample sizes.

The rest of the paper is organized as follows. Section 2 presents preliminaries. Section 3 then details our framework and methods. The experimental results are reported in Section 4. Section 5 reviews related work. Finally, Section 6 concludes the paper.

## 2 PRELIMINARIES

Definition 1. **(GPS Point)** *A GPS point $p = (x, y, t)$ records the longitude $x$ and the latitude $y$ at time $t$.*

Here, longitude $x$ and the latitude $y$ are spatial information, while time $t$ is temporal information.

Definition 2. **(Raw Trajectory)** *A raw trajectory $T$ is an time-ordered sequence of GPS points, i.e., $T = \langle p_i | 1 \le i \le n \rangle$, where $p_i$ is the $i^{th}$ GPS point of $T$.*

Definition 3. **(Road Network)** *A road network is represented as a directed graph $G = (V, E)$, where $V$ is a set of road intersections or end points, and $E$ is a set of directed edges. An edge $e_{i,j} = (v_i, v_j)$ connects $v_i$ to $v_j$.*

The direction between two vertices $v_i$ and $v_j$ ($v_i, v_j \in G$) is denoted as $dir_{i,j}$, where $dir_{i,j} = 0$ indicates that the road is blocked, $dir_{i,j} = 1$ indicates a one-way road, and $dir_{i,j} = 2$ indicates a two-way road. We aim to learn $dir_{i,j}$ for each edge $e_{i,j} = (v_i, v_j) \in E$ to enrich road network information.

Definition 4. **(Road Network Constrained Trajectory)** *Given a road network $G = (V, E)$ and a raw trajectory $T = \langle p_i | 1 \le i \le n \rangle$, The road network constrained trajectory of $T$ is a sequence of connected edges, denoted as $Tr$.*

Example 1. *In Figure 1 (a), $T_1 = \langle p_i | 1 \le i \le 6 \rangle$ is a raw trajectory. In Figure 2, the corresponding road network constrained trajectory for $T_1$ is $Tr_1 = \langle e_{12}, e_{13}, e_{19}, e_{25}, e_{20}, e_{15}, e_{16} \rangle$.*

Note that, this paper focuses on road network constrained trajectories. For simplicity, we refer to them as "trajectories" throughout the rest of the paper.

Definition 5. **(Trajectory Similarity Measure)** *A trajectory similarity measure is a function $f(T_i, T_j)$ to measure the distance between two trajectories $T_i$ and $T_j$.*

We perform Top-$k$ trajectory similarity search to evaluate the performance of trajectory similarity computation.

Definition 6. **(Top-$k$ Similarity Search)** *Given a query trajectory $T$, a set of trajectories $\mathcal{T} = T_i \mid 1 \le i \le |\mathcal{T}|$, and a similarity measure $f(\cdot, \cdot)$, a Top-$k$ similarity search returns a set of trajectories $\mathcal{S}$ such that $\mathcal{S} \subseteq \mathcal{T}$, $|\mathcal{S}| = k$, and for all $T_s \in \mathcal{S}$ and all $T_o \in \mathcal{T} - \mathcal{S}$, $f(T, T_s) \ge f(T, T_o)$.*

Example 2. *Continuing Example 1, given a query trajectory $Tr_2$ and the similarity measure LORS [45], SimRN returns $\mathcal{S} = Tr_4$ for a Top-1 similarity search, as $Tr_2$ and $Tr_4$ have the highest similarity.*

Building on these concepts, we define the trajectory similarity learning problem as follows.

Definition 7. **(Trajectory Similarity Learning)** *Given two trajectories $T_i$ and $T_j$, trajectory similarity learning aims to learn (i) representation vectors $r_i, r_j \in \mathbf{R}^d$ for $T_i$ and $T_j$, and (ii) an approximation function $\mathcal{F}(\cdot, \cdot)$ such that:*

$$\arg\min_{\mathcal{M}} |\mathcal{F}(r_i, r_j) - f(T_i, T_j)|, \tag{1}$$

*where $f(T_i, T_j)$ is the true similarity between $T_i$ and $T_j$ based on a given similarity measure, and $\mathcal{M}$ represents the parameter sets of the neural networks.*

## 3 FRAMEWORK

### 3.1 Framework Overview

Figure 3 illustrates the framework of SimRN, which consists the spatio-temporal prompt information extraction (STP) module, the trajectory representation based on DRL (TrajRL) module, and the graph contrastive learning (GCL) module.

- **STP Module.** This module is decipted in the yellow box of Figure 3. The trajectory dataset contains GPS points, while the corresponding road network includes vertices and edges with their lengths. STP extracts spatio-temporal features (i.e., direction and travel time) from this data as prompt information.
- **TrajRL Module.** This module is decipted in the blue box of Figure 3. To find the optimal model parameters for trajectory similarity learning, TrajRL transforms trajectory similarity representation into a decision-making problem and solves it using DRL. Moreover, TrajRL employs an actor-learner scheme for distributed processing to improve efficiency.
- **GCL Module.** This module is decipted in the green box of Figure 3. GCL performs model training based on the prompt information from STP and parameter selection from TrajRL. Specifically,

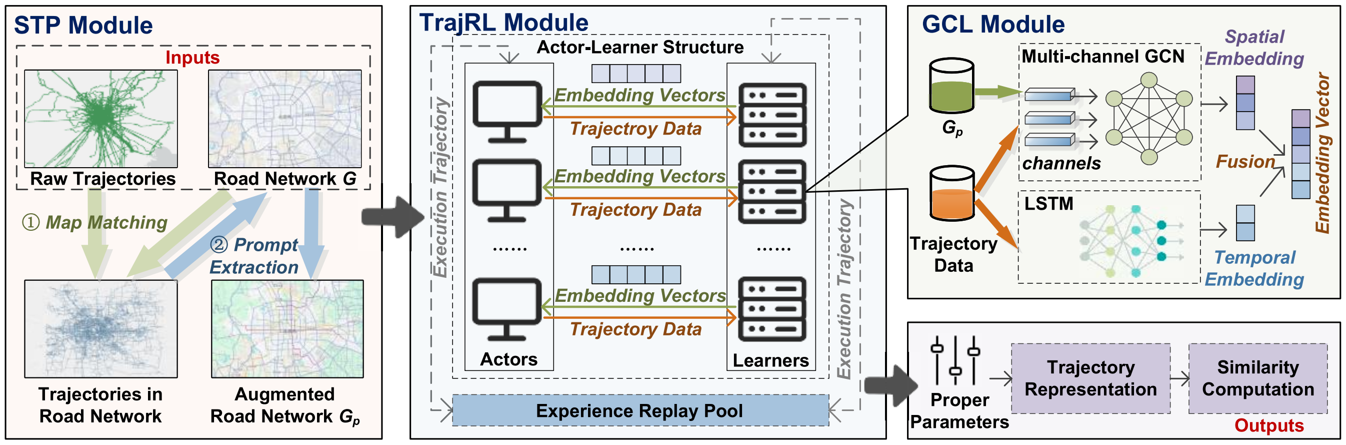


**Figure 3: SimRN Framework**

GCL uses graph contrastive learning to train the neural networks with a small training sample size.

## 3.2 STP Module

In real life, roads have distinct features [5] (i.e., directions and travel times), and the travel time of a particular road varies throughout the day due to changing traffic flows. Existing studies [9, 13, 18, 50] treat all vertices and edges in road networks uniformly, which can negatively impact the accuracy of trajectory similarity computation. To address this, STP has two main goals: (i) road network feature extraction, and (ii) road network augmentation.

***Road Network Feature Extraction.*** Given a road network $G$ and the corresponding road network-constrained trajectories $\mathcal{T}$, we extract spatial features (road directions) and temporal features (travel times). For spatial features, the direction between two vertices $v_i$ and $v_j$ ($v_i, v_j \in G$) $dir_{i,j}$ only has three valid values 0, 1, and 2 (cf. Section 2). For temporal features, to capture fluctuations in travel time due to varying traffic conditions (e.g., rush hours and off-peak periods), we use flexible temporal segmentation. The day is divided into $\epsilon$ equal periods, where $\epsilon$ is an adjustable parameter. This equal segmentation reduces data bias and inconsistency by ensuring all time intervals are equally represented, enhancing the robustness of the analysis. Without such segmentation, certain intervals (e.g., rush hours) may dominate the data, introducing bias, while others may be under-sampled, leading to inconsistencies. Smaller values of $\epsilon$ (e.g., $\epsilon = 4$) produce broader intervals (e.g., morning, afternoon, evening, night), while larger values (e.g., $\epsilon = 24$) yield finer-grained hourly segments. In this paper, we define 9:00–12:00 and 18:00–21:00 as rush hours following existing studies [3, 17]. Accordingly, we set $\epsilon = 8$ in our experiments to divide the day into eight periods (0:00–3:00, 3:00–6:00, 6:00–9:00, 9:00–12:00, 12:00–15:00, 15:00–18:00, 18:00–21:00, and 21:00–24:00), ensuring adequate coverage of rush hours.

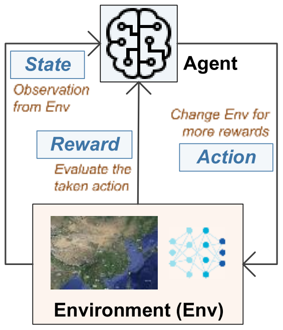
Using the above process, we get the direction set $Dir$ and travel time set $Travel$, respectively (with $dir_{i,j} \in Dir$, $travel_{i,j} \in Travel$). Next, we obtain the spatio-temporal features of road networks, named *prompt information*, by fusing the direction set $Dir$ and travel time set $Travel$, i.e., $Prompt = Dir + Travel$, where $Prompt \in \mathbf{R}^{n_e}$ denotes the prompt information. However, the trajectories in the dataset do not fully cover the entire road network, leaving some roads with unknown directions and travel times. To address this, we impute the missing directions and travel times of unknown roads using nearby roads through the following four steps. (i) We identify the roads $E_{unkn}$ with unknown directions and travel times and initially set both values to 0. (ii) We obtain the set of roads having prompt information, denoted as $E_{kn}$, and compute the average speeds (i.e., vectors with directions) from the lengths and travel times of four time periods of the roads. (iii) In a road network, a vertex is typically connected to multiple edges. This motivates us to use overlapping vertices between $E_{unkn}$ and $E_{kn}$ to fill in the missing road information. We assign the average speeds of vertices in $E_{kn}$ to the corresponding vertices in $E_{unkn}$. With the average speed ($sp$) and the length ($len$), we calculate the travel times and directions for four time periods for each road in $E_{unkn}$ using $Travel = len/sp$. (iv) We update $E_{kn}$ and $E_{unkn}$ and repeat the above three steps until all roads in the network have prompt information.

***Road Network Augmentation.*** We update the edge set $E$ of $G(V, E)$ with the extracted prompt information to create the augmented road network $G_p = (V, E_p)$ by $E_p = E + Prompt$.

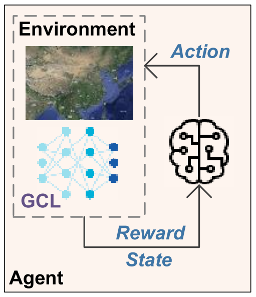
## 3.3 TrajRL Module

Given the training samples, the trajectory representation is influenced by model parameters. Therefore, selecting optimal parameters is crucial for accurately capturing trajectory similarities. This motivates us to transform trajectory similarity learning into a decision-making problem for parameter selection, aimed at finding the best parameters to generate representations that describe trajectory similarities. However, the manual parameter selection used in existing studies [9, 18, 50] is extremely time-consuming and impractical for complex models or large datasets. To address this, we introduce TrajRL, a distributed DRL agent designed to optimize parameter selection for trajectory similarity learning. TrajRL consists of three components: (i) the DRL paradigm, (ii) the actor-learner scheme, and (iii) the experience replay pool.

***DRL Paradigm.*** General DRL models [38, 53, 54] contain two components: agent and environment, which interact to optimize decision-making. The agent performs an “Action” (denoted as $a$) to reach a certain “State” (denoted as $s$), and the environment, composed of neural networks, returns a “Reward” (denoted as $re$) based



(a) General Agent



(b) Agent TrajRL

**Figure 4: Deep reinforcement learning**

on $a$—the better the action, the higher the reward. This trial-and-error process allows the agent to improve its decision-making, as shown in Figure 4(a).

Inspired by general DRL models, we design the first DRL agent, TrajRL, for trajectory similarity learning in road networks, which intelligently and automatically selects optimal parameters. Since our goal is to tune the neural network parameters, we treat the environment as part of the agent, as shown in Figure 4 (b). Thus, in our DRL paradigm, parameter selection with TrajRL becomes a self-learning process. Moreover, we devise other key elements in the DRL paradigm as follows.

- **Action.** The set of parameters $\mathcal{M} = \{m_i | 0 \le i \le 6\}$ used for training the model includes seven key parameters: the number of network layers (denoted as $m_0$), the number of neurons per layer (denoted as $m_1$), learning rate (denoted as $m_2$), training epochs (denoted as $m_3$), batch size (denoted as $m_4$), optimizer (denoted as $m_5$), and activation function (denoted as $m_6$).
- **State.** The state $s$ is the representation generated by training the model using the parameters in action $a$.
- **Reward.** We set the state $s$ as the query trajectories and set the reward $re$ as the average rank of their positive samples in the Top-$k$ similarity search results. The positive sample refer to the most similar trajectory to the query. A reward closer to 1 means the positive sample ranks higher on average, indicating better state and action choices.

Based on the key elements above, we outline the learning procedure of the TrajRL agent. First, TrajRL obtains the state $s_t$ and reward $re_t$ at time $t$, then selects an action $a_t$ based on $s_t$ and $re_t$ according to a given policy. Next, TrajRL performs the action $a_t$ and receives the next state $s_{t+1}$ and reward $re_{t+1}$ from the environment. This process repeats iteratively until the agent reaches an optimal state with sufficient rewards. The process is mathematically represented as follows.

$$a_t = \pi(s_t, re_t), \tag{2}$$
$$s_{t+1} = Network_{\mathcal{M}}(a_t), \tag{3}$$
$$re_{t+1} = Reward(s_{t+1}), \tag{4}$$

where $\pi$ denotes the action selection policy and $Reward(\cdot)$ denotes the reward function. $s_{t+1}$ is determined by the output of the trained networks $Network_{\mathcal{M}}$, based on the parameters $\mathcal{M}$. The initial action $a_0$ is either randomly or manually set.

***Actor-Learner Scheme.*** To improve efficiency in model training, we expand the DRL paradigm to distributed computing and introduce an actor-learner scheme.

As shown in Figure 5, the DRL training is divided into acting and learning. Actors generate rewards from the environment (cf. Eq.4), while learners handle action selection, model training, and state computation via model inference (cf. Eqs.2 and 3). Neural networks in the learners perform gradient calculations and training, while other tasks (e.g., Top-$k$ trajectory similarity search) are handled by the actors. This significantly improves computation efficiency. Moreover, since each learner operates independently, the actor-learner scheme can be easily applied to distributed learning.

We propose a partition strategy for efficient distributed similarity learning, where each learner trains a set of network layers. Instead of evaluating a single action at a time, we perform multiple actions in parallel to optimize decision-making—parameter selection. Specifically, given a set of actions and several distributed computational nodes, trajectory similarity learning in the actor-learner scheme follows three key steps. First, we construct and organize the neural network layers based on action information, such as the number of layers and neurons per layer, identifying any overlapping layers. This allows shared layers across different models to be trained only once, enhancing efficiency.

Next, we apply a dynamic programming algorithm [12] to partition the layers among nodes for training, with the state transition equation defined as follows.

$$dp[i][j] = \min_{0 \le k < i} (dp[k][j\text{-}1] + cost(c[k+1], c[k+2], \ldots, c[i])), \tag{5}$$

where $c[i]$ denotes the computational complexity of the $i^{th}$ network layer (i.e., the number of neurons in that layer), and $cost(c[k+1], c[k+2], \ldots, c[i])$ denotes the load cost of assigning layers $k+1^{th}$ to $i^{th}$ to the $j^{th}$ computational node. Finally, the trained neural network layers are selectively aggregated based on the model structure for each action to generate the final model and trajectory representation, which is then sent to the actors for reward computation.

***Experience Replay Pool.*** The core task of TrajRL is to find an optimal action selection policy that maximizes the reward. To achieve this, we set up a prioritized experience replay pool to store experiences during the agent's learning process. We record the execution trajectory $\tau$ of TrajRL to help update the policy, represented as: $\tau = \langle s_0, a_0, re_0, s_1, a_1, re_1, \ldots \rangle$, where $(s_i, a_i, re_i)$ denotes an experience at time $i$. Next, we prioritize experiences with higher $re$, allowing them to be reused more frequently to improve policy updates. The action selection policy, which determines the preferred action for a given state, is dynamic and evolves as the agent learns, since the priorities of actions change with new experiences.

## 3.4 GCL Module

In real-life scenarios, there may not be enough trajectories for effective model training. Therefore, achieving good performance with small training samples is crucial. We apply contrastive learning methods to address this. While most studies [13, 18, 60] use GCN and RNN models to capture spatial and temporal correlations, general GCNs focus on node features and only handle edge features with one-dimensional attributes. Since $G_p$ produced by the STP module has multi-dimensional edge features (cf. Section 3.2),

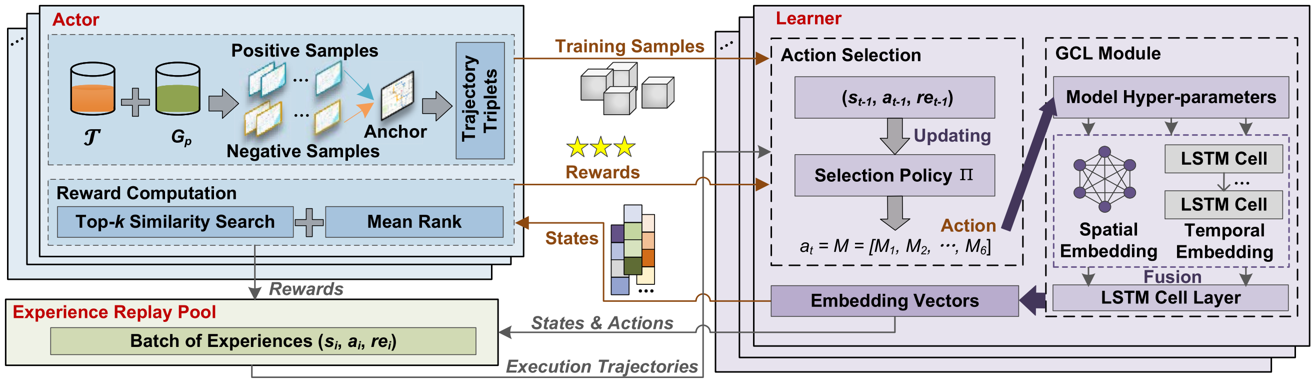


Figure 5: Actor-learner scheme

general GCNs are inadequate. To address this, we design the GCL module with two key goals: (i) expanding the training sample size and (ii) improving model training for similarity representation.

***Training Sample Size Expanding.*** Contrastive learning learns feature representations by comparing similar (positive) and dissimilar (negative) samples. To expand the sample size, we propose a strategy that generates positive and negative samples based on sub-graphs, then applies contrastive learning for trajectory similarity representation. Specifically, we treat each trajectory with road network constraints as a sub-graph of the road network. For each anchor sub-graph $GT$, we sample three sub-sub-graphs (sub-trajectories) with more than half the vertices of $GT$. The sub-sub-graphs of $GT$ are used as positive samples, while three sub-sub-graphs from other anchor sub-graphs are selected as negative samples. For each trajectory $T_a$, we generate three triplets $(T^p, T^a, T^n)$, consisting of a positive sample, an anchor trajectory, and a negative sample, which are used for contrastive learning. This effectively expands the training sample size, while preserving the spatial constraints and temporal dependencies of road network-constrained trajectories.

***Model Training for Trajectory Representation Learning.*** Using the triplets generated by our positive and negative sample strategy, we perform contrastive learning to embed trajectory representation vectors that preserve trajectory similarities. To enhance this, we introduce spatial embedding, temporal embedding, and spatio-temporal fusion for model training.

Spatial Embedding. We design a multi-channel GCN layer to replace the traditional GCN for spatial embedding. Specifically, we use three channels for edge features: length, direction, and travel time. Each channel serves as a constraint matrix to help model learn spatial information, and their outputs are combined to produce the final result. The multi-channel GCN function is defined below.

$$\begin{aligned}\mathcal{X}^{l+1} &= [\mathcal{X}_0^{l+1}, \mathcal{X}_1^{l+1}, ..., \mathcal{X}_q^{l+1}] = \mathrm{GCN}(\cdot) \\ &= Activate\left[\| \left(\tilde{\mathcal{D}}^{-\frac{1}{2}}\tilde{\mathcal{C}}_q^l \mathcal{D}^{-\frac{1}{2}}\mathcal{X}^l \mathcal{W}_q^l\right)\right],\end{aligned} \tag{6}$$

where $\mathcal{X}$ represents the output from each channel of the network layer, $l$ is the layer number, and $q$ is the number of channels. $\mathcal{D}$ is the unified degree matrix, $\mathcal{C}$ is the constraint matrix, and $\mathcal{W}$ is the weight matrix. $\|$ denotes vector concatenation, and $Activate$ is the activation function of the model.

Temporal Embedding. To account for time-periodic features, we use the date2vec [22] model to convert timestamps in year-month-day format into representation vectors ($t_i'$). These vectors are then input into the LSTM model to capture temporal sequence correlations, with the output denoted as $\mathcal{Y}$. The process is formulated as follows.

$$t_i' = \mathrm{date2vec}(timestamp_i), \tag{7}$$

$$\mathcal{Y}_i = \mathrm{LSTM}\left(t_i', \mathcal{Y}_{i-1}, i_i, f_i, o_i, m_i\right), \tag{8}$$

where $i_i$, $f_i$, $o_i$, and $m_i$ represent the input gate, forget gate, output gate, and memory cell, respectively. Moreover, $\mathcal{Y}_{i+1}$ is the state of the $(i+1)$-th hidden layer, updated based on $\mathcal{Y}_i$.

Spatio-Temporal Fusion. We use the spatial embedding $\mathcal{X}$ and temporal embedding $\mathcal{Y}$ for feature fusion. Following the spatio-temporal fusion method [13], we apply an LSTM layer to combine these spatial and temporal features. This captures spatio-temporal sequence correlations and embeds the trajectory similarity representation vectors, as formulated below.

$$r = \mathrm{LSTM}\left(\mathcal{X}, \mathcal{Y}\right) \tag{9}$$

Note that in Eq. 3, $Network$ refers to the models for spatio-temporal embedding, and $\mathcal{M}$ denotes their respective parameters.

Model Training Process. We train the model by the following steps. First, we generate triplets $(T^p, T^a, T^n)$ and perform spatial embedding, temporal embedding, and spatio-temporal fusion to obtain their representation vectors $(T^p, T^a, T^n) \rightarrow (r^p, r^a, r^n)$. Next, we train the model using the infoNCE [43] loss, as shown below.

$$\mathcal{L}_{infoNCE} = -\frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{\exp\left(\frac{r_i^a \cdot r_i^p}{\epsilon}\right)}{\sum_{j=1}^{N}\exp\left(\frac{r_i^a \cdot r_j^n}{\epsilon}\right)}\right), \tag{10}$$

where $N$ is the number of training samples, and $\epsilon$ is a hyperparameter that controls the convergence speed.

## 3.5 Novelty

We summarize the key contributions of SimRN to highlight its technical novelty. In addition, we provide a theoretical analysis of how each component—STP, TrajRL, and GCL—contributes to the overall performance.

### 3.5.1 *Technical Novelty.*

- **Prompt extraction.** Unlike existing methods that ignore the essential features of road network, we propose a prompt extraction method by designing a time segmentation strategy, which augments the road network information for similarity learning.

- **Reinforcement learning agent.** We innovatively combine trajectory similarity learning with decision-making problem to support automatically parameter selection, addressing the deficiency of manual tuning in existing methods. Based on this, we propose the first reinforcement learning agent for trajectory similarity learning by developing an effective actor-learner scheme.
- **Experience replay.** We introduce an effective experience replay pool for the designed agent to provide experiences with higher priority for efficient parameter selection.
- **Sub-graph augmentation.** To deal with insufficient training samples, we devise a sub-graph-based augmentation method to generate both negative and positive samples. The augmentation method can generate large amounts of positive and negative samples for graph contrastive learning in an unsupervised way. This reduces reliance on large training datasets while significantly lowering the computational cost of similarity matrix computation and sample selection of existing supervised methods, which would otherwise require quadratic time complexity.
- **Multi-channel GCN layer.** Rather than utilizing the single-channel GCN model that can only handle limited features, we propose a multi-channel GCN layer to extract prompt information from road networks, capturing comprehensive and essential data while filtering out noise and irrelevant information. This enhances data representation, simplifies model training, and improves accuracy in similarity learning.

#### 3.5.2 *Theoretical Analysis.*

***STP component.*** Existing methods ignore the extraction of essential information in road networks, which train the model by directly using the raw road network data containing noise. Assuming that $B$ is the raw data, which consists of essential information (denoted as $I$) and irrelevant noise (denoted as $C$), i.e., $B = I + C$. As a result, the noise $C$ hinders the accuracy of trajectory similarity learning.

STP component aims to extract the essential spatio-temporal features in road network as prompt information. The feature extraction function $D(\cdot)$ is presented as $D(B) = D(I) + D(C)$. Since noise $C$ is random and independent of $I$, its expectation is 0 [8] (i.e., $\mathbb{E}|D(C)| = 0, D(B) \approx D(I)$ where $\mathbb{E}$ denotes the expectation). Thus, the noise can be eliminated through feature extraction, and $D(B)$ is compact and discriminative for spatio-temporal properties [55].

$$|f_{true}(T_1, T_2) - \mathcal{F}_1| \leq |f_{true}(T_1, T_2) - \mathcal{F}_2|, \tag{11}$$

where $\mathcal{F}_1 = \mathcal{F}(r_1(D(B)), r_2(D(B)))$, $\mathcal{F}_2 = \mathcal{F}(r_1(B), r_2(B))$. $f_{true}(T_1, T_2)$ denotes the ground truth distance between two road network constrained trajectories $T_1$ and $T_2$. $r_1$ and $r_2$ denote the representation vectors of $T_1$ and $T_2$. $\mathcal{F}(\cdot, \cdot)$ is the distance measure between representation vectors. Lower values of $f_{true}$ and $\mathcal{F}$ indicate higher similarities. As a result, the extracted features can help to enhance the similarity representation [60], which improves accuracy.

***TrajRL component.*** Traditional learning-based methods manually tune the parameters for model training, i.e., $\theta^* = \arg\min_\theta \mathcal{L}(\theta)$ where $\theta$ denotes the model parameters, and $\theta^*$ denotes the tuned optimal parameters. This static method only test limited number of parameter combinations, leading to local optimal solutions and missed opportunities for global optimal tuning.

TrajRL component incorporates a deep reinforcement learning (DRL) agent, which is designed to optimize model parameters for long-term performance. The agent dynamically interacts with similarity computation results during training and uses a deep neural network to approximate the action selection policy $\pi$. This is due to General Approximation Theorem [15], that deep neural networks can approximate any continuous function including the action selection policy. This allows the agent to iteratively updating parameters using rewards, ultimately converging to identify the most optimal parameters for accurate similarity learning.

$$\nabla_\gamma J(\gamma) = \mathbb{E}_\pi \left[\nabla_\gamma \log \pi \left(a_t, s_t; \gamma\right) \cdot re_t\right], \gamma \leftarrow \gamma + \alpha \nabla_\gamma J(\gamma), \tag{12}$$

$$\theta^* = a_t = \pi(s_t, re_t; \gamma), \tag{13}$$

where $J(\gamma)$ is the policy optimization function to find the proper updating policy $\pi(a, s; \gamma)$. $re$ denotes the reward (i.e., similarity computation results). $\gamma$ is the selection policy updating parameter. $\alpha$ is the learning rate of policy updating. $a$ represents the action (i.e., selected parameters $\theta$) of agent. $s$ is the state (i.e., the model trained with selected parameters) when executing $a$.

Moreover, the process of model training is very inefficient, where the whole model training time is denoted as: $T_{tra} = T_{re} + T_\pi + T_{Network}$ where $T_{re}$ denotes the time of computing rewards, which is very time-consuming with quadric time complexity. $T_\pi$ denotes the time of updating policy and selecting parameters (cf. Eqs. 12 and 13). $T_{Network}$ represents the model training time using selected parameters.

To address this, TrajRL component employs an actor-learner scheme, enabling computation parallelization. Specifically, we assign the time-consuming rewards computation to actors, while perform parameter updating and model training on learners. Next, we utilize distributed techniques to enable computation parallelization, reducing the whole training time:

$$T^*_{tra} = \frac{T_{re}}{NA} + \frac{T_\pi + T_{Network}}{NL} + c < T_{tra}, \tag{14}$$

where $NA$ and $NL$ denote the number of actors and learners, respectively. $c$ denotes the communication time, which can be safely ignored (cf Section 4.3.2 of the revised paper).

Further, we introduce an experience replay pool, which contains various experiences with priority for updating policy. Assuming that there are $N^*$ experiences in the replay pool, where $M^*$ of them are with high priority (i.e., $M^* \ll N^*$). The probability of choosing a proper experience for policy updating is denotes as below.

$$P_{priority}(i) = \frac{p_i}{\sum_{k=1}^{N^*} p_k} \approx \frac{p_i}{M^* \cdot p_i} = \frac{1}{M^*} \gg \frac{1}{N^*}, \tag{15}$$

where $p_i$ denotes the priority of the $i$-th experience. Note that, the experience replay pool increases the probability of finding the useful experiences for policy updating, improving the efficiency.

***GCL component.*** Traditional methods fail to deal with essential features, as they only use single-channel GCN layer for feature extraction: $H = Activate(AXW)$, where $A$, $X$, and $W$ are adjacent matrix, feature matrix, and weight matrix of the graph, respectively. $Activate$ is the activation function. GCL component integrates a multi-channel GCN layer to extract prompt information from road networks as: $H^* = Activate\left(\sum_{k=1}^{K} A_k X W_k\right)$, where $K$ denotes the number of channels. As a result, multi-channel GCN layer can capture more features, enhancing the capability of model representation and improving accuracy.

Table 2: Statistics of datasets

| Dataset | RN | Sampling Interval | Duration | Avg. Length |
|---|---|---|---|---|
| **T-Drive** | Beijing | 177 seconds | 2 weeks | 150 |
| **Porto** | Porto | 15 seconds | 7 weeks | 60 |
| **Rome** | Rome | 7 seconds | 4 weeks | 15 |
| **SF** | San Francisco | 10 seconds | 3 weeks | 10 |

In addition, GCL also incorporates a sub-graph-based augmentation method to generate large amounts of positive and negative samples for graph contrastive learning.

$$|\mathcal{T}_{new}| = |\mathcal{T}_{original}| + |\mathcal{T}^{+}| + |\mathcal{T}^{-}| \gg |\mathcal{T}_{original}| \quad (16)$$

Here, $\mathcal{T}$ denotes the trajectory dataset. According to Eq 16, GCL expands the training dataset and supports robust model training, even with limited original samples.

## 4 EXPERIMENT

In this section, we evaluate SimRN through a series of experiments on two real-life datasets, aiming to answer the following questions:

- **Q1**: How does SimRN's accuracy in trajectory similarity computation compare to other learning-based methods?
- **Q2**: How efficient and scalable is SimRN?
- **Q3**: How does SimRN perform when training with different sample sizes?
- **Q4**: How do the individual modules impact trajectory similarity computation and model training?

### 4.1 Experimental Settings

*4.1.1* ***Datasets.*** We evaluate SimRN on four real-world datasets: T-Drive, Porto, Rome, and SF.

- **T-Drive**[1] contains 1.5 million GPS points collected from 10,357 taxis in Beijing, from Feb. 2 to Feb. 8, 2008.
- **Porto**[2] contains 1.7 million GPS points collected from 442 taxis in Porto, Portugal, from Aug. 2011 to Apr. 2012.
- **Rome**[3] contains 1.7 million GPS points collected from 320 taxis in Rome, Italy, from Feb. 1 to Mar. 2, 2014.
- **SF**[4] contains 1.0 million GPS points collected from 500 taxis in San Francisco, USA, from May 17 to Jun. 10, 2008.

Detailed information of both datasets is summarized in Table 2, where "RN" is the city corresponding to each dataset. Note that the trajectories from both datasets are collected as raw trajectories. We remove those with fewer than 5 GPS points and randomly select 10,000 trajectories from each dataset for evaluation. We obtain the road networks of Beijing, Porto, Rome and San Francisco from OpenStreetMap [33].Map-matching is an essential step to align raw trajectories with road networks and convert them into road network-constrained trajectories [40]. We employ a KD-Tree-based map-matching algorithm [4] for its low time complexity of $O(|V| \log |V|)$, where $|V|$ is the total number of vertices in the road network. Each dataset is processed through map-matching only once [9, 13, 18, 60]. Thus, these preprocessing overheads (about 2316s for T-Drive, 2058s for Porto, 750s for Rome, and 520s for SF) are not included in the following experimental reports.

[1]https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/
[2]https://archive.ics.uci.edu/dataset/339/
[3]https://ieee-dataport.org/open-access/crawdad-romataxi
[4]https://ieee-dataport.org/open-access/crawdad-epflmobility

*4.1.2* ***Baselines.*** We compare SimRN with three state-of-the-art learning-based methods: GTS, ST2Vec, and GRLSTM, and six state-of-the-art non-learning-based methods: NetDTW, NetEDR, NetERP, TP, LORS, and LCRS.

- **GTS** [18] employs a GCN based trajectory-aware random walk scheme to learn the representation of each POI.
- **ST2Vec** [13] captures fine-grained spatial and temporal correlations between trajectory pairs using GCN and LSTM models.
- **GRLSTM** [60] builds a knowledge graph to capture the correlations between GPS points and road network for representation.
- **NetDTW** [48] finds the optimal alignment between two trajectories by minimizing the distance between mapped points, while respecting the spatial constraints of the road network.
- **NetEDR** [24] measures trajectory similarity by counting the minimal number of edits required to match one trajectory to another while considering spatial constraints.
- **NetERP** [24] calculates the minimal transformation cost (i.e., insertions, deletions and substitutions) between two trajectories.
- **TP** [39] consider both spatial and temporal correlations to capture the pattern changes and finer spatio-temporal relationships.
- **LORS** [45] computes trajectory similarity by identifying and comparing overlapping edges.
- **LCRS** [56] identifies the longest common road segment between two trajectories consisted of road segments.

All baselines are designed for trajectory similarity computation in road networks. Learning-based baselines are trained using the hyperparameters specified in their original papers. The goal of them is to approximate non-learning-based methods for computing trajectory similarity. For NetEDR, the similarity threshold is set to 1 km. For NetERP, the gap point is set to the centroid of the bounding rectangle enclosing the dataset.

*4.1.3* ***Evaluation Metrics.*** We compare SimRN with three state-of-the-art baselines in terms of accuracy, efficiency, scalability, and model generality. For accuracy and model generality, we use two metrics: *HR@k* and *Rk@t* [30]. *HR@k* (hitting ratio of Top-$k$ similarity search) measures the overlap between the Top-$k$ results and the ground truth, while *Rk@t* (Top-$t$ recall for the Top-$k$ ground truth) is the fraction of the Top-$k$ ground truth included in the Top-$t$ results. Higher values of *HR@k* and *Rk@t* (closer to 1) indicate better performance. In our experiments, we evaluate *HR@10*, *HR@50*, and *R10@50*. The ground truth is derived using a non-learning-based similarity measure $f(\cdot,\cdot)$ to compute the Top-10 and Top-50 results for a query trajectory *T*. We consider six non-learning-based measures: NetDTW, NetEDR, NetERP, TP, LORS, and LCRS. For efficiency and scalability, we measure the average training time per epoch ($T_{tra}$) and the similarity computation time ($T_{com}$).

*4.1.4* ***Experimental Settings.*** The initial setting of $\mathcal{M}$ (i.e., initial parameters) is based on two key features of datasets: data distribution and sampling interval.

- **Data distribution.** For evenly distributed datasets, it is able for the model to update gradients and learn features stably with few overfitting and rapid convergence. Thus, a higher number of network layers, neurons per layer and learning rate should be set for capturing deeper features. Moreover, optimizers (e.g., SGD [16]) and activation functions (e.g., ReLU [14]) that ensure

Table 3: Accuracy evaluation, with bold numbers highlighting the highest accuracy

| Measures | Methods | T-Drive | | | Porto | | | Rome | | | SF | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *HR*@10 | *HR*@50 | *R*10@50 | *HR*@10 | *HR*@50 | *R*10@50 | *HR*@10 | *HR*@50 | *R*10@50 | *HR*@10 | *HR*@50 | *R*10@50 |
| NetDTW | GTS | 0.3814 | 0.5609 | 0.7480 | 0.4532 | 0.5726 | 0.8069 | 0.4096 | 0.5574 | 0.6930 | 0.3991 | 0.5393 | 0.7131 |
| | ST2Vec | 0.4051 | 0.5767 | 0.7783 | 0.3633 | 0.5983 | 0.7540 | 0.3403 | 0.5478 | 0.7057 | 0.2143 | 0.5512 | 0.6739 |
| | GRLSTM | 0.2715 | 0.4379 | 0.7541 | 0.3197 | 0.5395 | 0.7985 | 0.2830 | 0.3791 | 0.6160 | 0.3846 | 0.5749 | 0.7241 |
| | SimRN | **0.4841** | **0.6195** | **0.8220** | **0.5457** | **0.6308** | **0.8654** | **0.4607** | **0.6733** | **0.8215** | **0.4321** | **0.6775** | **0.8544** |
| NetEDR | GTS | 0.5267 | 0.6656 | 0.7838 | 0.4018 | 0.5765 | 0.7547 | 0.3660 | 0.5373 | 0.7830 | 0.3423 | 0.5891 | 0.7550 |
| | ST2Vec | 0.2346 | 0.2405 | 0.2463 | 0.2502 | 0.25982 | 0.2765 | 0.2772 | 0.3142 | 0.3715 | 0.3216 | 0.3502 | 0.4191 |
| | GRLSTM | 0.5301 | 0.6117 | 0.7479 | 0.3875 | 0.5168 | 0.8050 | 0.3714 | 0.5901 | 0.7713 | 0.3180 | 0.4707 | 0.7357 |
| | SimRN | **0.5792** | **0.7139** | **0.8489** | **0.4933** | **0.6574** | **0.8148** | **0.5614** | **0.7361** | **0.8815** | **0.5426** | **0.6911** | **0.7567** |
| NetERP | GTS | 0.3621 | 0.4970 | 0.7416 | 0.4604 | 0.6665 | 0.8165 | 0.3210 | 0.4814 | 0.7096 | 0.311 | 0.5172 | 0.7303 |
| | ST2Vec | 0.3465 | 0.5071 | 0.7589 | 0.3605 | 0.5309 | 0.7190 | 0.3286 | 0.5789 | 0.7185 | 0.3141 | 0.6051 | 0.7076 |
| | GRLSTM | 0.3738 | 0.5386 | 0.7762 | 0.3294 | 0.5066 | 0.8130 | 0.3593 | 0.4911 | 0.6726 | 0.4057 | 0.5697 | 0.7627 |
| | SimRN | **0.5421** | **0.6722** | **0.8144** | **0.5815** | **0.7354** | **0.8841** | **0.5324** | **0.7269** | **0.8258** | **0.5472** | **0.7122** | **0.8679** |
| TP | GTS | 0.2736 | 0.4524 | 0.7402 | 0.2933 | 0.4452 | 0.6450 | 0.3774 | 0.5251 | 0.6997 | 0.2929 | 0.4381 | 0.6885 |
| | ST2Vec | 0.4417 | 0.6227 | 0.8029 | 0.3754 | 0.5969 | 0.7683 | 0.4268 | 0.6492 | 0.7799 | 0.3554 | 0.6372 | 0.7037 |
| | GRLSTM | 0.3279 | 0.4013 | 0.5035 | 0.3824 | 0.4862 | 0.6343 | 0.3389 | 0.5013 | 0.4983 | 0.3243 | 0.5556 | 0.5943 |
| | SimRN | **0.5097** | **0.6992** | **0.8528** | **0.4417** | **0.6661** | **0.8266** | **0.5850** | **0.6687** | **0.8514** | **0.4982** | **0.6595** | **0.7731** |
| LORS | GTS | 0.4190 | 0.5824 | 0.7590 | 0.4452 | 0.5138 | 0.7596 | 0.4439 | 0.5149 | 0.7267 | 0.3934 | 0.5270 | 0.7264 |
| | ST2Vec | 0.1326 | 0.3234 | 0.4139 | 0.1613 | 0.1876 | 0.2099 | 0.1589 | 0.2834 | 0.3189 | 0.2321 | 0.3283 | 0.4497 |
| | GRLSTM | 0.4323 | 0.6714 | 0.7809 | 0.3370 | 0.5105 | 0.7639 | 0.3533 | 0.5481 | 0.7431 | 0.3546 | 0.5819 | 0.7329 |
| | SimRN | **0.5213** | **0.7570** | **0.8422** | **0.5348** | **0.6239** | **0.8873** | **0.4794** | **0.6029** | **0.7459** | **0.4350** | **0.6241** | **0.7913** |
| LCRS | GTS | 0.3169 | 0.4926 | 0.7301 | 0.3732 | 0.5454 | 0.7766 | 0.3382 | 0.4241 | 0.6619 | 0.3194 | 0.4847 | 0.6973 |
| | ST2Vec | 0.1499 | 0.1814 | 0.2332 | 0.4004 | 0.4021 | 0.7999 | 0.1710 | 0.3402 | 0.4502 | 0.1877 | 0.3618 | 0.4223 |
| | GRLSTM | 0.4081 | 0.6424 | 0.7197 | 0.3800 | 0.6546 | 0.7967 | 0.3170 | 0.4540 | 0.6787 | 0.2439 | 0.4096 | 0.6580 |
| | SimRN | **0.5254** | **0.7390** | **0.8274** | **0.5478** | **0.6847** | **0.9303** | **0.5371** | **0.6708** | **0.8584** | **0.4854** | **0.5347** | **0.7566** |

Table 4: Overall training time of learning-based methods

| Methods | T-Drive | Porto | Rome | SF |
|---|---|---|---|---|
| GTS | 1782.00s | 856.10s | 882.40s | 842.10s |
| ST2Vec | 2418.32s | 5157.10s | 2073.34s | 1010.90s |
| GRLSTM | 4580.16s | 2317.24s | 2263.98s | 1387.75s |
| SimRN | 1122.25s | 744.62s | 751.94s | 626.82s |

Table 5: Efficiency evaluation of non-learning-based methods

| $T_{com}$ | NetDTW | NetEDR | NetERP | TP | LORS | LCRS |
|---|---|---|---|---|---|---|
| T-Drive | 184.29s | 170.31s | 160.21s | 155.58s | 192.87s | 182.74s |
| Porto | 161.46s | 198.51s | 131.37s | 133.91s | 125.31s | 187.40s |
| Rome | 179.54s | 156.48s | 143.61s | 142.64s | 134.34s | 161.28s |
| SF | 191.51s | 181.49s | 164.10s | 167.07s | 179.27s | 193.89s |

quick model convergence and consistent updates without introducing significant oscillations or deviation are best suited for stable gradients. On the contrary, for unevenly distributed datasets, the gradients of the model vary greatly during training process, which can easily result in overfitting and local optimal solutions. Thus, it is necessary to keep strong generalization ability for model, which requires less network layers and neurons per layer, and lower learning rate. In addition, optimizers (e.g., Adam [23]) and activation functions (e.g., LeakyReLU [49]) that stabilize gradient updates and mitigate vanishing or exploding gradients are most suitable for handling varying gradients.

- **Sampling interval.** For data with low sampling interval which contains more sufficient information, a higher number of network layers, training epochs, and batch size can help to deal with large amount of information data. On the contrary, for data with high sampling interval, lower parameters are required. Note that, optimizers and activation functions primarily affect gradient stability and convergence speed, which depend on data distribution rather than sampling intervals.

Based on the above guidelines, for datasets with a uniform distribution and a low sampling interval (less than 60 seconds) such as Porto, Rome, and SF, we set the initial parameters to $\mathcal{M}_0$={3, 128, 0.001, 150, 128, *SGD*, *ReLU*}. For datasets with an uneven distribution and a high sampling interval, such as T-Drive, we set the initial parameters to $\mathcal{M}_0$={1, 64, 0.0001, 120, 64, *Adam*, *LReLU*}.

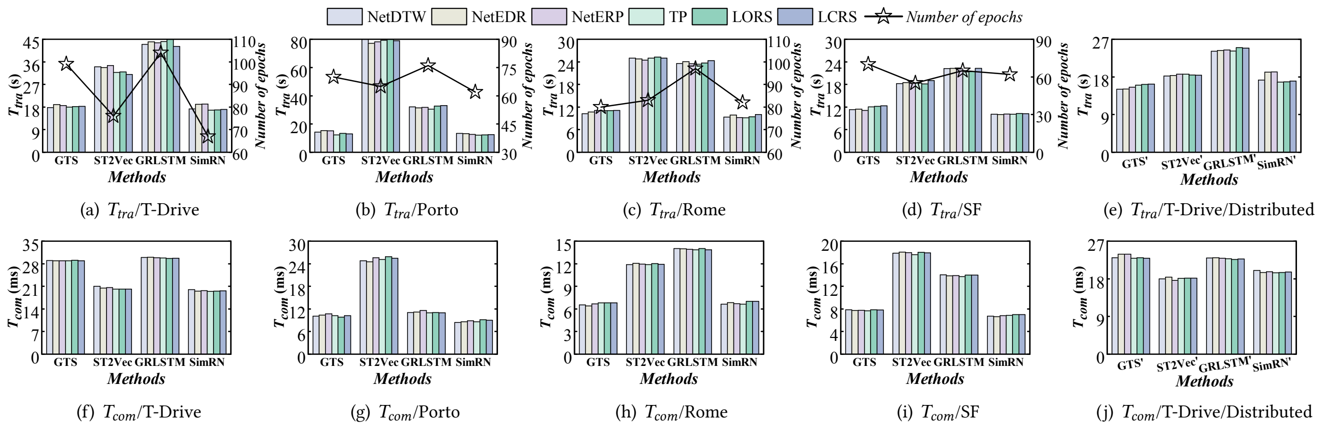
*4.1.5* ***Implementation Details.*** For non-learning-based baselines, we conduct experiments on an 8-core Intel Xeon E5-2640 v4 @ 2.40GHz CPU. For learning-based methods, we split each dataset into training, validation, and test sets in a 3:1:6 ratio, and run the experiments on a 4-node cluster, each node equipped with a GeForce RTX 3090 GPU, 2.40GHz processor, and 24GB RAM.

It is worth mentioning that, SimRN is designed for distributed environments. First, it implements a distributed training scheme that scales across multiple machines to handle large-scale data, going beyond parallel computation limited to a single machine. This is achieved using an actor-learner architecture and a dynamic model partitioning strategy, ensuring efficient and scalable training. Second, distributed communication primitives (e.g., all-reduce) are integrated to facilitate node-to-node communication [32]. In the experiments, we simulate a 4-node cluster on a single machine equipped with four GPUs. Following [2, 35, 59], each GPU is treated as an independent node.

## 4.2 Accuracy Evaluation (Q1)

We evaluate the accuracy performance of four learning-based methods. The results are reported in Table 3.

(i) Learning-based methods vary in their ability to approximate different non-learning-based similarity measures. Specifically, ST2Vec outperforms GTS and GRLSTM when approximating NetDTW, NetERP, and TP distances but performs poorly with NetEDR, LORS,



(a) $T_{tra}$/T-Drive (b) $T_{tra}$/Porto (c) $T_{tra}$/Rome (d) $T_{tra}$/SF (e) $T_{tra}$/T-Drive/Distributed

(f) $T_{com}$/T-Drive (g) $T_{com}$/Porto (h) $T_{com}$/Rome (i) $T_{com}$/SF (j) $T_{com}$/T-Drive/Distributed

**Figure 6: Efficiency evaluation of learning-based methods**

**Table 6: Scalability evaluation on # of distributed nodes**

| # of Distributed Nodes | T-Drive | | | Porto | | | Rome | | | SF | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $T_{tra}$ | $T_{com}$ | $T_c$ | $T_{tra}$ | $T_{com}$ | $T_c$ | $T_{tra}$ | $T_{com}$ | $T_c$ | $T_{tra}$ | $T_{com}$ | $T_c$ |
| 1 | 37.00s | 23.75ms | / | 25.10s | 14.70ms | / | 31.21s | 12.92ms | / | 38.87s | 13.72ms | / |
| 2 | 25.18s | 22.47ms | 1.60s | 19.88s | 10.12ms | 0.82s | 18.39s | 11.03ms | 0.22 | 18.87s | 10.72ms | 0.17s |
| 4 | 16.75s | 19.47ms | 2.27s | 12.01s | 8.60ms | 1.22s | 9.17s | 6.65ms | 0.34s | 10.11s | 6.85ms | 0.28s |

and LCRS distances. This is because GTS and GRLSTM are specifically proposed for spatial constraints of road networks, whereas ST2Vec considers both spatial and temporal correlations. However, NetEDR, LORS and LCRS distances are much more dependent on road network information, making the temporal information ignored by ST2Vec when measuring the similarity.

(ii) SimRN achieves the best accuracy on all of the six non-learning-based methods. This is because, it not only captures spatio-temporal features of trajectories, but also extracts prompt information: directions and travel times of road networks, which enhances the model's ability to better capture trajectory similarities. Moreover, the proposed TrajRL module automatically selects optimal parameters for model training, resulting in much better trajectory representation vectors than all baselines.

## 4.3 Efficiency and Scalability Evaluation (Q2)

We first study the efficiency on both non-learning-based and learning-based methods. Next, we evaluate their scalability by varying data cardinality and the number of distributed nodes.

*4.3.1* ***Efficiency Evaluation.*** First, we evaluate the abilities of four learning-based methods to approximate the six non-learning-based methods. The results are reported in Figure 6. Specifically, Figures 6(a)–6(e) report the training time per epoch ($T_{tra}$) and the average number of epochs of each model. Figures 6(f)–6(j) present the computation time ($T_{com}$) of performing Top-50 similarity search for 3,000 trajectories on four datasets. Table 4 shows the overall training time of learning-based methods. Table 5 reports the computation time ($T_{com}$) of six non-learning-based methods. We further evaluate the impact of utilizing distributed techniques in SimRN on model efficiency. Since no existing parallel implementation for trajectory similarity learning is available, to enable a more comprehensive comparison, we have manually extended the learning-based methods GTS, ST2Vec, and GRLSTM to distributed methods (i.e., GTS′, ST2Vec′, and GRLSTM′). These methods include a model training process that benefits from distributed training by parallelizing computations across nodes. In contrast, non-learning-based methods lack a training process, making distributed implementation less relevant. The results on the T-Drive dataset are reported in Figures 6(e) and 6(j). The observations are as below.

(i) The computation time $T_{com}$ of learning-based methods is much less than non-learning-based methods. This is because, the time complexity of non-learning-based methods are all quadratic–$O(n^2)$ (where $n$ is the number of GPS points in the trajectories), while the vectors computation of learning-based methods is Euclidean distance, reducing the time complexity of learning-based methods to linear–$O(d)$ (where $d$ refers to the dimensionality of the representation vectors of trajectories).

(ii) The training procedure of learning-based methods is very time-consuming (i.e., $T_{tra}$ > 15s). This is because, the features capturing process during model training needs much time cost.

(iii) SimRN achieves the lowest training time per epoch ($T_{tra}$), computation time ($T_{com}$), and overall training time compared to learning-based baselines. This is because, first, SimRN leverages distributed techniques for parallel training, resulting in less $T_{tra}$ and $T_{com}$. Second, SimRN incorporates an experience replay pool with prioritization, enabling to efficient select better actions.

(iv) As shown in Figures 6(e) and 6(j), the parallelized versions, GTS′, ST2Vec′, and GRLSTM′, demonstrate improved efficiency compared to their standalone counterparts due to the benefits of distributed computing. Notably, SimRN consistently outperforms all baselines in both standalone and distributed settings. This is attributed to SimRN's support for dynamic partitioning during distributed model training and similarity computation, making it highly adaptable to varying trajectories and model parameters.

*4.3.2* ***Scalability Evaluation.*** Figure 7 reports the computation time $T_{com}$ of the six non-learning-based and four learning-based methods on four datasets when varying the cardinality of trajectories (i.e., 20%, 40%, 60%, 80%, and 100%). Table 6 presents the training

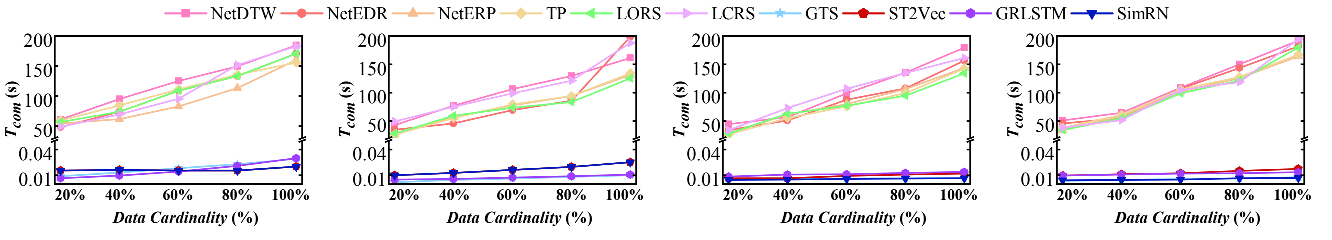

(a) T-Drive (b) Porto (c) Rome (d) SF

**Figure 7: Scalability evaluation on varying data cardinality**

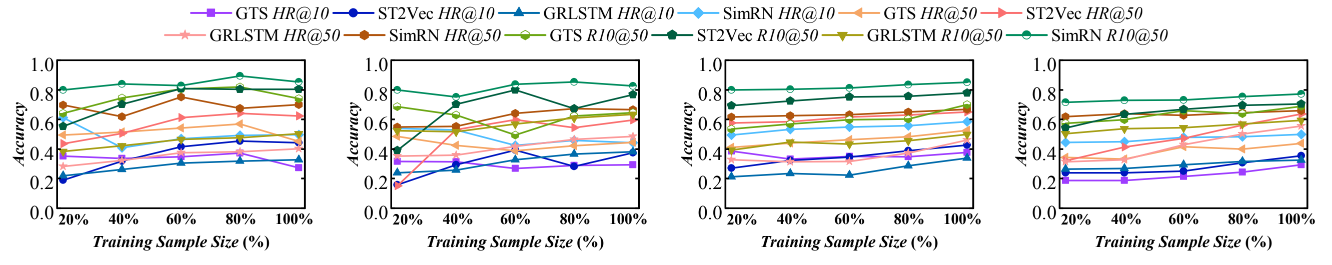

(a) T-Drive (b) Porto (c) Rome (d) SF

**Figure 8: Model generality evaluation on varying training samples size**

**Table 7: Model generality evaluation on incremental training**

| Methods | T-Drive | | | SF | | |
|---|---|---|---|---|---|---|
| | *HR@10* | *HR@50* | *R10@50* | *HR@10* | *HR@50* | *R10@50* |
| SimRN | 0.4305 | 0.6731 | 0.8126 | 0.4817 | 0.6506 | 0.7654 |
| GRLSTM* | 0.3341 | 0.3928 | 0.5195 | 0.3684 | 0.5516 | 0.6143 |
| SimRN* | 0.4519 | 0.7020 | 0.8442 | 0.5209 | 0.7393 | 0.8708 |

time $T_{tra}$, computation time $T_{com}$ and communication overhead $T_c$ per epoch of SimRN when varying the number of distributed nodes (i.e., 1, 2, and 4). "/ " indicates no communication overhead. Due to limited space, we mainly report the results on TP distance. The observations are as below.

(i) As shown in Figure 7, computation times for all methods increase as the cardinality of trajectories grows. Non-learning-based methods show a sharp increase due to the time required for point-matching distance calculations, whereas learning-based methods grow more slowly because they use pre-trained models for Top-$k$ similarity search with linear time complexity.

(ii) As depicted in Table 6, for SimRN, its scalability performance improves with the growth of the number of distributed nodes, for the reason that more distributed nodes provide more computing resources, reducing computational load and improving efficiency.

(iii) As shown in Table 6, during distributed training, $T_c$ is much smaller than the training time $T_{tra}$, which can be considered negligible. This is because, we use a 4-node cluster (i.e., 4 GPUs on one machine) for distributed learning, where communication overhead minimized by the high-speed bus [32]. This setup is significantly more efficient than network-based communication. As a result, the communication overhead, which is orders of magnitude smaller than the epoch training time, can be safely ignored.

## 4.4 Model Generality Evaluation (Q3)

We evaluate the model generality of each learning-based method by varying the size of training samples. For the test sets of four datasets, we apply the models trained on different training sample sizes (i.e., 20%, 40%, 60%, 80%, and 100% of the training sets), and report the accuracy by performing Top-10 and Top-50 similarity search using the same steps in Section 4.2. Here, we only use the ground-truth of TP distance due to the limited space. Figure 8 depicts the results on the four datasets. The observations are as below.

**Table 8: Ablation study on the T-Drive dataset**

| Components | *HR*@10 | *HR*@50 | *R*10@50 | $T_{tra}$ (s) | $T_{com}$ (ms) |
|---|---|---|---|---|---|
| GCL w/o MC-GCL | 0.3811 | 0.5619 | 0.8002 | 35.24 | 21.98 |
| GCL | 0.3977 | 0.5841 | 0.7930 | 35.54 | 22.60 |
| GCL+STP | 0.4432 | 0.6120 | 0.8202 | 35.70 | 22.95 |
| GCL+STP+DRL | 0.5088 | 0.6893 | 0.8499 | 37.00 | 23.75 |
| GCL+STP+TrajRL | 0.5097 | 0.6992 | 0.8528 | 16.75 | 19.47 |

(i)SimRN The *HR@10* and *HR@50* of ST2Vec, GRLSTM, and generally improve as the training sample size increases, as the models capture more comprehensive information from additional samples. However, in Figure 8, SimRN shows accuracy deterioration. This occurs because the model adjusts its parameters to adapt to the growing data, which can sometimes lead to a very large number of network layers and neurons per layer. In such cases, the model captures more irrelevant information and noise, which may result in overfitting and cause accuracy to decline.

(ii) SimRN achieves the highest accuracy across all training sample sizes, particularly with smaller sizes (20% and 40%). This is because its positive and negative sample generation strategy provides more samples for contrastive learning than other methods. As a result, SimRN maintains strong and stable accuracy across various sample sizes, demonstrating excellent model generality.

Further, we explore the generality performance of our SimRN when dealing with incoming new data. We compare SimRN with GRLSTM, the state-of-the-art method that outperforms all baselines on T-Drive and SF datasets. Table 7 reports the accuracy. SimRN refers to the original method without model updates for new data, while GRLSTM* and SimRN* are the incrementally updated versions of GRLSTM and SimRN. We observe that SimRN* can incrementally update the model with new data while maintaining better stability and accuracy. This is because, unlike GRLSTM*, our method efficiently selects parameters that adapt to changing data by utilizing experiences stored in the experience replay pool, enabling effective gradient computation for updates. Moreover, SimRN achieves strong performance even without incremental updates. This is due to the GCL module, which expands the training sample size using the original data, enhancing generalization and adaptability for similarity computation with new data.

### 4.5 Ablation Study (Q4)

We conduct ablation studies by studying the effectiveness and efficiency (i.e., *HR@10, HR@50, R10@50*, $T_{tra}$, and $T_{com}$) of four key components (i.e., GCL, GCL+STP, GCL+STP+DRL, GCL+STP+TrajRL) embedded in SimRN. Specifically, GCL is the proposed trajectory similarity model with multi-channel GCN and LSTM; GCL+STP is the model with prompt information extracted; GCL+STP+DRL applies the DRL-based framework for GCL+STP to automatically select the model parameters; and GCL+STP+TrajRL is processed by applying actor-learner scheme for distributed computing, which is SimRN essentially. In addition, to further evaluate the impact of multi-channel GCN, we introduce a new variant GCL w/o MC-GCL, which replaces the multi-channel GCN in GCL with a single-channel GCN. Note that, directly removing the multi-channel GCN is not feasible because both spatial and temporal embeddings are required. We calculate the ground truth for similarity trajectories on the T-Drive dataset using TP distance. The results are reported in Table 8, and the observations are as follows.

(i) GCL, which serves as the baseline trajectory similarity model with a multi-channel GCN and LSTM, achieves higher accuracy (*HR@10*, *HR@50*, *R10@50*) compared to GCL w/o MC-GCL. This is because the multi-channel GCN captures fine-grained spatial and temporal correlations from trajectories and road networks, preserving richer information for similarity learning.

(ii) GCL+STP and GCL+STP+DRL outperform the GCL in terms of computation accuracy, which indicates that our proposal of prompt information extraction is effective for trajectory similarity learning. In addition, GCL+STP+TrajRL (SimRN) has greatly improved the efficiency (i.e., $T_{tra}$, and $T_{com}$) of GCL+STP+DRL, because the introducing of actor-learner scheme supports efficient distributed training and processing, which reduces the time cost.

Overall, this set of experiments validate the great performance of four key components of SimRN.

## 5 RELATED WORK

### 5.1 Non-learning-based Methods

Early trajectory similarity measures [24, 48] in road networks extend raw trajectory measures. These methods first perform map matching to convert raw trajectories into road network-constrained trajectories, consisting of road vertices or segments. They then define similarity measures by adapting classic distance metrics such as DTW [52], EDR [7], and ERP [6], typically by aggregating shortest path distances between vertices or segments of two trajectories. Later studies introduce new distance measures specifically for road network-constrained trajectories, including LORS [45], LCRS [56], and TP [39]. LORS measures distance based on the length of overlapping edges, while LCRS identifies the longest common edge between two trajectories. Unlike these, TP accounts for both spatial and temporal similarity, while others focus only on spatial distance. However, non-learning-based methods often have quadratic time complexity ($O(c^2)$), limiting their scalability for large-scale trajectory analysis. Moreover, distance measures rely on manual point matching to compute the similarity between two road network constrained trajectories, making non-learning-based methods fail to capture deeper features hidden in the data.

### 5.2 Learning-based Methods

Recently, deep representation learning for trajectory similarity computation has gained attention due to its powerful approximation capabilities. Learning-based methods use neural networks to approximate distance functions for non-learning-based measures. Typically, GCNs are used to capture road network information, while RNNs (e.g., GRU and LSTM) represent trajectories by transforming high-dimensional data into low-dimensional vectors, reducing time complexity to linear. Note that, the learning-based methods are also required to perform map matching to obtain road network constrained trajectories for similarity learning. Specifically, Han et al. [18] introduced GTS, the first model for spatial trajectory similarity learning in road networks. Fang et al.[13] developed ST2Vec, the first to learn temporal trajectory similarity in road networks. Zhou et al. [60] proposed GRLSTM, extending LSTM to road networks by incorporating a trajectory knowledge graph to capture sequence features. While these methods enable trajectory similarity learning in road networks, they do not fully account for road network-specific features, and rely on manually tuned parameters, which negatively impact effectiveness. Moreover, these methods train neural networks sequentially and require large training datasets, leading to inefficient similarity computation.

## 6 CONCLUSION

In this paper, we propose SimRN, an efficient and effective trajectories similarity learning framework in road networks based on DRL, which is the first distributed DRL-based model for trajectory similarity computation. We transform trajectory similarity computation into a decision-making problem to select the proper model parameters automatically, with the spatio-temporal features extracted from the road network for improving the accuracy. To efficiently compute the trajectory similarity, we develop an actor-learner scheme for parallelized computation, and introduce an experience replay pool to store training experiences and guide the parameters selection. In addition, we present a self-supervised contrastive learning paradigm to generate sufficient samples and a multi-channel GCN combined with an LSTM model for similarity learning, enabling to deal with insufficient training samples. Extensive experiments using two public real-life datasets confirm that SimRN outperforms the state-of-the-art methods in terms of computation accuracy, training efficiency, and model generality. In the future, it is of interest to extend SimRN for streaming trajectory similarity learning, to develop its potential for automatically online analyses in urban scenarios.

## 7 ACKNOWLEDGEMENT

This work was supported by the NSFC under Grants No. (62472377, 62402422, 62025206, U23A20296). Tianyi Li is the corresponding author of this paper.

## REFERENCES

[1] Helmut Alt and Michael Godau. 1995. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.* 5 (1995), 75–91.

[2] Muhammed Fatih Balin, Kaan Sancak, and Umit V. Catalyurek. 2023. MG-GCN: A Scalable multi-GPU GCN Training Framework. In *Proceedings of the 51st International Conference on Parallel Processing*. Article 79, 11 pages.

[3] Mattia Belloni, Davide Tarsitano, and Edoardo Sabbioni. 2024. An experimental analysis of driver influence on battery electric bus energy consumption. In *VPPC*. 1–5.

[4] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. 2005. On Map-Matching Vehicle Tracking Data. *PVLDB*, 853–864.

[5] Bo Chen, Huaijie Zhu, Wei Liu, Jian Yin, Wang-Chien Lee, and Jianliang Xu. 2021. Querying Optimal Routes for Group Meetup. *Data Sci. Eng.* 6, 2 (2021), 180–191.

[6] Lei Chen and Raymond T. Ng. 2004. On The Marriage of Lp-norms and Edit Distance. In *VLDB*. 792–803.

[7] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and Fast Similarity Search for Moving Object Trajectories. In *SIGMOD*. 491–502.

[8] Yitian Chen, Yanfei Kang, Yixiong Chen, and Zizhuo Wang. 2020. Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing* 399 (2020), 491–501.

[9] Zhen Chen, Dalin Zhang, Shanshan Feng, Kaixuan Chen, Lisi Chen, Peng Han, and Shuo Shang. 2024. KGTS: Contrastive Trajectory Similarity Learning over Prompt Knowledge Graph Embedding. In *AAAI*. 8311–8319.

[10] Chris J. Maddison Arthur Guez Laurent Sifre et al David Silver, Aja Huang. 2016. Mastering the game of Go with deep neural networks and tree search. *Nat.* 529, 7587 (2016), 484–489.

[11] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J. Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB* 1, 2 (2008), 1542–1552.

[12] Chenguang Fang, Shaoxu Song, and Yinan Mei. 2022. On Repairing Timestamps for Regular Interval Time Series. *Proc. VLDB Endow.* 15, 9 (2022), 1848–1860.

[13] Ziquan Fang, Yuntao Du, Xinjun Zhu, Danlei Hu, Lu Chen, Yunjun Gao, and Christian S. Jensen. 2022. Spatio-Temporal Trajectory Similarity Learning in Road Networks. In *SIGKDD*. 347–356.

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. In *AISTATS*, Vol. 15. 315–323.

[15] A. N. Gorban and D. C. Wunsch Ii. 1998. The general approximation theorem. In *IEEE World Congress on IEEE International Joint Conference on Neural Networks*.

[16] Robert Mansel Gower, Nicolas Loizou, Xun Qian, Alibek Sailanbayev, Egor Shulgin, and Peter Richtárik. 2019. SGD: General Analysis and Improved Rates. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. 5200–5209.

[17] Priyanshu Gupta, Swagata Payra, R. Bhatla, and Sunita Verma. 2024. WRF-Chem modeling study of heat wave driven ozone over southeast region, India. *Environmental Pollution* 340 (2024), 122744.

[18] Peng Han, Jin Wang, Di Yao, Shuo Shang, and Xiangliang Zhang. 2021. A Graph-based Approach for Trajectory Similarity Computation in Spatial Networks. In *SIGKDD*. 556–564.

[19] Danlei Hu, Lu Chen, Hanxi Fang, Ziquan Fang, Tianyi Li, and Yunjun Gao. 2024. Spatio-Temporal Trajectory Similarity Measures: A Comprehensive Survey and Quantitative Study. *IEEE Trans. Knowl. Data Eng.* 36, 5 (2024), 2191–2212.

[20] Danlei Hu, Ziquan Fang, Hanxi Fang, Tianyi Li, Chunhui Shen, Lu Chen, and Yunjun Gao. 2024. Estimator: An Effective and Scalable Framework for Transportation Mode Classification over Trajectories. *TITS* (2024).

[21] Zhang Xiong Jia Wen, Chao Li. 2011. Behavior pattern extraction by trajectory analysis. *Frontiers of Computer Science* 5, 1 (2011), 37.

[22] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. 2020. Time2Vec: Learning a Vector Representation of Time. In *ICLR*.

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

[24] Satoshi Koide, Chuan Xiao, and Yoshiharu Ishikawa. 2020. Fast Subtrajectory Similarity Search in Road Networks under Weighted Edit Distance Constraints. *PVLDB* 13, 11 (2020), 2188–2201.

[25] Ioannis Kontopoulos, Antonios Makris, Dimitris Zissis, and Konstantinos Tserpes. 2021. A computer vision approach for trajectory classification. In *MDM*. 163–168.

[26] D. Kumar, S. Rajasegarar, M. Palaniswami, X. Wang, and C. Leckie. 2015. A Scalable Framework for Clustering Vehicle Trajectories in a Dense Road Network. In *SIGKDD*.

[27] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. 2008. Trajectory Outlier Detection: A Partition-and-Detect Framework. In *ICDE*. 140–149.

[28] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. 2008. Trajectory Outlier Detection: A Partition-and-Detect Framework. In *ICDE*. 140–149.

[29] Tianyi Li, Lu Chen, Christian S Jensen, Torben Bach Pedersen, Yunjun Gao, and Jilin Hu. 2022. Evolutionary clustering of moving objects. In *ICDE*. IEEE, 2399–2411.

[30] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. 2018. Deep Representation Learning for Trajectory Similarity Computation. In *ICDE*. 617–628.

[31] Shuo Ma, Yu Zheng, and Ouri Wolfson. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*. 410–421.

[32] Xupeng Miao, Xiaonan Nie, Yingxia Shao, Zhi Yang, Jiawei Jiang, Lingxiao Ma, and Bin Cui. 2021. Heterogeneity-Aware Distributed Machine Learning Training via Partial Reduce. In *SIGMOD*. 2262–2270.

[33] OpenStreetMap. 2022. https://master.apis.dev.openstreetmap.org/. (2022).

[34] Punit Rathore, Dheeraj Kumar, Sutharshan Rajasegarar, Marimuthu Palaniswami, and James C. Bezdek. 2019. A Scalable Framework for Trajectory Prediction. *TITS* 20, 10 (2019), 3860–3874.

[35] Aswathy Ravikumar and Harini Sriraman. 2023. Computationally Efficient Neural Rendering for Generator Adversarial Networks Using a Multi-GPU Cluster in a Cloud Environment. *IEEE Access* 11 (2023), 45559–45571.

[36] Yehezkel S. Resheff. 2016. Online trajectory segmentation and summary with applications to visualization and retrieval. In *IEEE BigData*. 1832–1840.

[37] M. T. Robinson. 1990. The temporal development of collision cascades in the binary collision approximation. *Nucl. Instrum. Methods Phys. Res. Sect. B* 48, 1-4 (1990), 408–413.

[38] Mohammad Reza Samsami and Hossein Alimadad. 2020. Distributed Deep Reinforcement Learning: An Overview. *CoRR* abs/2011.11012 (2020). https://arxiv.org/abs/2011.11012

[39] Shuo Shang, Lisi Chen, Zhewei Wei, Christian S. Jensen, Kai Zheng, and Panos Kalnis. 2017. Trajectory Similarity Join in Spatial Networks. *PVLDB* 10, 11 (2017), 1178–1189.

[40] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. DITA: Distributed In-Memory Trajectory Analytics. In *SIGMOD*. 725–740.

[41] Yu Suzuki, Jun Ishizuka, and Kyoji Kawagoe. 2008. A Similarity Search of Trajectory Data Using Textual Information Retrieval Techniques. In *DASFAA*, Vol. 4947. 627–634.

[42] Uber. 2022. https://expandedramblings.com/index.php/uber-statistics/. (2022).

[43] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *CoRR* abs/1807.03748 (2018).

[44] Michail Vlachos, Dimitrios Gunopulos, and George Kollios. 2002. Discovering Similar Multidimensional Trajectories. In *ICDE*. 673–684.

[45] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. 2018. Torch: A Search Engine for Trajectory Data. In *SIGIR*. 535–544.

[46] T. U. Wien, T. Eiter, T. Eiter, H. Mannila, and H. Mannila. 1994. Computing discrete Fréchet distance. *Technical report, Citeseer* 64, 3 (1994), 636–637.

[47] Dong Xie, Feifei Li, and Jeff M. Phillips. 2017. Distributed Trajectory Similarity Search. *PVLDB* 10, 11 (2017), 1478–1489.

[48] Xingzhe Xie, Wilfried Philips, Peter Veelaert, and Hamid K. Aghajan. 2014. Road network inference from GPS traces using DTW algorithm. In *ITSC*. 906–911.

[49] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical Evaluation of Rectified Activations in Convolutional Network. *CoRR* abs/1505.00853 (2015).

[50] Peilun Yang, Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, and Wenjie Zhang. 2022. TMN: Trajectory Matching Networks for Predicting Similarity. In *ICDE*. 1700–1713.

[51] Di Yao, Gao Cong, Chao Zhang, and Jingping Bi. 2019. Computing Trajectory Similarity in Linear Time: A Generic Seed-Guided Neural Metric Learning Approach. In *ICDE*. 1358–1369.

[52] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. 1998. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *ICDE*. 201–208.

[53] Qiyue Yin, Tongtong Yu, Shengqi Shen, Jun Yang, Meijing Zhao, Wancheng Ni, Kaiqi Huang, Bin Liang, and Liang Wang. 2024. Distributed Deep Reinforcement Learning: A Survey and a Multi-player Multi-agent Learning Toolbox. *Mach. Intell. Res.* 21, 3 (2024), 411–430.

[54] Sha Liu Lanning Wang Haohuan Fu Xin Liu Zuoning Chen Yizhou Yang, Longde Chen. 2025. Behaviour-diverse automatic penetration testing: a coverage-based deep reinforcement learning approach. *Frontiers of Computer Science* 19, 3 (2025), 193309.

[55] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In *IJCAI*. 3634–3640.

[56] Haitao Yuan and Guoliang Li. 2019. Distributed In-memory Trajectory Similarity Search and Join on Road Network. In *ICDE*. 1262–1273.

[57] 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.

[58] Hanyuan Zhang, Xinyu Zhang, Qize Jiang, Baihua Zheng, Zhenbang Sun, Weiwei Sun, and Changhu Wang. 2020. Trajectory Similarity Learning with Auxiliary Supervision and Optimal Matching. In *IJCAI*. 3209–3215.

[59] Da Zheng, Xiang Song, Chengru Yang, Dominique LaSalle, and George Karypis. 2022. Distributed Hybrid CPU and GPU training for Graph Neural Networks on Billion-Scale Heterogeneous Graphs. In *SIGKDD*. 4582–4591.

[60] Silin Zhou, Jing Li, Hao Wang, Shuo Shang, and Peng Han. 2023. GRLSTM: Trajectory Similarity Computation with Graph-Based Residual LSTM. In *AAAI*. 4972–4980.