

Having It Both Ways: Single Trajectory Embedding for Similarity Computation with Pairwise Learning

Jianing Si^{*†¶}, Haitao Yuan^{✉*‡}, Xiang Li[†], Nan Jiang[†], Xiao Ma[†], Guoliang Li[§], Shangguang Wang^{¶†}

[†]BUPT, China [‡]NTU, Singapore [§]THU, China [¶]Beiyou Shenzhen Institute, China

{sijianing, lx, jn_bupt, maxiao18, sgwang}@bupt.edu.cn, haitao.yuan@ntu.edu.sg, liguoliang@tsinghua.edu.cn

Abstract—Trajectory similarity measure is a fundamental component in trajectory databases, supporting many downstream trajectory tasks. Existing similarity functions often exhibit unacceptable time complexities, hampering their efficiency for real-world scenarios. To address this limitation, learning-based approximation techniques utilizing trajectory embeddings have been proposed. However, creating a robust embedding model presents challenges, including the lack of direct involvement in the computational similarity process, adherence to non-metric similarity spaces, and the integration of precise similarity computation alignments. To address these challenges, we introduce DTisT, a novel embedding framework that enhances trajectory embeddings by pairwise learning from dual-trajectory input models. DTisT not only captures the dynamics of trajectory similarity computation through a dual-trajectory learning model but also integrates a learnable virtual trajectory to align the embedding space with non-metric similarity spaces effectively. Additionally, we incorporate aligned information from actual similarity computations into our embedding process using an attention mask mechanism. To ensure effective learning, we adopt a pre-train and fine-tune strategy, utilizing contrastive learning during the pre-training stage. Extensive experiments conducted on two real datasets demonstrate that DTisT surpasses state-of-the-art methods, showcasing its effectiveness in trajectory similarity embedding.

I. INTRODUCTION

Trajectory similarity computation is pivotal for most trajectory analytics applications, including similar trajectory search [1]–[3], trajectory clustering [4], [5], and trajectory anomaly detection [6]–[8]. Furthermore, in many query operations over a trajectory dataset, such as range queries to find all trajectories passing through a spatial or spatio-temporal query range, trajectory similarity search is a fundamental operator that is non-trivial to process [9]. Numerous trajectory similarity functions exist, such as Dynamic Time Warping (DTW) [10], the Hausdorff distance [11], the Fréchet distance [12], and Edit distance with Real Penalty (ERP) [13]. However, the majority of these similarity/distance functions possess unacceptable time complexity, rendering them inefficient for real-world large-scale scenarios. In response, heuristic methods have been developed, applying approximations for acceleration purposes [14]–[16]. However, all these approaches are specifically tai-

This paper was supported by NSF of China(62425203, 62032003, 62372061), Shenzhen Science and Technology Program JCYJ 20240813165400002 and State Key Laboratory Of Networking And Switching Technology. Guoliang Li was supported by National Key R&D Program of China (2023YFB4503600), NSF of China (61925205, 62232009, 62102215), Zhongguancun Lab, Huawei and Beijing National Research Center for Information Science and Technology (BNRist).

^{*}Equal contribution. [✉]Corresponding author.

lored for a certain similarity metric and do not synergistically address this challenge well.

Therefore, various learning-based approximation techniques for trajectory similarity have been introduced. Broadly, these methods can be categorized into *Pairwise Learning* and *Embedding Learning*. *Pairwise Learning* approaches, such as TMN [17], primarily accept two trajectories as input and employ neural networks to model the similarity computation, culminating in the derivation of a similarity value. However, these methods overlook efficiency because they rely on the online encoding of trajectories, which can be computationally intensive when applied to the trajectory similarity search. Consequently, *Embedding Learning* approaches, including **trajCL**, **trajGAT**, **Neutraj**, **Traj2SimVec**, and **T3S** [18]–[22], have garnered significant attention. These techniques strive to develop an embedding model that transforms each single trajectory into a vector, and then employs vector similarity metrics, such as Euclidean distance, to measure trajectory similarity. These vectors can be precomputed and reused, thus, the trajectory similarity search can be efficiently accelerated by leveraging LSH [23] or IVFPQ [24], achieving a time complexity substantially lower than the traditional methods [25]. However, there are still some unresolved challenges in developing an effective single trajectory embedding model.

(C1) How to simultaneously achieve the efficiency of Embedding Learning and the precision of Pairwise Learning? *Pairwise Learning* methods achieve high precision by learning and simulating similarity calculations between dual trajectory inputs. However, they require forming pairs with every trajectory in the database for retrieval, which is impractical. *Embedding Learning* methods vectorize trajectories in advance, needing only the given trajectory to be vectorized during retrieval. Yet, they face a precision bottleneck due to their limitation of single trajectory input, which hinders learning the similarity calculation process. The challenge remains to enable single-input embedding models to effectively simulate similarity calculations between dual trajectories.

(C2) How can the model bridge the gap between the trajectory embedding space and the similarity space? Notably, many trajectory similarity functions do not satisfy the triangle inequality, resulting in their corresponding similarity spaces being non-metric. In contrast, the embedding space is inherently metric. This fundamental divergence is frequently overlooked in existing methods. As illustrated in Fig. 1, the challenge emerges for the non-metric DTW function $f(\cdot, \cdot)$,

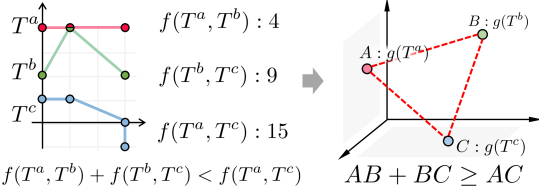


Fig. 1: An Example of Triangle Inequality Issue over DTW Similarity Space and Embedding Space.

where $f(T^a, T^b) + f(T^b, T^c) < f(T^a, T^c)$. However, for any embedding model $g(\cdot)$, the inequality $|g(T^a) - g(T^b)| + |g(T^b) - g(T^c)| \geq |g(T^a) - g(T^c)|$ invariably holds.

(C3) How can the aligned information inherent in similarity functions be effectively integrated into trajectory embedding? Existing learning paradigms only utilize the similarity value as a supervisory signal to train the embedding model, neglecting the aligned information inherent in the actual trajectory similarity computation. This neglect can result in compromised accuracy and interoperability. To illustrate, let's consider three distinct trajectories: T^a , T^b and T^c . If the training data only provides that $f(T^a, T^b) = f(T^a, T^c)$ for a similarity function $f(\cdot, \cdot)$, the learned embeddings for T^b and T^c would be identical, which is a flawed outcome.

To address the challenges above, we introduce a novel framework, **DTisT** (using **D**ual **T**rajecrory similarity to **I**nstruct **S**ingle **T**rajecrory embedding), designed to generate robust embeddings for trajectories. First, **to address C1**, we introduce an additional *Pairwise Learning* model named *dual-T* to assist the learning of single trajectory embedding mode *single-T*. In particular, *dual-T* accepts a pair of trajectories as input and simulates the similarity calculation process, producing a trajectory pair embedding. The norm value of this embedding is trained to represent the similarity between the trajectories. As a result, the pair embedding space in *dual-T* aligns with the difference vector of the trajectory embedding space in *single-T*. Therefore, we employ this pair embedding as a supervisory signal to regulate the disparity between the two embeddings produced by the *single-T* module. Through this, we can distill the knowledge acquired by the *dual-T* model during the simulation of similarity calculations into the *single-T* model, thereby enhancing the effectiveness of the *Embedding Learning* approach. **To address C2**, we introduce a learnable virtual trajectory and pair it with the actual trajectory as the input for *single-T*. Consequently, rather than directly producing a single trajectory embedding, the *single-T* module generates a pair embedding that captures the variance between the provided trajectory and the virtual trajectory. Thus, the difference between the trajectory embedding space and the similarity space can be learned through this virtual trajectory. **To address C3**, we integrate aligned information, such as matched point pairs from the actual similarity calculation, into the *dual-T* module. Specifically, we employ the multi-head attention mechanism to discern the correlation between any pair of points. Subsequently, we construct an attention mask to selectively filter valid correlations based on this aligned information. This attention mask serves a pivotal role, simulating the matching process present in the actual similarity

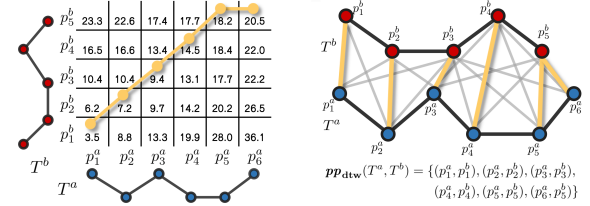


Fig. 2: An Example of DTW Trajectory Similarity Computation and Matched Point Pairs.

computation. Furthermore, to ensure the seamless convergence of the entire embedding model, we adopt a pre-train and fine-tune learning strategy. Notably, we leverage contrastive learning techniques to pre-train the embedding model. This involves asymmetrically selecting distant and proximate trajectories as positive and negative samples, respectively.

In summary, we make the following contributions:

- (1) We introduce a comprehensive framework **DTisT** to effectively employ the pairwise learning technique to guide the learning of single trajectory embedding model. (Sec. III) In particular, we seamlessly integrate aligned information from actual similarity computations into the pairwise learning process. (Sec. IV-B) To mitigate the triangle constraint of embeddings, we incorporate a learnable virtual trajectory in the trajectory embedding model. (Sec. IV-C)
- (2) To improve accuracy and efficiency in long trajectory tasks, we use a grid-based sub-trajectory aggregation method for better trajectory representation. (Sec. II) Our adaptable base encoder, featuring a recurrent neural network and an attention mechanism, effectively captures both short-term and long-term correlations within trajectories. (Sec. IV-A)
- (3) Our innovative learning strategy adopts a pre-train and finetune pipeline. Notably, during the pre-training phase, we leverage the supervised contrastive learning, incorporating asymmetric positive-negative example sampling, thereby bolstering the model's robustness. (Sec. V)
- (4) Through extensive evaluations on two real-world datasets, our empirical results demonstrate that **DTisT** markedly surpasses existing state-of-the-art methods in trajectory similarity embedding. (Sec. VI)

II. PRELIMINARY

In this section, we first introduce some key concepts about trajectory and trajectory similarity. Next, we formalize the problem of trajectory embedding for similarity computation.

A. Basic Concepts

Trajectory. A trajectory $T = [p_1, \dots, p_{|T|}]$ is a sequence of points obtained from GPS devices. Each point $p_i = (x_i, y_i)$ or $p_i = (x_i, y_i, t_i)$ can be a 2-dimensional or 3-dimensional tuple, representing latitude and longitude (or time). For simplicity, we consider points as 2-dimensional tuples in this paper following previous work [18], [20], and it's straightforward to extend it to 3-dimensional tuples.

Trajectory Similarity. Given two trajectories $T^a = [p_1^a, \dots, p_{|T^a|}^a]$ and $T^b = [p_1^b, \dots, p_{|T^b|}^b]$, the similarity between T^m and T^n is defined as the distance between the two corresponding point sequences $[p_1^a, \dots, p_{|T^a|}^a]$ and

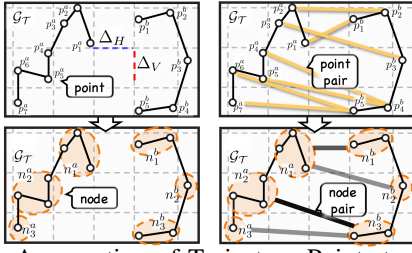


Fig. 3: The Aggregation of Trajectory Points to Grid Nodes. $[p_1^b, \dots, p_{T^b}^b]$ within a distance metric, such as DTW [10], Hausdorff [11], LCSS [26] and Fréchet [12]. For simplicity, we use the following DTW distance as an example of similarity due to its widespread use in real-world scenarios.

$$\begin{aligned} f_{\text{dtw}}(T^a, T^b) &= \text{DTW}[p_{T^a}^a, p_{T^b}^b] \\ \text{DTW}[p_i^a, p_j^b] &= \mathbf{dist}(p_i^a, p_j^b) + \min\{\text{DTW}[p_{i-1}^a, p_j^b], \\ &\quad \text{DTW}[p_i^a, p_{j-1}^b], \text{DTW}[p_{i-1}^a, p_{j-1}^b]\} \end{aligned} \quad (1)$$

where $\mathbf{dist}(p_i^a, p_j^b)$ is the distance between p_i^a and p_j^b which can be the Euclidean distance or another suitable metric depending on the point definition. Consequently, we can employ this similarity metric to assess the distance between any pair of trajectories. Specifically, for a given trajectory T^a , if another trajectory is proximate to T^a , we designate it as a positive trajectory, represented as T^{a+} . Conversely, if it is distant from T^a , we label it as a negative trajectory, denoted as T^{a-} .

Matched Point Pairs. Like Formula 1, the computation of similarity between two trajectories can be conceptualized as the summation of distances between specific point pairs. These pairs are chosen from the Cartesian product of the corresponding point sequences, with the objective of minimizing the specific similarity's cumulative distance. Additionally, it is imperative that each point in one trajectory has a corresponding matched point in the other trajectory. These chosen pairs are termed matched point pairs. Specifically, the Hausdorff distance is calculated by selecting the maximum distance from a set of matched point pairs, where each matched pair consists of a point from one trajectory and its closest point from another trajectory.

Example 1: As depicted in Fig. 2, consider two trajectories $T^a = [p_1^a, \dots, p_6^a]$ and $T^b = [p_1^b, \dots, p_5^b]$. The initial step involves computing the distance for each pair (p_i^a, p_j^b) where $1 \leq i \leq 6$ and $1 \leq j \leq 5$, derived from the Cartesian product. Subsequently, an accumulated distance matrix is recursively constructed. The value at the matrix element (i, j) is equivalent to the similarity $f_{\text{dtw}}(T^{a-(|T^a|-i)}, T^{b-(|T^b|-j)})$. The optimal warping route can then be traced within this matrix, commencing from the terminal matched pair (p_6^a, p_5^b) . Specifically, the final matched pairs along this route, illustrated using yellow lines, are represented as $\mathbf{pp}_{\text{dtw}}(T^a, T^b) = \{(p_1^a, p_1^b), (p_2^a, p_2^b), (p_3^a, p_3^b), (p_4^a, p_4^b), (p_5^a, p_5^b), (p_6^a, p_5^b)\}$.

Trajectory Grid Node. Consider a trajectory database, denoted as \mathcal{T} , comprising trajectories situated on a map $\mathcal{G}_{\mathcal{T}}$. As illustrated in Fig. 3, we partition $\mathcal{G}_{\mathcal{T}}$ into uniform grids, each characterized by identical horizontal spacing Δ_H and vertical spacing Δ_V . Referring to Fig. 3, for a given trajectory T , when successive points reside within the same grid, we categorize these points as a trajectory grid node (namely a sub-trajectory),

represented as n . This implies that the trajectory T can be substituted with the grid node sequence $[n_1, \dots, n_L]$, where L denotes the count of nodes in trajectory T . Furthermore, the node n_i symbolizes a series of contiguous points situated within the pertinent grid. To elaborate, the node n_i can be expressed as the point subsequence $[p_{h_i}, p_{h_i+1}, \dots, p_{h_i+l_i-1}]$, where l_i represents the quantity of points. The relationships $h_{i+1} = h_i + l_i$ ensures that nodes remain sequentially connected in the original trajectory, adhering to the point order.

Matched Node Pairs. Similarly, the aforementioned matched point pairs can be aggregated into node pairs in the same manner by grid cells. Formally, given two trajectories (i.e., T^a and T^b) and their matched point pairs $\mathbf{pp}(T^a, T^b)$, we can extract the corresponding matched node pairs $\mathbf{np}(T^a, T^b)$ as outlined below:

$$\begin{aligned} \mathbf{np}(T^a, T^b) &= \{(n^a, n^b) \mid \forall n^a \in T^a, \forall n^b \in T^b, \\ &\quad \exists p^a \in n^a, \exists p^b \in n^b, (p^a, p^b) \in \mathbf{pp}(T^a, T^b)\} \end{aligned} \quad (2)$$

In addition, we can calculate the number of corresponding point pairs for each matched node pair (n^a, n^b) as below:

$$\mathbb{P}^{(n^a, n^b)} = \left| \{(p^a, p^b) \mid \forall p^a \in n^a, \forall p^b \in n^b, (p^a, p^b) \in \mathbf{pp}(T^a, T^b)\} \right|$$

Example 2: Referring to Fig. 3, consider two trajectories: $T^a = [p_1^a, \dots, p_6^a]$ and $T^b = [p_1^b, \dots, p_5^b]$. When partitioned into grid cells using the specified Δ_H and Δ_V distances, these trajectories can be equivalently represented as $T^a = [n_1^a, n_2^a, n_3^a]$ and $T^b = [n_1^b, n_2^b, n_3^b]$, respectively. Given the matched point pairs, illustrated by yellow lines, as $\mathbf{pp}(T^a, T^b) = \{(p_1^a, p_1^b), (p_2^a, p_2^b), (p_3^a, p_3^b), (p_4^a, p_4^b), (p_5^a, p_4^b), (p_6^a, p_5^b), (p_7^a, p_5^b)\}$, we can deduce the corresponding matched node pairs as $\mathbf{np}(T^a, T^b) = \{(n_1^a, n_1^b), (n_1^a, n_2^b), (n_2^a, n_3^b), (n_3^a, n_3^b)\}$. These pairs are distinctly highlighted by the bold black lines in the bottom right subplot of the figure. Furthermore, the associated node-pair values, represented by the grayscale intensity of these bold black lines, are given by: $\mathbb{P}^{(n_1^a, n_1^b)} = 2$, $\mathbb{P}^{(n_1^a, n_2^b)} = 1$, $\mathbb{P}^{(n_2^a, n_3^b)} = 3$, and $\mathbb{P}^{(n_3^a, n_3^b)} = 1$.

B. Problem Formulation

Trajectory Embedding. The concept of trajectory embedding pertains to an encoding function, $g(\cdot)$, which maps a trajectory T to a d -dimensional vector, represented as $e = g(T) \in \mathbb{R}^d$. Distinct from conventional embedding challenges, the primary objective of trajectory embedding is to devise an encoder model such that the Euclidean distance of embeddings $\sqrt{\|e^a - e^b\|^2}$ closely approximates the provided trajectory similarity measure, such as $f_{\text{dtw}}(T^a, T^b)$. This task is also referred to as trajectory embedding for similarity computation.

Definition 1 (Trajectory Embedding for Similarity Computation): Given a trajectory database $\mathcal{T} = \{T^1, T^2, \dots, T^N\}$ containing N trajectories, and a trajectory similarity metric $f(\cdot, \cdot)$, our objective is to learn an embedding model, $g_{\theta}(\cdot)$, such that the discrepancy between the Euclidean distance of the embeddings and the actual similarity $f(T^i, T^j)$ for any pair of trajectories is minimized. The formulation is as follows:

$$\underset{\theta}{\operatorname{argmin}} \sum_{1 \leq i, j \leq N} \left| \sqrt{\|g_{\theta}(T^i) - g_{\theta}(T^j)\|^2} - f(T^i, T^j) \right|$$

III. AN OVERVIEW OF SOLUTION FRAMEWORK

Fig. 4 illustrates the framework of our proposed model **DTisT**. We will first introduce the framework architecture and then explain the pipeline of model training and inference.

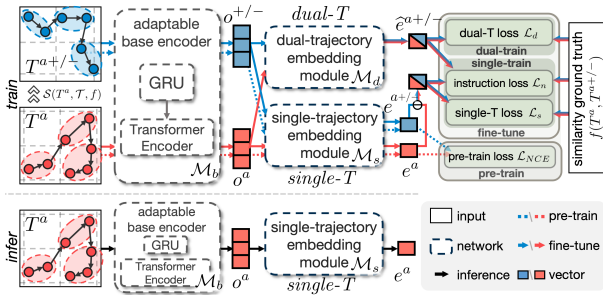


Fig. 4: The Architecture Overview of DTisT.

A. Framework Architecture

In terms of architecture, **DTisT** contains three modules:

- 1) The first part, denoted as \mathcal{M}_b , embodies the adaptable base encoder module. Its primary objective is to transform each trajectory T from a point embedding sequence into a node embedding sequence o . This transformation takes into account short-term and long-term sequential correlations within the point sequence and the grid node sequence, respectively. In particular, this module integrates two sequential encoding models: the GRU (an RNN model adept at capturing short-term correlations at the point level) and the Transformer Encoder (an attention-based model designed to grasp long-term correlations at the grid node level). As a result, the length of the node embedding sequence o aligns with the number of grids the trajectory T traverses. This is considerably more compact than the point sequence, enhancing the efficiency of subsequent embedding processes.
- 2) The second part, denoted as *single-T* \mathcal{M}_s , aims to convert the node embedding sequence o of each trajectory T into a final embedding vector e .
- 3) The third part, denoted as *dual-T* \mathcal{M}_d , is designed to integrate the aligned information, thereby directing the learning process of the second module. Specifically, *dual-T* ingests the intermediate node embedding sequences of a pair of trajectories and subsequently produces the pair embedding, denoted as \hat{e} . The norm of this embedding serves as a representation of their similarity. In essence, this pair embedding can be employed to steer the trajectory embedding learning. This is achieved by minimizing the discrepancy between the vector \hat{e}^{a-b} and the difference vector $e^a - e^b$ for the trajectories T^a and T^b .

B. The Pipeline of Model Training and Inference

As illustrated in Fig. 4, the training approach aims to train all three modules within a two-phase pipeline that includes a pre-training phase and a fine-tuning phase. During inference, only the modules \mathcal{M}_b and \mathcal{M}_s are used.

Pre-training. It's imperative to note that our ultimate objective is to harness the learned models, \mathcal{M}_b and \mathcal{M}_s , to effectively embed each trajectory into a meaningful vector. To prime the parameters of these modules optimally, we initiate their training with a contrastive learning paradigm. Specifically, we craft a novel sampling technique to select positive and negative trajectories, represented as $T^{a+/-}$, to serve as pre-training examples for any given trajectory T^a . Fundamentally, we employ the pre-training loss, \mathcal{L}_{NCE} , to ensure that the

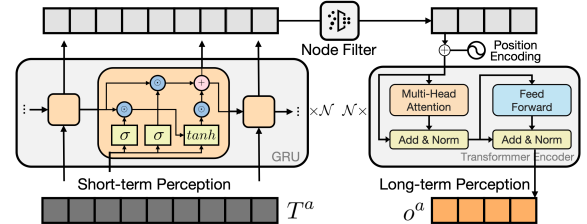


Fig. 5: The architecture of Adaptable Base Encoder.

embedding model adheres to the constraint: the similarity between the generated embedding of T^a and its positive counterpart T^{a+} should surpass that of its negative counterpart T^{a-} . Consequently, these pre-trained modules are enhanced to capture the ability to discriminate remoteness and proximity. For the sake of convenience, we use $*^{a+/-}$ to denote $*^{a+}$ or $*^{a-}$ ($*$ indicates T , o , e or \hat{e}) if there is no ambiguity in the following sections.

Fine-tuning. Directly fine-tuning both embedding modules simultaneously can lead to instability. To address this, we segment the fine-tuning stage into two distinct training branches: dual-training and single-training. Firstly, we utilize the same training data as the pre-training phase, where each trajectory is processed through the module \mathcal{M}_b , resulting in the generation of a corresponding node embedding sequence for the trajectory. In the dual-training branch, we input pairs of node embedding sequences (specifically, the positive or negative pairs $(o^a, o^{a+/-})$) into the *dual-T* module \mathcal{M}_d , producing the pair embedding denoted as $\hat{e}^{a+/-}$. We then compute the discrepancy between the norm value of the pair embedding and the actual similarity, represented by $f(T^a, T^{a+/-})$. The modules \mathcal{M}_b and \mathcal{M}_d are fine-tuned by minimizing this discrepancy, which we term the *dual-T* loss \mathcal{L}_d . Conversely, in the single-training branch, the modules \mathcal{M}_b and \mathcal{M}_s are employed to generate two distinct embeddings for the trajectories in a given pair (e.g., $(e^a, e^{a+/-})$ for the positive/negative pair). The difference between these embeddings, $e^a - e^{a+/-}$, serves as the pair embedding, with its norm value interpreted as the similarity between the trajectories in the pair. We optimize the modules by minimizing the difference between $|e^a - e^{a+/-}|$ and the actual similarity $f(T^a, T^{a+/-})$. This is referred to as the *single-T* loss \mathcal{L}_s . Furthermore, to integrate the aligned information between trajectories in a pair, we keep the *dual-T* module parameters constant and use its output, the pair embedding $\hat{e}^{a+/-}$, as a guiding signal for the *single-T* module's training. Specifically, we introduce the instruction loss \mathcal{L}_n to minimize the difference between $e^a - e^{a+/-}$ and $\hat{e}^{a+/-}$, further constraining the learning of the modules \mathcal{M}_b and \mathcal{M}_s .

Inference. Only the modules \mathcal{M}_s and \mathcal{M}_b are used in the inference pipeline. The trajectory T^a is encoded by \mathcal{M}_b into a sequence of node embeddings o , which is then processed by \mathcal{M}_s to produce the final trajectory embedding e^a . Consequently, during inference, **DTisT** operates as a single trajectory embedding method, ensuring that the online similar retrieval process excludes encoding trajectories, thus maintaining a consistent retrieval time complexity.

IV. MODEL STRUCTURE

In this section, we elucidate how we bring the previously discussed motivations to design distinct modules, including the adaptable base encoder module \mathcal{M}_b (Sec. IV-A), the dual-trajectory embedding module \mathcal{M}_d (Sec. IV-B) and the single-trajectory embedding module \mathcal{M}_s (Sec. IV-C).

A. Adaptable Base Encoder

This module seamlessly integrates point-level and trajectory-level granularity using grid node embedding, effectively linking short-term and long-term perspectives. To alleviate the challenges posed by trajectory length adaptation and inefficient memory utilization, we survey several canonical seq2vec architectures, including RNN-link and Transformer. Our findings illuminated that RNN-based architectures perform better for shorter trajectories, whereas the Transformer displays contrasting behavior, favoring longer trajectories. Consequently, we strategically employed the RNN-like network, GRU, for short-term sequence perceptions and harnessed the Transformer encoder for long-term sequence apprehensions. As illustrated in Fig. 5, the entire module unfolds in a tripartite progression: initial short-term perception, followed by node filtering, culminating in the long-term perception phase.

Short-term Perception. Given the original trajectory $T^a = [p_1^a, \dots, p_{|T^a|}^a]$, we leverage the GRU model to accumulate sequential point information into the hidden sequence $\mathbf{x}_{p^a} = [x_{p_1^a}, x_{p_2^a}, \dots, x_{p_{|T^a|}^a}]$. Consequently, this hidden content x is able to aggregate short-term prefix points at each trajectory point. This capability establishes a foundational framework for the grid-level aggregation characteristic of the Node Filter.

Node Filter. In this step, our objective is to derive hidden representations for the grid node sequence $[n_1^a, \dots, n_L^a]$ corresponding to the provided trajectory T^a using the hidden sequence \mathbf{x}_{p^a} . Given the defined properties of a trajectory grid node, each node n_i^a encompasses the point subsequence $[p_{h_i}, \dots, p_{h_i+l_i-1}]$. To represent the sequence effectively, we capitalize on the forgetting mechanism inherent to the GRU network. Specifically, we designate the final point within each grid cell as the representative node for its respective grid sequence segment. Hence, $x_{p_{h_i+l_i-1}^a}$ can be utilized to depict the hidden representation $x_{n_i^a}$ of node n_i^a , that is $x_{n_i^a} = x_{p_{h_i+l_i-1}^a}$. Consequently, this allows us to derive the associated node hidden sequence $\mathbf{x}_{n^a} = [x_{n_1^a}, x_{n_2^a}, \dots, x_{n_L^a}]$, where L , significantly smaller than $|T^a|$, indicates the count of grid nodes traversed by T^a .

Long-term Perception. To further encode the hidden sequence \mathbf{x}_{n^a} , our aim is to discern the long-term correlations throughout the entire trajectory. Drawing inspiration from the study presented in [27], we incorporate the attention mechanism by deploying the Transformer encoder to transition the sequence \mathbf{x}_{n^a} to o^a . This facilitates the collation of trajectory information at the grid level, aligning points of varying density distributions within the trajectory to uniform granularity.

Remark. For a comprehensive formulation of the adaptable base encoder, please refer to our technical report [28], which

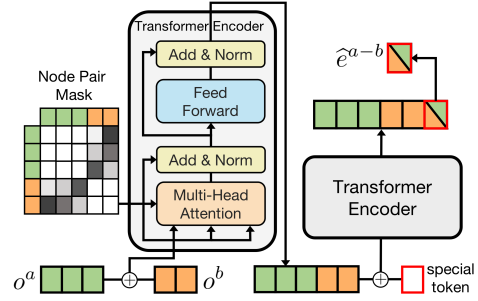


Fig. 6: The Architecture of Dual-Trajectory Embedding.

includes detailed descriptions of the processes involved in both short-term and long-term perception.

B. Dual-Trajectory Embedding

As delineated in Fig. 6, the dual trajectory embedding module unfolds in two sequential encoding phases, with both leveraging the capabilities of the Transformer Encoder. In the initial phase, termed the node-pair supervised matching encoder, we prioritize aligned information by integrating the node-pair mask into the attention mechanism. Subsequently, in the phase denominated as the logical difference aggregator, a specialized token is introduced. This facilitates the aggregation of the node embedding sequence, culminating in the coherent embedding difference, represented as \hat{e}^{a-b} .

Node-pair Supervised Matching Encoder. Given the input pair (o^a, o^b) , this module generates a node-pair accumulated sequence \hat{o}^{a-b} supervised by the ground truth of aligned node-pairs. Actually, the attention network of the Transformer Encoder is naturally suitable for simulating a matching mechanism. To harness this network architecture, we commence by concatenating the two trajectories (o^a, o^b) to yield a unified node sequence represented as $o^{a-b} = o^a \oplus o^b$, where \oplus symbolizes the sequence concatenation operation. Subsequently, we design a mapping function, transitioning from node-pair information to the attention mask as follows:

$$Mask_{(i,j)}^{a-b} = \begin{cases} W_m \cdot \mathbb{P}(n_i^a, n_j^b - L^a) + b_m & i \leq L^a, j > L^a \\ W_m \cdot \mathbb{P}(n_i^a - L^a, n_j^b) + b_m & i > L^a, j \leq L^a \\ W_m \cdot \mathbf{0} + b_m & \text{else} \end{cases}$$

where both W_m and b_m are learnable parameters, and play a pivotal role in discerning the significance of varying node pairs within the attention-mask matrix. Notably, b_m is initialized to an extremely large negative value, which ensures that responses from irrelevant nodes during attention are effectively masked, while simultaneously emphasizing node-pairs with a higher number of point-pairs to receive proportionally greater attention. As a result, we get the mask matrix $Mask^{a-b} \in \mathbb{R}^{(L^a+L^b) \times (L^a+L^b)}$, which is then applied to compute the attention score as follows:

$$Att_{o^{a-b}}^{H_h}(i) = \sum_j \sigma \left(\frac{\mathbf{W}_Q^{H_h} o_i^{a-b} \cdot \mathbf{W}_K^{H_h} o_j^{a-b} + Mask_{(i,j)}^{a-b}}{\sqrt{d}} \right) \mathbf{W}_V^{H_h} o_j^{a-b}$$

where $\mathbf{W}_Q^{H_h}, \mathbf{W}_K^{H_h}, \mathbf{W}_V^{H_h} \in \mathbb{R}^{(d/H_h) \times d}$ denote the parameter matrices associated with Self-Attention. Additionally, H_h indicates the h-th head within this multi-head self-attention architecture. The function σ signifies the activation function, while $o_i^{a-b} \in \mathbb{R}^d$ represents the vector corresponding to the i-th node within the concatenated trajectory o^{a-b} . The

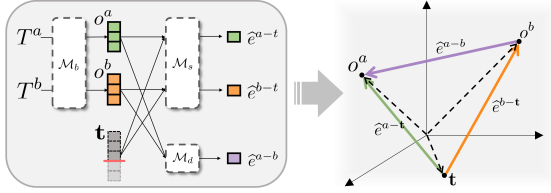


Fig. 7: An Example of Single and Dual Embeddings.

subsequent stages align with standard practices in Transformer Encoder, and can be summarized as follows:

$$\begin{aligned}\hat{o}^{a-b} &= \mathbf{LayerNorm} \left(\left(\text{Att}_{o^{a-b}}^{H_1} \oplus \dots \oplus \text{Att}_{o^{a-b}}^{H_n} \right) \mathbf{W}_M + o^{a-b} \right) \\ \bar{o}^{a-b} &= \mathbf{ReLU}(\mathbf{W}_F \cdot \hat{o}^{a-b} + \mathbf{b}_F) \\ \hat{o}^{a-b} &= \mathbf{LayerNorm}(\bar{o}^{a-b} + \hat{o}^{a-b})\end{aligned}$$

Logical Difference Aggregator. The main task of the latter Encoder is to convert the output sequence \hat{o}^{a-b} into a vector, the embedding logical difference e^{a-b} . Following the seq2vec approach in NLP tasks, we introduce a learnable special token $\mathbf{v}_d \in \mathbb{R}^{1 \times d}$ to aggregate the information of the entire node-pair accumulated sequence at a token position, ultimately outputting the embedding vector for the two trajectories. To be brief:

$$e^{a-b} = \mathbf{TransformerEncoder}(\hat{o}^{a-b} \oplus \mathbf{v}^d)[-1]$$

where \oplus represents the concatenation operator of the token and sequence. $[-1]$ represents the last token in the resulting sequence, which corresponds to the position of \mathbf{v}^d .

C. Single-Trajectory Embedding

Given an intermediate result of a single trajectory, denoted as o^a , its corresponding real trajectory embedding, e^a , is derived. It's worth noting that the network architecture of the *single-T* model aligns with that of the *dual-T model*. To achieve this goal, we introduce a synthetic trajectory, denoted as \mathbf{t} , to emulate the trajectory matching process. Actually, this virtual trajectory which serves as single-T's trainable parameter, acts as a heuristic intermediate representation, facilitating the activation of high-priority nodes from the viewpoint of the whole trajectory database. During initialization, we set the sequence length of \mathbf{t} to match that of the longest trajectory ($\mathbf{t} \in \mathbb{R}^{\max(|T|) \times d}$). Subsequently, it is truncated to mirror the length of o^a and concatenated with o^a , as given by:

$$o^{a-t} = o^a \oplus \mathbf{t}[:L^a] \quad (3)$$

To enable this virtual trajectory to proactively provide discriminative results during attention activation, especially at pivotal input nodes, and to facilitate the extraction of salient nodes from the input trajectory o^a , we define the node-pair relation and its corresponding fabricated mask matrix $Mask_{a-t}$ as follows:

$$Mask_{a-t}(i, j) = \begin{cases} W_m \cdot \mathbf{1} + b_m, & \text{if } i = j + L^a \text{ or } i + L^a = j, \\ W_m \cdot \mathbf{0} + b_m, & \text{otherwise.} \end{cases} \quad (4)$$

Here, $Mask_{a-t} \in \mathbb{R}^{2L^a \times 2L^a}$ represents the attention mask for the concatenated input and trainable trajectories. These are aligned in the order of their respective trajectory nodes, with each node on o^a paired with its counterpart on \mathbf{t} . Similarly, we can get the pair embedding \hat{e}^{a-t} as bellow:

$$\begin{aligned}\hat{o}^{a-t} &= \mathbf{TransformerEncoder}(o^{a-t}, Mask_{a-t}) \\ \hat{e}^{a-t} &= \mathbf{TransformerEncoder}(\hat{o}^{a-t} \oplus \mathbf{v}^s)[-1]\end{aligned} \quad (5)$$

where $\mathbf{v}^s \in \mathbb{R}^{1 \times d}$ denotes a special token, aggregating the information of the entire trajectory. To this end, the pair

embedding can be regarded as the single trajectory embedding, which is denoted as $e^a = \hat{e}^{a-t}$.

Discussion. As illustrated in Fig. 7, both single embedding and dual embedding space are congruent, stemming from the analogous architecture of the *single-T* and *dual-T* models. Consequently, the supervisory signal, denoted as \hat{e}^{a-b} , optimally directs the learning processes of \hat{e}^{a-t} and \hat{e}^{b-t} . This alignment is captured by the relation $\hat{e}^{a-b} = \hat{e}^{a-t} - \hat{e}^{b-t}$ within the shared embedding space. Notably, the variable-length truncation of \mathbf{t} introduces slight differences in \hat{e}^{a-t} and \hat{e}^{b-t} , relaxing the strict transitive relations among trajectory embeddings, which in turn eases the triangle inequality constraints.

V. MODEL LEARNING

In this section, we elaborate on the details of learning the whole model. Initially, we introduce an innovative sampling technique tailored to generate training data (Sec. V-A). Subsequently, our attention shifts to the design of the contrastive learning loss, denoted as \mathcal{L}_{NCE} , which is pivotal during the pre-training phase (Sec. V-B). Lastly, we elucidate the various loss functions (i.e., \mathcal{L}_s , \mathcal{L}_d and \mathcal{L}_n) employed during the fine-tuning phase (Sec. V-C).

A. Positive-Negative Divergent Sampling

Prior research, such as **trajCL** [18] and **trajGAT** [19], has shown that sampling both positive and negative trajectory pairs is crucial for optimizing the trajectory embedding model. Notably, traditional contrastive methods in trajectory, such as [18], [29], predominantly utilize data augmentation to derive positive or negative contrastive samples, introducing potential noise into the data. Differing from these methods, our approach leverages an asymmetrically designated similarity function to discern distant/near trajectories within the database. This method allows us to innately source trajectories based on their inherent similarity metrics, designating them as positive or negative training pairs. Consequently, our sampling procedure can be divided into $\mathcal{S}(T^a, \mathcal{T}, f) = \mathcal{S}^+(T^a, \mathcal{T}, f) + \mathcal{S}^-(T^a, \mathcal{T}, f)$, where $\mathcal{S}^+(\cdot)$ and $\mathcal{S}^-(\cdot)$ denote the sampling for positive and negative pairs, respectively.

Sampling Positive Trajectory Pairs. Given that each trajectory T^a , firstly we randomly sampled m trajectories from the database \mathcal{T} to construct a subset \mathcal{T}^a , in order to visit all different trajectories. Then we select the η most similar pairs as positive samples. This can be mathematically denoted as:

$$\mathcal{S}^+(T^a, \mathcal{T}, f) = \mathbf{sort} \left(\left\{ T^i : f(T^a, T^i) | T^i \in \mathcal{T}^a \right\} \right) [: \eta] \quad (6)$$

This approach is grounded in the understanding that the most similar pairs can pose challenges in differentiation. By incorporating these challenging examples into the training data, we aim to bolster the model's discriminative capabilities.

Sampling Negative Trajectory Pairs. An intuitive method is to select the η most dissimilar pairs as negative samples from its subset \mathcal{T}^a . However, this strategy would cause the model to focus excessively on distinct parts of trajectories, resulting in insufficient model robustness. To address this issue, we sample η negative samples based on the probability from the rest part of its positive sampling, which can be formulated as follows:

$$S^-(T^a, \mathcal{T}, f) = \left\{ T^i | T^i \sim p_f^a(T^i), T^i \in \mathcal{T}_*^a, 1 \leq i \leq \eta \right\} \quad (7)$$

$$p_f^a(T^i) = f(T^i, T^a) / \sum_{T^j \in \mathcal{T}_*^a} f(T^j, T^a)$$

$$\mathcal{T}_*^a = \mathcal{T}^a - S^+(T^a, \mathcal{T}, f)$$

where the probability $p_f^a(T^i)$ used in this sampling process is a normalization of the similarity value $f(T^i, T^a)$ with the anchor trajectory, allowing trajectories further from the anchor to have a higher probability of being selected as negative trajectories. To facilitate understanding, we have included an example of a negative sample in our technical report [28].

As a result, we get η positive pairs and η negative pairs for each trajectory T^a , and denote them as $\mathbf{T}_a^+ = \{T_1^{a+}, T_2^{a+}, \dots, T_\eta^{a+}\}$ and $\mathbf{T}_a^- = \{T_1^{a-}, T_2^{a-}, \dots, T_\eta^{a-}\}$ respectively. Similarly, the associated single embeddings can be denoted as $\mathbf{e}_a^+ = \{e_1^{a+}, \dots, e_\eta^{a+}\}$ and $\mathbf{e}_a^- = \{e_1^{a-}, \dots, e_\eta^{a-}\}$, and the associated dual embeddings can be denoted as $\widehat{\mathbf{e}}_a^+ = \{\widehat{e}_1^{a+}, \dots, \widehat{e}_\eta^{a+}\}$ and $\widehat{\mathbf{e}}_a^- = \{\widehat{e}_1^{a-}, \dots, \widehat{e}_\eta^{a-}\}$. Notably, the elements in \mathbf{e}_a^+ and $\widehat{\mathbf{e}}_a^+$ should be sorted in ascending order based on the similarity distance $f(\cdot, \cdot)$ corresponding to T^a and trajectories in \mathbf{T}_a^+ which means similar ones should be closer to the front. \mathbf{e}_a^- and $\widehat{\mathbf{e}}_a^-$ should be sorted in descending order which means dissimilar ones should be closer to the front.

B. Pre-Training Loss

Following previous contrastive learning research, we utilize InfoNCE [30], [31] as the loss function. The objective is to maximize the compactness between \mathbf{e}^a and \mathbf{e}_a^+ and maximize the separation between \mathbf{e}^a and \mathbf{e}_a^- , making similar trajectories closer in semantics and dissimilar trajectories further apart. The loss \mathcal{L}_{NCE}^a for T^a is defined as follows:

$$\mathcal{L}_{NCE}^a = - \sum_{i=1}^{\eta} \log \frac{\exp\left(\frac{\cos(e^a, e_i^{a+})}{\tau}\right)}{\exp\left(\frac{\cos(e^a, e_i^{a+})}{\tau}\right) + \sum_{j=1}^{\eta} \exp\left(\frac{\cos(e^a, e_j^{a-})}{\tau}\right)} \quad (8)$$

where $\cos(\cdot, \cdot)$ is the cosine similarity function and τ is the temperature parameter that controls the contribution of the negative samples [32], and $\exp(\cdot)$ represents the exponential function.

C. Fine-Tuning Losses

Initially, given the ground truth $f(T^a, T^{a+/-})$, we utilize the loss \mathcal{L}_s or the loss \mathcal{L}_d to gauge the disparity between the ground truth and the estimated result of the *single-T* model or the *dual-T* model. Notably, both losses adhere to a consistent format termed the *index weighted embedding loss*. Regarding the instruction loss \mathcal{L}_n , its primary function is to assess the discrepancy between the outputs of the two embedding models.

Index Weighted Embedding Loss. Different samples may possess varying degrees of significance during model learning. To address this, we assign weights to trajectory pairs based on their similarity. Specifically, for a given trajectory T^a , positive trajectories (\mathbf{T}_a^+) in close proximity to T^a and negative trajectories (\mathbf{T}_a^-) at a greater distance from T^a should exert a higher influence and hence receive larger weights. Given the pre-existing order in \mathbf{T}_a^+ and \mathbf{T}_a^- , we define the weight for the i -th pair, either (T^a, T_i^{a+}) or (T^a, T_i^{a-}) , using its corresponding index as:

$$\omega_i = \frac{\eta - i + 1}{\sum_{q=1}^{\eta} (\eta - q + 1)} = \frac{2(\eta - i + 1)}{\eta(\eta + 1)} \quad (9)$$

Algorithm 1: Model Learning for DTisT

Input: trajectory database \mathcal{T} , similarity metric f , sampling function \mathcal{S} , pre-training epoch I_p , training epoch I , modules ($\mathcal{M}_b, \mathcal{M}_s, \mathcal{M}_d$), and their corresponding parameters ($\theta_b, \theta_s, \theta_d$).
Output: best model parameters θ_b^*, θ_s^* .

```

1 for  $i \leftarrow 1 \dots I_p$  do
  // the pre-training phase
2  $[T^a, \mathbf{T}^{a+}, \mathbf{T}^{a-}] \leftarrow \mathcal{S}(\mathcal{T}, f)$ ;
3  $[o^a, \mathbf{o}^{a+}, \mathbf{o}^{a-}] \leftarrow \mathcal{M}_b([T^a, \mathbf{T}^{a+}, \mathbf{T}^{a-}])$ ;
4  $[e^a, \mathbf{e}^{a+}, \mathbf{e}^{a-}] \leftarrow \mathcal{M}_s([o^a, \mathbf{o}^{a+}, \mathbf{o}^{a-}])$ ;
5 using Formula 8 to compute  $\mathcal{L}_{NCE}^a$  for each  $T^a$ ;
6  $\theta_b, \theta_s \leftarrow \mathbf{BP}(\sum_a \mathcal{L}_{NCE}^a)$ 
7 for  $i \leftarrow 1 \dots I$  do
  // the fine-tuning phase
8  $[T^a, \mathbf{T}^{a+}, \mathbf{T}^{a-}] \leftarrow \mathcal{S}(\mathcal{T}, f)$ ;
9  $[o^a, \mathbf{o}^{a+}, \mathbf{o}^{a-}] \leftarrow \mathcal{M}_b([T^a, \mathbf{T}^{a+}, \mathbf{T}^{a-}])$ ;
10  $[e^a, \mathbf{e}^{a+}, \mathbf{e}^{a-}] \leftarrow \mathcal{M}_s([o^a, \mathbf{o}^{a+}, \mathbf{o}^{a-}])$ ;
11  $[\widehat{\mathbf{e}}^{a+}], [\widehat{\mathbf{e}}^{a-}] \leftarrow \mathcal{M}_d([o^a, \mathbf{o}^{a+}], \mathcal{M}_d([o^a, \mathbf{o}^{a-}])$ ;
12 using Formula 10,11,13 to compute  $\mathcal{L}_s^a, \mathcal{L}_d^a, \mathcal{L}_n^a$ ;
13 while  $\sum_a \mathcal{L}_s^a > \gamma \sum_a \mathcal{L}_d^a$  do
14    $\theta_d \leftarrow \mathbf{BP}(\sum_a \mathcal{L}_d^a)$ ;
15  $\theta_b, \theta_s \leftarrow \mathbf{BP}(\sum_a \mathcal{L}_s^a + \sum_a \mathcal{L}_n^a)$ ;

```

Next, we can leverage these weights to compute the *single-T* loss \mathcal{L}_s^a and the *dual-T* loss \mathcal{L}_d^a for the given trajectory T^a .

(1) *Computing \mathcal{L}_s^a .* Initially, we employ the function $\frac{1}{\exp(\cdot)}$ to normalize all distances, mitigating the effects of significant distance variance. Subsequently, the squared error is used to ascertain the loss. Furthermore, to augment the model's proficiency in distinguishing between distances of negative trajectory pairs, the $\text{ReLU}(\cdot)$ function is utilized. This ensures that the estimated distance between T^a and any negative trajectory T_i^{a-} surpasses the ground truth value $f(T^a, T_i^{a-})$. Consequently, the *single-T* loss \mathcal{L}_s^a for trajectory T^a can be expressed as:

$$\mathcal{L}_s^a = \mathcal{L}_s^{a+} + \mathcal{L}_s^{a-}$$

$$\mathcal{L}_s^{a+} = \sum_{i=1}^{\eta} \omega_i \left(\frac{1}{\exp(\sqrt{\|e^a - e_i^{a+}\|^2})} - \frac{1}{\exp(f(T^a, T_i^{a+}))} \right)^2 \quad (10)$$

$$\mathcal{L}_s^{a-} = \sum_{i=1}^{\eta} \omega_i \left(\text{ReLU}\left(\frac{1}{\exp(\sqrt{\|e^a - e_i^{a-}\|^2})} - \frac{1}{\exp(f(T^a, T_i^{a-}))}\right) \right)^2$$

(2) *Computing \mathcal{L}_d^a .* We use the norm value of the pair embedding $\widehat{e}^{a+/-}$ to measure the distance between T^a and $T^{a+/-}$. Similarly, the *dual-T* loss \mathcal{L}_d^a for trajectory T^a could be formulated as follows:

$$\mathcal{L}_d^a = \mathcal{L}_d^{a+} + \mathcal{L}_d^{a-}$$

$$\mathcal{L}_d^{a+} = \sum_{i=1}^{\eta} \omega_i \left(\frac{1}{\exp(\sqrt{\|\widehat{e}_i^{a+}\|^2})} - \frac{1}{\exp(f(T^a, T_i^{a+}))} \right)^2 \quad (11)$$

$$\mathcal{L}_d^{a-} = \sum_{i=1}^{\eta} \omega_i \left(\text{ReLU}\left(\frac{1}{\exp(\sqrt{\|\widehat{e}_i^{a-}\|^2})} - \frac{1}{\exp(f(T^a, T_i^{a-}))}\right) \right)^2$$

Instruction Loss. For illustrative purposes and without loss of generality, we use positive trajectory pairs to elucidate the concept of the instruction loss. Given a positive pair (T^a, T_i^{a+}) , the intent behind this loss is to harness the difference embedding \widehat{e}_i^{a+} , as produced by the high-precision *dual-T* model, to calibrate the actual embedding difference $(e^a - e_i^{a+})$ derived from the *single-T* model. To achieve this goal, we first propose the metric $\mathcal{D}_* = \lambda \mathcal{D}_{MSE} + (1 - \lambda) \mathcal{D}_{\cos}$ to evaluate the vector difference, where $\mathcal{D}_{MSE}(\alpha, \beta) = (\alpha - \beta)^2$ is used to

TABLE I: Trajectory Dataset

Dataset	Porto	Chengdu
Trajectories	1.49M	5.22M
Points	80.53M	1.05B
Longitude Range	13.89W-6.06W	104.04E-104.13E
Latitude Range	39.12N-41.94N	30.65N-30.72N
#Avg. of Trajectory Points	54.05	202.01
#Max. of Trajectory Points	3881	6801
#Std. of Trajectory Points	44.35	138.79

measure the vector norm difference, and $\mathcal{D}_{\cos}(\alpha, \beta) = \frac{\alpha \cdot \beta}{\|\alpha\| \cdot \|\beta\|}$ is used to measure the vector direction difference, and λ is a hyperparameter controlling each contribution. In addition, we propose an enhanced weight ϵ_i for each pair (T^a, T_i^{a+}) as follows:

$$\epsilon_i = \omega_i \times \mathfrak{I}(e^a - e_i^{a+}) \times I[\mathfrak{I}(e^a - e_i^{a+}) > \mathfrak{I}(\hat{e}_i^{a+})]$$

$$\mathfrak{I}(e^a - e_i^{a+}) = \left(\frac{1}{\exp(\sqrt{\|e^a - e_i^{a+}\|^2})} - \frac{1}{\exp(f(T^a, T_i^{a+}))} \right)^2 \quad (12)$$

$$\mathfrak{I}(\hat{e}_i^{a+}) = \left(\frac{1}{\exp(\sqrt{\|\hat{e}_i^{a+}\|^2})} - \frac{1}{\exp(f(T^a, T_i^{a+}))} \right)^2$$

where ω_i is defined in Formula 9, $\mathfrak{I}(\cdot)$ measures the distance between the actual similarity and the corresponding vector's norm, and $I[\cdot]$ is Iverson Bracket [33], outputting 1 if the condition within the brackets is satisfied, and 0 otherwise. For instance, should the condition $\mathfrak{I}(e^a - e_i^{a+}) > \mathfrak{I}(\hat{e}_i^{a+})$ hold, it signifies that the *dual-T* model yields a superior embedding for T_i^{a+} . Consequently, this necessitates the activation of ϵ_i , thereby permitting the *dual-T* to instruct and enhance the *single-T*. Conversely, in scenarios where the *single-T* is outperformed by the *dual-T*, the application of $I[\cdot]$ to adjust its value to 0 becomes imperative. Then, we can get the instruction loss \mathcal{L}_n^{a+} for the positive trajectory pair as $\mathcal{L}_n^{a+} = \sum_{i=0}^{\eta} \epsilon_i \mathcal{D}_* \left((e^a - e_i^{a+}), \hat{e}_i^{a+} \right)$. Similarly, we can use the same method to obtain \mathcal{L}_n^{a-} for the negative trajectory pair as $\mathcal{L}_n^{a-} = \sum_{i=0}^{\eta} \epsilon_i \mathcal{D}_* \left((e^a - e_i^{a-}), \hat{e}_i^{a-} \right)$. At last, we can derive the total instruction loss as follows:

$$\mathcal{L}_n^a = \mathcal{L}_n^{a-} + \mathcal{L}_n^{a+} \quad (13)$$

To this end, we illustrate the whole pre-training and fine-tuning process in Algorithm 1.

VI. EXPERIMENTS

In this section, we evaluate the effectiveness, efficiency, scalability, and discrimination ability of our model **DTisT**.

A. Experimental Setup

Dataset: Our experimental evaluations are conducted utilizing two publicly available trajectory datasets collected from distinct cities: Porto [34] and Chengdu [35]. Employing the preprocessing methodology outlined in [20], we obtain the refined dataset, the specifics of which are summarized in Table I. For each dataset, we randomly split these data into 3 parts by 6:2:2 for training, validation, and testing.

Baseline methods: We compare our model with four trajectory similarity embedding methods:

(1) **trajCL** [18] incorporates contrastive learning into trajectory similarity embedding, performing contrastive learning pre-training by constructing contrastive learning samples with several trajectory augmentation methods. In particular, we use the version with the best performance, where all layers of the encoder are fine-tuned.

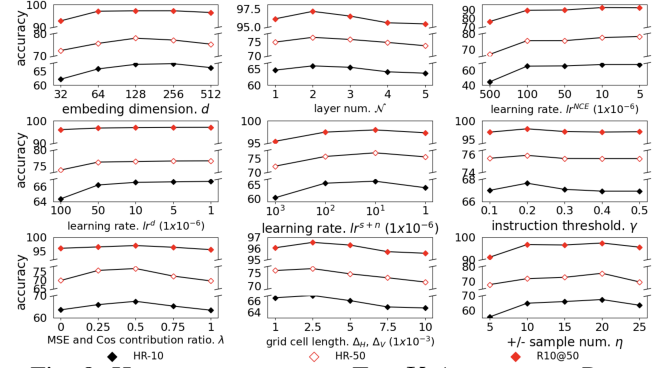


Fig. 8: Hyper-parameters vs. Top-K Accuracy on Porto.

(2) **trajGAT** [19] constructs a quadtree for trajectories, transforming the sequence problem into a graph problem, and embeds the trajectories using Graph Attention Networks.

(3) **Neutraj** [20] builds a massive information matrix based on the spatial arrangement of all points, and constructs an RNN network with an attention mechanism based on neighboring points.

(4) **T3S** [21] decouples the attention mechanism of the information matrix mentioned above from the RNN network, and separately aggregates the trajectories after pooling at the point level and processing with LSTM at the sequence level.

(5) **Traj2SimVec** [22] utilizes the RNN network to encode trajectory information and leverages the similarity information between sub-trajectories to aid the similarity training process, thereby enhancing the accuracy of similarity calculations.

Evaluation Metrics. We utilize four prominent trajectory distance functions: **DTW** [10], **Fréchet** [12], **ERP** [13], and **Hausdorff** [11], to calculate the ground truth similarity. To ensure a fair comparison, we adhere to the evaluation methodology established in prior studies such as [17]–[21], involving top- K similarity search. The evaluation employs key metrics, namely HR-5, HR-10, HR-50, R1@5 and R10@50. HR- K represents the top- K hitting rate, signifying the proportion of overlap between ground truth top- K and learned top- K . $RK_1@K_2$ denotes the recall of learned top- K_2 for the top- K_1 ground truth. Unless otherwise stated, the accuracy values mentioned in this section are expressed as percentages, where higher values indicate superior performance.

Environment settings. All methods are implemented using the PyTorch framework and optimized using Adam [36]. The computational platform run on Ubuntu 18.04.6 OS, employing an Intel Xeon Silver 4216 CPU and an NVIDIA A40 GPU.

B. Settings of Important Hyper-parameters

We primarily measure the following three aspects of hyper-parameters: (1) Model-related parameters, including the dimension of embedding d and the network layer \mathcal{N} ; (2) Hyper-parameters related to model learning, such as several learning rates during the training process ($lr^{NCE} : \mathcal{L}_{NCE}, lr^d : \mathcal{L}_d, lr^{s+n} : \mathcal{L}_n + \mathcal{L}_s$), the performance threshold γ that the dual-T model needs to achieve before entering the instruction phase, as well as the ratio λ between the MSE Loss and cosine similarity Loss in \mathcal{L}_n ; (3) Data-related parameters, such as the granularity of grid-cell division Δ_H and Δ_V , and the count of sampled distant and nearby trajectories considered for a given

TABLE II: Accuracy Performance Comparison with Different Similarity Measures.

Dataset	Method	DTW					Fréchet					ERP					Hausdorff				
		HR-5	HR-10	HR-50	R1@5	R10@50	HR-5	HR-10	HR-50	R1@5	R10@50	HR-5	HR-10	HR-50	R1@5	R10@50	HR-5	HR-10	HR-50	R1@5	R10@50
Porto	trajCL	29.43	32.80	35.78	40.18	55.91	40.91	45.30	61.30	64.62	88.44	15.74	22.27	35.31	25.88	52.28	23.06	27.30	28.52	30.77	48.67
	trajGAT	28.41	32.19	41.95	47.43	68.71	15.36	19.10	26.17	26.34	44.34	15.91	22.75	31.72	26.72	58.61	47.94	55.50	65.97	67.57	92.09
	Neutraj	28.66	32.74	38.09	46.10	65.67	35.71	43.35	52.64	55.25	80.21	43.38	48.66	58.43	65.83	85.09	18.30	21.19	28.71	27.25	55.80
	T3S	12.31	20.43	27.31	21.92	45.74	16.02	19.53	24.69	26.91	45.61	12.13	18.71	22.15	19.38	39.67	17.15	19.48	24.84	26.53	43.32
	Traj2SimVec	21.37	25.17	31.37	26.65	50.36	10.92	13.83	26.29	18.75	43.68	13.74	19.86	24.41	22.68	39.72	7.66	9.34	16.75	12.43	25.85
	sT	45.64	48.61	58.77	70.55	89.25	43.09	50.14	61.13	62.83	91.38	40.88	45.82	55.25	63.15	86.54	48.02	55.89	63.24	67.75	93.21
	DTisT	51.92	58.67	70.78	76.35	95.45	49.78	56.45	66.60	68.89	94.35	57.64	62.56	71.68	80.65	96.21	57.01	62.15	71.72	79.53	96.60
Chengdu	trajCL	39.77	41.50	53.25	60.51	77.06	49.93	55.91	69.44	72.63	93.71	29.81	34.20	48.53	47.32	71.94	46.16	52.21	67.60	66.41	91.00
	trajGAT	27.69	35.38	38.73	43.05	76.06	38.97	43.47	45.13	57.54	86.90	20.72	24.85	30.64	36.95	63.66	65.85	68.67	75.42	91.07	97.30
	Neutraj	44.38	48.26	63.91	65.33	91.09	40.38	45.16	50.02	59.49	89.15	63.87	66.99	73.29	89.27	97.29	51.31	58.04	71.32	73.18	93.91
	T3S	30.03	35.95	48.26	48.53	74.00	33.21	39.36	43.23	50.25	68.73	37.69	43.42	46.91	61.35	71.80	27.98	33.78	35.70	41.73	63.14
	Traj2SimVec	40.49	43.31	55.56	60.55	79.68	42.74	48.68	60.83	62.35	87.98	52.93	57.72	61.94	77.83	88.38	22.45	27.02	37.58	34.72	53.36
	sT	55.91	60.62	66.37	79.20	94.83	59.96	64.27	69.65	81.45	95.62	46.08	59.51	64.28	85.53	93.61	53.38	59.68	68.51	75.82	95.27
	DTisT	63.09	68.23	77.24	85.88	98.18	62.61	67.16	72.38	86.05	97.93	67.35	70.04	76.80	91.83	98.18	66.70	68.97	76.07	91.63	97.62

trajectory, denoted as η . In particular, given a hyper-parameter, we first select its value range according to the experience under some constraints (e.g., the limitation of GPU memory). Then, we conduct experiments on the validation of Porto to determine its optimal value. All results are shown in Fig. 8. Notably, we select the default setting of all hyper-parameters for the optimal setting as follows: $d = 128$, $\mathcal{N} = 2$, $lr^{NCE} = 1 \times 10^{-4}$, $lr^d = 5 \times 10^{-5}$, $lr^{s+n} = 1 \times 10^{-5}$, $\gamma = 0.2$, $\lambda = 0.5$, $\Delta_H/\Delta_V = 2.5 \times 10^{-3}$ and $\eta = 20$.

Guideline. \mathcal{N} and d control the model size. It is advisable to appropriately increase both parameters to enhance accuracy, provided that GPU memory constraints and inference latency are not issues. Additionally, these two parameters should be matched to the scale of the training data to avoid overfitting. Δ_H and Δ_V are responsible for determining the grid size. For datasets with densely packed trajectory points, reducing these values can improve embedding precision. In contrast, for datasets with sparsely distributed trajectory points, increasing these values can reduce GPU memory usage and expedite both training and inference processes. The sampling number η should be adjusted according to the scale of the training dataset. Augmenting the number of positive and negative samples is beneficial in large datasets to avoid underfitting.

C. Comparison with Baselines

1) *Effectiveness Comparison with Baselines:* To compare **DTisT** with existing trajectory similarity embedding methods, we conduct comprehensive experiments using two distinct trajectory datasets: Porto and Chengdu. The experimental results are summarized in Table II¹, with values presented in percentages. The results unequivocally demonstrate that²:

(1) **DTisT** outperforms the baseline across a majority of metrics. For instance, considering the DTW distance metric on the Porto dataset, **DTisT** achieves an accuracy almost twice as high as the strongest baseline, **Neutraj**, in the HR-10 metric. In other metrics, **DTisT** consistently exhibits a significant performance gap compared to the baseline. This robust performance across multiple metrics underscores the general effectiveness of **DTisT**.

¹Here **sT** refers to only *single-T* without instruction by *dual-T*, serving as a supplement to the ablation **A** in Sec. VI-E

²Unlike existing baseline methods, which typically exclude trajectories exceeding 100 points, our baseline comparisons forego such biased filtering using the full original data.

(2) The accuracy metrics of baseline methods noticeably decrease, often by at least one-third, when faced with complex datasets, especially Porto. However, **DTisT** showcases remarkable robustness in handling these complex datasets. The degradation in accuracy, at most only an eighth, underscores the robustness of **DTisT** and the significant performance gap it achieves compared to the baselines.

(3) When comparing these different methods horizontally taking Hausdorff as an example, unlike a cumulative distance algorithm (DTW), the Hausdorff distance depends only on the distance between a single point on each trajectory. The difficulty of embedding trajectories is relatively simple for Hausdorff. On the Chengdu dataset, the **DTisT** accuracy of Hausdorff doesn't show a huge advantage with the strongest baseline. Possibly because of data regularity and task simplicity. Both are already close to the theoretical upper limit for single-trajectory embedding tasks, indicating a bottleneck in single-trajectory embedding. Nevertheless, in complex data and with complex distance metrics, as both data and measurement methods become more intricate, the advantages of **DTisT** become increasingly pronounced.

2) Efficiency Comparison with Baselines:

Training and inference efficiency. As illustrated in Table III-(a), **DTisT** showcases a notable advantage in memory efficiency during the training phase. This can be attributed to the trajectory points to trajectory node aggregation employed in the adaptable base encoder. By compressing long sequences into shorter ones prior to passing through the high-memory-consuming Transformer network, **DTisT** effectively mitigates memory overhead, resulting in substantial memory savings.

Similarity retrieval efficiency. We compare the time taken by the Embedding Method and the Pairwise Method to retrieve the ten most similar trajectories for a given trajectory across different trajectory numbers³. As indicated in Table III-(b), the time overhead for using the Pairwise Method is particularly significant and increases linearly with the number of trajectories. Unfortunately, this latency does not support real-time tasks, especially for methods like Fréchet. In contrast, the Embedding Method not only significantly reduces end-

³Considering that the Embedding Method performs approximate retrieval, we first retrieve its top 50 most similar trajectories and then conduct an accurate calculation among the 50 trajectories for fair comparison (their accuracy in R10@50 consistently exceeds 95% in Table II).

TABLE III: Efficiency Comparison

	DTisT	trajCL	trajGAT	Neutraj	T3S	Traj2SimVec
training memory(GB)	22.19	41.65	24.72	41.34	27.59	27.23
training time(min)	12.19	14.62	6.95	16.34	9.57	6.09
infer mem(MB)	2227	1729	1865	4167	3709	2511
infer time(ms)	41.7	15.6	995.6	367.8	37.5	220.6

(a) Training and inference efficiency

numbers	Embedding Method			Pairwise Method			
	DTisT	trajCL	T3S	DTW	Fréchet	ERP	Hausdorff
1k	0.07s	0.04s	0.06s	0.88s	310.90s	2.50s	9.83s
10k	0.21s	0.18s	0.20s	8.44s	3058.10s	24.79s	97.67s
100k	1.63s	1.61s	1.62s	85.27s	30260.39s	244.06s	986.31s

(b) Top-10 search time across different trajectory number

average length	Embedding Method			Pairwise Method			
	DTisT	trajCL	T3S	DTW	Fréchet	ERP	Hausdorff
100 (points)	4.92ms	7.02ms	4.63ms	1.1ms	653.6ms	2.7ms	10.7ms
500 (points)	5.74ms	9.15ms	6.15ms	0.02s	24.27s	0.06s	0.25s
1000 (points)	6.81ms	14.29ms	8.42ms	0.19s	197.34s	1.49s	0.38s

(c) Similarity calculation time across different trajectory length

to-end latency but also achieves sub-linear time complexity. This efficiency gain is due to the simpler nature of similarity computations in the Embedding Method compared to the Pairwise Method. Furthermore, as the number of trajectories increases, the impact of model inference time on end-to-end latency (similar trajectory search) becomes almost negligible. As the number increases, the computation of distances between vectors emerges as the primary time-consuming element. Therefore, under the current size of the trajectory database [37], **DTisT** and other Embedding Methods tend to converge in terms of efficiency.

Long trajectory efficiency. In this paper, a long trajectory denotes a trajectory comprising a substantial number of points. The complexity of similarity computation correlates directly with the number of points. Furthermore, the inference latency, memory usage, and the search time are influenced by the length of this input sequence. Table III-(c) illustrates the computation time across various methods at differing trajectory lengths. As trajectory lengths extend, **DTisT** exhibits significant advantages relative to alternative approaches. This benefit arises from the long-term and short-term perception within **DTisT**, which enables the model to non-destructively shorten the sequence length at an early stage, thereby enhancing computational speed.

3) *Discrimination Ability Comparison with Baselines:* To analyze the discriminative effect of different methods on near or far trajectory embedding, we utilize t-SNE [38] to visualize the distribution of embeddings. Specifically, we randomly sample 1000 trajectories from Chengdu and Porto and applied t-SNE to reduce the dimensions of their embedding vectors obtained from both **DTisT** and baseline methods, resulting in two-dimensional points. Subsequently, we employ DBSCAN [39] with DTW and Fréchet distance to cluster these trajectories. The clustering results will be displayed on the two-dimensional points with different colors. Trajectory points in the same class imply smaller distances between them and should be close to each other. On the other hand, there should be a clear boundary between different classes, indicating larger distances. According to Fig. 9, we have the following observations:

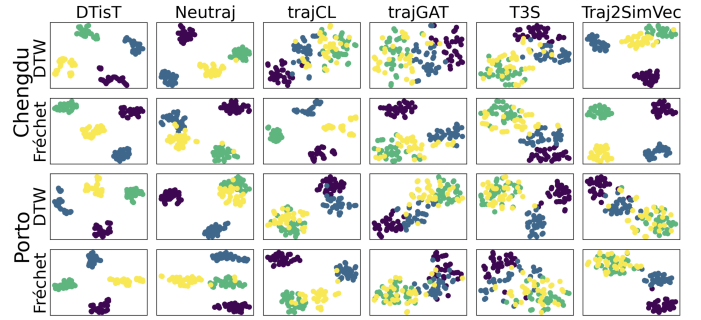


Fig. 9: Visualization of t-SNE Embeddings.

(1) For most **T3S**, **trajGAT** and **trajCL**, there is significant overlap between embeddings of different categories, and no clear boundaries. These models leverage an attention mechanism that prioritizes intra-sequence associations over inter-sequence differences, leading to suboptimal performance in recognizing similarities between trajectories. As a result, these models exhibit limited discriminative effectiveness.

(2) Compared to **T3S** and **trajCL**, **Neutraj** and **Traj2SimVec** show superior performance. However, a closer examination of trajectory distributions within the same grid reveals that instances of two categories overlapping still sporadically occur. These models only embed single trajectories in isolation, lacking dual-trajectory assistance and overlooking the intrinsic nature of similarity computation as an inter-trajectory task, thereby encountering a performance bottleneck.

(3) **DTisT**'s embeddings exhibit a linear separation between different categories. Moreover, trajectories within the same category form a tight cluster. These observations suggest that **DTisT** effectively generates discriminative features. By distilling knowledge from a single-T model into a dual-T framework and integrating short and long-term trajectory perception, this approach facilitates precise similarity computations between trajectories and produces high-quality embeddings, leading to superior performance over existing state-of-the-art models.

(4) When comparing the effectiveness of the same embedding method across different datasets and similarity metrics, it is observable that **trajCL** and **Traj2SimVec** show significantly better results on the Fréchet distance of Chengdu, consistent with the accuracy presented in Table II. It is not difficult to conclude that categories with higher accuracy in Table II tend to also perform better in clustering results, demonstrating consistency in effectiveness between the two downstream tasks of trajectory clustering and similar trajectory search.

4) *Scalability Comparison with Baselines:* To compare the scalability, we train different models by varying the training data size. In particular, we sample 20%, 40%, 60%, 80% and 100% from the training data of Porto with the DTW distance and Chengdu with the Fréchet distance, and then collect their accuracy of approximate retrieval on the each test data. From Fig. 10, we have the following observations:

(1) All methods show a decrease in model performance as the training data size decreases because larger training data allows the model to learn more diverse data scenarios.

(2) **DTisT** is more effective and stable compared to other baseline methods. Taking HR-10 accuracy as an example, even

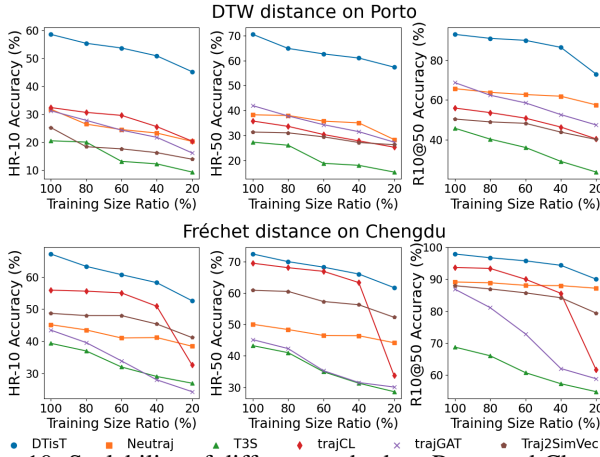


Fig. 10: Scalability of different method on Porto and Chengdu.

when using only 20% of the data, it outperforms the training results of most other models using 100% of the data.

(3) **trajCL** has a notable performance decline at 20% data coverage in Chengdu. This phenomenon is attributed to its reliance on simple trajectory augmentation techniques during pre-training, which predisposes it to overfitting in scenarios of limited data availability. In contrast, our approach **DTisT** implements a supervised pre-training methodology that enhances data quality through the asymmetric sampling of positive and negative trajectories. This strategy effectively captures correlations among diverse trajectories, thereby addressing the issue of overfitting in environments with limited data.

D. Long Trajectories Study vs. trajGAT

In order to assess the effectiveness and efficiency of **DTisT** on long trajectories, we select the **trajGAT** model, specifically designed for long trajectories, as a reference. The method **trajGAT** constructs a tree-like aggregation structure for trajectories using its well-designed quadtree structure. It efficiently aggregates information from trajectory points using the graph attention mechanism from graph neural networks, achieving high efficiency on long trajectories. We randomly sample trajectories from the Porto dataset to create five datasets with average trajectory lengths of approximately 100, 250, 500, 750, and 1000. We then train and validate these datasets separately, recording their training performance and efficiency. (In the case of **trajGAT** on the 1000-length trajectory dataset, an ‘*cuda out of memory*’ situation occurred, exceeding 48GB. Due to hardware limitations, we cannot show its results on this specific dataset.) The results are depicted in the Fig. 11, and we have the following observations:

- (1) In terms of accuracy, as trajectories become longer, both **DTisT** and **trajGAT** exhibit some degradation in performance. However, **DTisT** maintains a significant lead, demonstrating its clear advantage when dealing with long trajectories.
- (2) In evaluating efficiency, we conduct a comparison based on training times and memory usage, accounting for the increase in training time and memory usage as trajectories lengthen. **DTisT** demonstrates a substantial advantage in GPU memory conservation, attributed to the implementation of grid cell trajectory aggregation.

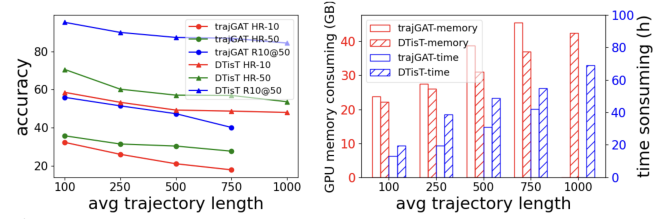


Fig. 11: The Effectiveness and Efficiency on Long Trajectories.

TABLE IV: Ablation Result.

	⓪	Ⓐ	Ⓑ	Ⓒ	Ⓓ	Ⓔ	Ⓕ	Ⓖ
Dual-T Instruction	✓	×	✓	✓	✓	✓	✓	✓
Point-pair Supervised	✓	×	×	✓	✓	✓	✓	✓
Synthetic Trajectory	✓	✓	×	×	✓	✓	✓	✓
Contrastive Learning	✓	✓	✓	✓	×	✓	✓	✓
Grid-cell Aggregation	✓	✓	✓	✓	✓	×	✓	✓
Divergent Sampling	✓	✓	✓	✓	✓	✓	×	✓
Pre-Training	✓	✓	✓	✓	✓	✓	✓	×
HR-10 (%)	58.67	48.61	54.76	52.25	51.60	49.71	29.30	53.73
HR-50 (%)	70.78	58.77	63.75	59.63	61.54	60.47	41.27	62.97
R10@50 (%)	95.45	89.25	92.22	90.11	89.72	88.86	66.77	91.89

E. Ablation Studies

As shown in Table IV, we conduct extensive ablation studies on Porto with the DTW distance to evaluate the effectiveness of key designs in **DTisT**. To validate the effectiveness of the innovative aspects we propose, we conduct the following ablation experiments to assess the core innovations in our method. The ⓪ represents the experimental group which is the whole **DTisT** framework.

- (1) Dual-T Instruction (Ⓐ): To thoroughly validate the effectiveness of our proposed *dual-T* module for instructing the *single-T* module, we conduct an in-depth examination to ascertain whether the single trajectory embedding model can indeed glean more semantic information from the dual trajectory model during similarity calculation. In particular, we tailor the instruction process to solely self-train the single trajectory model. It is important to emphasize that in this altered setup, the *dual-T* module becomes non-functional, rendering the supervision of point-pair information for the *dual-T* model ineffective.
- (2) Matched Point-pair Supervision (Ⓑ): In Sec. IV-B, to examine the supervisory effect of matched point-pairs between trajectories, we conduct the experiment Ⓑ without Point-pair. In this experiment, the attention mask matrix of the dual-T model was set to the default, eliminating the information gained from point-pair information for the model. From the experimental results, it is evident that the inclusion of point-pair information significantly improves the model’s accuracy.
- (3) Synthetic Trajectory (Ⓒ): In Sec. IV-C, to achieve symmetry between the dual-T model and single-T model, we introduce a synthetic trajectory into the single-T model. Here, we remove this virtual trajectory to evaluate the synthetic trajectory’s effectiveness. Experimental results indicate that the inclusion of the synthetic trajectory significantly enhances model performance. Symmetric dual-T and single-T models enable single embedding space and dual embedding space to be congruent, facilitating easy instruction processing and relaxing triangle inequality constraints.
- (4) Contrastive Learning in the Pre-training Phase (Ⓓ): In the realm of model training, leveraging a pre-training process often leads to superior models compared to training from

scratch. We conduct an analysis focusing on the effects of contrastive learning pre-training in contrast to pre-training solely with similarity label supervision. It is apparent that contrastive learning pre-training provides a better initial state for the model through unlabeled positive and negative trajectory pairs, enabling it to gain greater robustness.

(5) Sub-trajectory Node (\mathbb{E}): To assess the impact of aggregating points within grid cells into nodes, as detailed in Sec. IV-A (Long-Short-term Perception), we conduct a comparative study. Specifically, we remove this aggregation node filter and its sub-trajectory embedding enabling the direct mapping from point-level to trajectory-level. The results indicate that the sub-trajectory aggregation associates point-level and trajectory-level from a local perspective, resulting in a more accurate and refined representation.

(6) Divergent Sampling (\mathbb{F}): To evaluate the effect of Divergent Sampling in Sec. V-A, we establish symmetrical sampling approach for positive and negative examples, selecting nearest η and the farthest η trajectories from the anchor. It is evident that probabilistic sampling of dissimilar trajectories allows the model to perceive trajectories from more categories, significantly enhancing model performance.

(7) Pre-training (\mathbb{G}): In our method, both single-T and dual-T share an Adaptable Base Encoder layer. To avoid inherent instabilities in the instruction loss between dual-T embedding and single-T embedding, we incorporate the pre-training technique before fine-tuning both single-T and dual-T modules. To verify the effectiveness of the pre-training process, we transformed this two-stage task into a multi-task learning experiment, referencing [40]. Comparative experiments, denoted as \mathbb{O} , demonstrate that the two-stage training of single-T and dual-T, by achieving better initial parameters, ultimately yields superior results.

VII. RELATED WORK

A. Trajectory Representation Learning for Optimizing Similarity Computation

Learning-based approximation methods endeavor to model trajectory similarity calculations using neural networks. For instance, the authors in [41] devised a method to make DTW distance differentiable through heuristic rules and integrated it into loss functions for efficient backpropagation. Another work [17] introduced a measure-general neural network employing an attention mechanism to simulate the matching operation of similarity calculations, achieving a universal high-precision similarity approximation. However, these methods, primarily categorized as pairwise learning methods due to their utilization of two trajectory inputs, solely address the challenges of the computing phase. To concurrently address retrieve phase challenges, trajectory embedding learning for similarity computation emerged as a promising approach. This technique involves mapping trajectories or trajectory segments into high-dimensional vectors and aggregating trajectory semantic information to enhance downstream task effectiveness. Consequently, the search phase is transformed into a vector retrieval problem, which can be significantly optimized, potentially up to a logarithmic level. The pioneering work of

[42] related trajectory embedding vectors in the representation space to trajectory similarity by measuring similarity based on the magnitude of distance in the representation space. Another representative method, [20], partitioned the map into grids and saved hidden embeddings for each region. However, it faced challenges with accuracy, particularly with longer trajectories. To counter these issues, [19] proposed an approach based on GNN to encode trajectories. They pre-segmented trajectories into quadtree-based partitions based on their positions, enabling positional information to be treated as a graph structure. In addition, other methods [43], [44] incorporate road network information to denoise traffic trajectory data, prioritizing performance in downstream tasks over achieving higher precision. Thus, these methods apply exclusively to traffic trajectory data, serving specific downstream tasks.

However, existing embedding-based learning methods have predominantly focused on the embedding process for single trajectories, overlooking the accuracy advantages provided by Pairwise Learning in simulating similarity calculations, the non-metric nature of the embedding space, and the potential alignment information that can be optimized and supervised in pairwise-based learning. This oversight has led to shortcomings in terms of effectiveness.

B. Trajectory Representation Learning for Other Tasks

Trajectory data serves diverse downstream tasks like Time Estimation, Prediction, Clustering, and more [45], [46]. Incorporating trajectory representation learning has significantly advanced these tasks. For instance, Jiang et al. [47] integrate travel semantics and road segments into trajectory embedding, enabling precise travel time prediction. Similarly, Suris et al. [48] model partial trajectory observations as probability distributions, enhancing accurate future segment predictions. In clustering, Hoseini et al. [49] embed both transitive and non-transitive relationships between trajectories, achieving robust trajectory clustering. Trajectory embedding aids autonomous driving [50], identifying unusual scenarios based on trajectory density distributions. Trajectory representation has found utility in tour recommendation [51]. Chen et al. [51] blend spatio-temporal trajectory features with Markov chains, enhancing Point-of-Interest recommendations. Beyond, trajectory representation enhances tasks like anomaly detection [52], destination prediction [53], and classification [54], [55] by injecting trajectory semantics into multi-modal data.

VIII. CONCLUSION

In this paper, we studied the problem of trajectory embedding for similarity computation. We proposed an effective and novel method called the Dual Trajectory similarity model Instruct the Single Trajectory similarity model **DTisT** which incorporates a novel framework using pairwise learning instructing embedding learning and introduces point pair ground truth to supervise its pairwise learning. Furthermore, we aggregate trajectories by grids to address the issue of long trajectories and further enhance its performance by incorporating contrastive learning pretraining. Extensive experiments on real datasets confirmed the effectiveness of our method.

REFERENCES

- [1] H. Yuan and G. Li, "Distributed in-memory trajectory similarity search and join on road network," in *ICDE*, 2019, pp. 1262–1273.
- [2] Z. Shang, G. Li, and Z. Bao, "DITA: distributed in-memory trajectory analytics," in *SIGMOD*, G. Das, C. M. Jermaine, and P. A. Bernstein, Eds., 2018, pp. 725–740.
- [3] H. Li, G. Li, J. Liu, H. Yuan, and H. Wang, "Ratel: Interactive analytics for large scale trajectories," in *SIGMOD*, 2019, pp. 1949–1952.
- [4] K. Buchin, M. Buchin, D. Duran, B. T. Fasy, R. Jacobs, V. Sacristan, R. I. Silveira, F. Staals, and C. Wenk, "Clustering trajectories for map construction," in *SIGSPATIAL*, 2017.
- [5] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, and X. Qin, "Fast large-scale trajectory clustering," *VLDB Endowment*, vol. 13, no. 1, p. 29–42, sep 2019.
- [6] H. Wu, W. Sun, and B. Zheng, "A fast trajectory outlier detection approach via driving behavior modeling," in *CIKM*, 2017, pp. 837–846.
- [7] Y. Liu, K. Zhao, G. Cong, and Z. Bao, "Online anomalous trajectory detection with deep generative sequence modeling," in *ICDE*, 2020, pp. 949–960.
- [8] M. Chen, H. Yuan, N. Jiang, Z. Bao, and S. Wang, "Urban traffic accident risk prediction revisited: Regionality, proximity, similarity and sparsity," in *CIKM*, 2024, pp. 281–290.
- [9] D. Xie, F. Li, and J. M. Phillips, "Distributed trajectory similarity search," *VLDB*, pp. 1478–1489, 2017.
- [10] B. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *ICDE*, 1998, pp. 201–208.
- [11] S. Atev, G. Miller, and N. P. Papanikolopoulos, "Clustering of vehicle trajectories," *TITS*, vol. 11, no. 3, pp. 647–657, 2010.
- [12] T. Eiter and H. Mannila, "Computing discrete fr chet distance," 1994.
- [13] L. Chen and R. T. Ng, "On the marriage of lp-norms and edit distance," in *VLDB*, 2004, pp. 792–803.
- [14] A. Backurs and A. Sidiropoulos, "Constant-Distortion Embeddings of Hausdorff Metrics into Constant-Dimensional l_p Spaces," in *APPROX/RANDOM*, 2016, pp. 1 : 1 – –1 : 15.
- [15] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir, "Computing the discrete fr chet distance in subquadratic time," in *SODA*, 2013, pp. 156–167.
- [16] P. K. Agarwal, K. Fox, J. Pan, and R. Ying, "Approximating dynamic time warping and edit distance for a pair of point sequences," in *SoCG*, 2016, pp. 6:1–6:16.
- [17] P. Yang, H. Wang, D. Lian, Y. Zhang, L. Qin, and W. Zhang, "TMN: trajectory matching networks for predicting similarity," in *ICDE*, 2022, pp. 1700–1713.
- [18] Y. Chang, J. Qi, Y. Liang, and E. Tanin, "Contrastive trajectory similarity learning with dual-feature attention," in *ICDE*, 2023, pp. 2933–2945.
- [19] D. Yao, H. Hu, L. Du, G. Cong, S. Han, and J. Bi, "Trajgat: A graph-based long-term dependency modeling approach for trajectory similarity computation," in *KDD*, 2022, pp. 2275–2285.
- [20] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," in *ICDE*, 2019, pp. 1358–1369.
- [21] P. Yang, H. Wang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "T3S: effective representation learning for trajectory similarity computation," in *ICDE*, 2021, pp. 2183–2188.
- [22] 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 *IJCAI*, 2020, pp. 3209–3215.
- [23] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *STOC*, 1998, pp. 604–613.
- [24] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *PAMI*, pp. 117–128, 2010.
- [25] Y. Chang, E. Tanin, G. Cong, C. S. Jensen, and J. Qi, "Trajectory similarity measurement: An efficiency perspective," *VLDB*, pp. 2293–2306, 2024.
- [26] M. Vlachos, D. Gunopulos, and G. Kollios, "Discovering similar multidimensional trajectories," in *ICDE*, 2002, pp. 673–684.
- [27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.
- [28] Ddist. [Online]. Available: <https://github.com/liujiaozhiren/DTisT>
- [29] M. Halawa, O. Hellwich, and P. Bideau, "Action-based contrastive learning for trajectory prediction," in *ECCV*, 2022, pp. 143–159.
- [30] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NeurIPS*, 2013, pp. 3111–3119.
- [31] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," 2019.
- [32] F. Wang and H. Liu, "Understanding the behaviour of contrastive loss," in *CVPR*, 2021, pp. 2495–2504.
- [33] K. E. Iverson, *A Programming Language*. John Wiley & Sons, 1962.
- [34] Kaggle, "Porto taxi trajectory dataset," <https://www.kaggle.com/competitions/pkdd-15-predict-taxi-service-trajectory-i/data>, 2015.
- [35] DiDi Research, "Didi gaia open dataset," <https://outreach.didichuxing.com/>, 2022.
- [36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.
- [37] W. Chen, Y. Liang, Y. Zhu, Y. Chang, K. Luo, H. Wen, L. Li, Y. Yu, Q. Wen, C. Chen *et al.*, "Deep learning for trajectory data management and mining: A survey and beyond," *arXiv*, 2024.
- [38] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne," *JMLR*, vol. 9, no. 11, 2008.
- [39] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*, 1996, pp. 226–231.
- [40] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," *NeurIPS*, 2020.
- [41] X. Cai, T. Xu, J. Yi, J. Huang, and S. Rajasekaran, "Dtwnet: a dynamic time warping network," in *NeurIPS*, 2019, pp. 11 636–11 646.
- [42] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *ICDE*, 2018, pp. 617–628.
- [43] Z. Fang, Y. Du, X. Zhu, D. Hu, L. Chen, Y. Gao, and C. S. Jensen, "Spatio-temporal trajectory similarity learning in road networks," in *SIGKDD*, 2022, pp. 347–356.
- [44] S. Zhou, J. Li, H. Wang, S. Shang, and P. Han, "Grlstm: trajectory similarity computation with graph-based residual lstm," in *AAAI*, 2023, pp. 4972–4980.
- [45] S. Wang, Z. Bao, J. S. Culpepper, and G. Cong, "A survey on trajectory data management, analytics, and learning," *CSUR*, pp. 39:1–39:36, 2022.
- [46] A. Graser, A. N. Jalali, J. Lampert, A. Weissenfeld, and K. Janowicz, "Deep learning from trajectory data: a review of deep neural networks and the trajectory data representations to train them," in *EDBT/ICDT*, 2023.
- [47] J. Jiang, D. Pan, H. Ren, X. Jiang, C. Li, and J. Wang, "Self-supervised trajectory representation learning with temporal regularities and travel semantics," in *ICDE*, 2023, pp. 843–855.
- [48] D. Suris Coll-Vinent and C. Vondrick, "Representing spatial trajectories as distributions," *NeurIPS*, pp. 13 731–13 744, 2022.
- [49] F. S. Hoseini, S. Rahrovani, and M. H. Chehreghani, "A generic framework for clustering vehicle motion trajectories," *arXiv*, 2020.
- [50] J. Wiederer, A. Bouazizi, M. Troina, U. Kressel, and V. Belagiannis, "Anomaly detection in multi-agent trajectories for automated driving," in *CoRL*, 2022, pp. 1223–1233.
- [51] D. Chen, C. S. Ong, and L. Xie, "Learning points and routes to recommend trajectories," in *CIKM*, 2016, pp. 2227–2232.
- [52] R. Jiao, J. Bai, X. Liu, T. Sato, X. Yuan, Q. A. Chen, and Q. Zhu, "Learning representation for anomaly detection of vehicle trajectories," *CoRR*, 2023.
- [53] L. Han, X. Ma, L. Sun, B. Du, Y. Fu, W. Lv, and H. Xiong, "Continuous-time and multi-level graph representation learning for origin-destination demand prediction," in *KDD*, 2022, pp. 516–524.
- [54] Y. Endo, H. Toda, K. Nishida, and J. Ikeda, "Classifying spatial trajectories using representation learning," *IJDSA*, pp. 107–117, 2016.
- [55] D. Yao, C. Zhang, Z. Zhu, J. Huang, and J. Bi, "Trajectory clustering via deep representation learning," in *IJCNN*, 2017, pp. 3880–3887.