Trajectory Similarity Measurement: An Efficiency Perspective

Yanchuan Chang
The University of Melbourne
yanchuanc@student.unimelb.edu.au

Egemen Tanin
The University of Melbourne
etanin@unimelb.edu.au

Gao Cong
Nanyang Technological University
gaocong@ntu.edu.sg

Christian S. Jensen
Aalborg University
csj@cs.aau.dk

Jianzhong Qi*
The University of Melbourne
jianzhong.qi@unimelb.edu.au

ABSTRACT

Trajectories that capture object movement have numerous applications, in which similarity computation between trajectories often plays a key role. Traditionally, trajectory similarity is quantified by means of non-learned measures, e.g., Hausdorff, that operate directly on the trajectories. Recent studies exploit deep learning to map trajectories to d-dimensional vectors, called embeddings. Then, some distance measure, e.g., Manhattan, is applied to the embeddings to quantify trajectory similarity. The resulting similarities are inaccurate: they only approximate the similarities obtained using the non-learned measures. As embedding distance computation is efficient, focus has been on obtaining embeddings of high accuracy.

Adopting an efficiency perspective, we analyze the time complexities of both the non-learned and the learning-based approaches, finding that the time complexities of the former approaches are not necessarily higher. Through extensive experiments on open datasets, we find that only a few learning-based approaches can deliver the promised higher efficiency, when the embeddings can be pre-computed, while non-learned approaches are more efficient for one-off computations. Among the learning-based approaches, the self-attention-based ones are the fastest and the most accurate. These results have implications for the use of trajectory similarity approaches given different application requirements.

PVLDB Reference Format:
Yanchuan Chang, Egemen Tanin, Gao Cong, Christian S. Jensen, and Jianzhong Qi. Trajectory Similarity Measurement: An Efficiency Perspective. PVLDB, 17(9): 2293 - 2306, 2024.
doi:10.14778/3665844.3665858

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

1 INTRODUCTION

A trajectory is a sequence of timestamped point locations that captures the movement of an object such as a vehicle or a person. The ability to quantify the similarity between two trajectories is essential in spatio-temporal data mining [28, 57, 66, 67, 73]. Due to the rich location and movement information encoded in trajectories and many application settings, no single universal trajectory similarity measure exists. Rather, different trajectory similarity measures have been proposed for different settings. These can be largely classified into two categories: *non-learned measures* and *learned measures*.

Early studies focus on non-learned measures [7, 8, 18, 19, 38, 54, 64]. They typically work by matching the points between two trajectories (see Figure 1) and are hand-crafted to capture similarity. For example, *Hausdorff* [7] computes a point matching that minimizes the sum of point-to-trajectory distances of two trajectories. Popular non-learned measures like the above have quadratic time complexity in the number of points on trajectories to examine. This complexity is considered a drawback in the literature [13, 21, 24, 76, 78, 84].

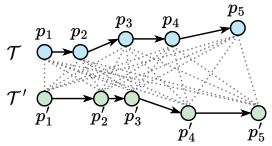


Figure 1: Computation of non-learned measures (dotted lines indicate point-to-point distance computation)

Learned measures [20, 29, 33, 75, 76, 78, 79] mainly aim to improve the computational efficiency by exploiting deep learning and have recently attracted substantial interest. They generally follow the steps: (1) encode trajectories as vectors (called *trajectory embeddings*), and (2) compute the vector distance (e.g., the Manhattan distance) between trajectory embeddings to serve as the trajectory distance (see Figure 2). For example, *t2vec* [43], *NEUTRAJ* [78], and *TMN* [75] use recurrent neural networks (RNNs) [22, 35], while *T3S* [76] and *TrajCL* [13] use self-attention models [62].

Among learned measures, some (e.g., NEUTRAJ and T3S) "learn" to *approximate existing non-learned measures* (e.g., Hausdorff), i.e., the training signals are the ground-truth trajectory similarities provided by non-learned measures. Such sacrifice in accuracy is expected to be rewarded by higher computational efficiency.

Another series of the learned measures (e.g., t2vec and TrajCL) improves not only the efficiency but also *the measurement effectiveness*. Such methods typically use self-supervised learning techniques to learn robust measures from unlabeled trajectories directly without relying on any non-learned measure. Once trained, they generally have better effectiveness, especially on measuring low-quality trajectories, than the non-learned ones.

A general perception of the learned measures in either category is that the learning-based approach yields superior efficiency, due to the simple vector format of embeddings and the highly optimized implementations of deep learning processes.

*Corresponding author
This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 17, No. 9 ISSN 2150-8097.
doi:10.14778/3665844.3665858

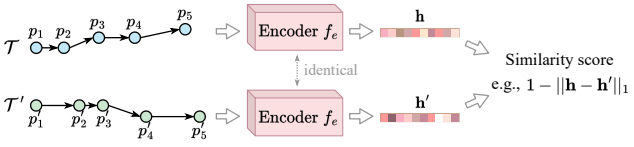


Figure 2: Computation of learned measures

However, we observe that the time complexities of the learned measures are not necessarily lower than those of the non-learned ones. For example, T3S [76] and TrajCL [13] also have quadratic time complexity in the number of trajectory points (covered in Section 3.2). Although previous studies [13, 21, 24, 75, 76, 78] report that the learned measures take less time to compute, they do not share detailed experimental settings, e.g., as to whether GPUs or CPUs are used for measuring trajectory similarity.

These observations prompt two questions: *(1) Do learned measures have better computation time and space efficiencies than non-learned measures? (2) How do the learned measures compare with each other in terms of accuracy given training time constraints, e.g., due to different application requirements?*

This study provides the first answers to these questions based on comprehensive experiments on the efficiency of the trajectory similarity measures, as well as the accuracy of the learned measures given training time constraints. This way, we aim to provide a foundation for the community to pursue promising research directions and to provide guidance on similarity measure selection.

We start by analyzing the time complexities of representative non-learned and learned measures and discuss their strengths and limitations (Section 3). We find that most of the existing learned measures do not necessarily have lower time complexity.

Then, we design three meta-algorithms for parallelizing the computation of non-learned measures (detailed in a technical report [5] due to the space constraint). We implement seven non-learned measures that leverage these algorithms to use CUDA streaming cores, thus enabling GPU-based empirical comparisons with learned measures. The meta-algorithms provide a comprehensive basis for future studies to implement new non-learned measures.

We compare the efficiency of both types of measures using GPUs and CPUs on real datasets in a variety of settings (Section 4). We use the measures for trajectory similarity computation, trajectory clustering [17, 40], and trajectory kNN queries [34, 73, 85]. Previous studies [43, 78, 79] use spatial indices like R-trees [10] for trajectory kNN queries, which do not fit the learned measures. We use a kNN query framework designed for high-dimensional vectors to better realize the potential of the learned measures. We also investigate the impact of training time constraints on the accuracy of kNN queries for the learned measures, to guide similarity measure selection.

To sum up, we make the following contributions:

(1) We review existing non-learned and learned spatio-temporal measures and analyze their time complexities, finding that the learned measures do not necessarily have the lowest efficiency.

(2) We report on an extensive evaluation of the efficiency of both types of measures on GPUs and CPUs, finding that: (i) The non-learned measures are most efficient for one-off computation (online matching of incoming trajectories or kNN queries with data updates, e.g., for ride-sharing). (ii) The learned measures are most efficient for offline trajectory clustering and kNN queries, or when the trajectory embeddings can be pre-computed and reused, although they only offer approximate results. (iii) Further, among the learned measures, the self-attention-based ones are the fastest to train and offer the highest accuracy for kNN queries.

(3) We cover a simulated experiment to show a learned measure that outperforms non-learned measures at one-off computation, and we provide future directions for achieving such a learned measure.

Several survey papers [58, 59, 65] cover non-learned measures, one of which [65] also mentions a few early studies of learned measures. None of these papers include a comprehensive comparison between the two types of measures, which is our focus.

2 PRELIMINARIES

Trajectory. A spatio-temporal trajectory $\mathcal{T} = [p_1, p_2, \ldots, p_n]$ is a sequence of n points, where point p_i is either given by a pair of coordinates (x_i, y_i) or a pair of timestamped coordinates (x_i, y_i, t_i). Trajectories without timestamps are also called paths.

Non-learned trajectory similarity measure. Given a trajectory dataset D, the similarity between two trajectories is defined by a function $f\colon D \times D \to \mathbb{R}_{\geqslant 0}$.

A non-learned measure $f_h\colon D \times D \to \mathbb{R}_{\geqslant 0}$, e.g., Hausdorff, is a handcrafted function to capture the similarity between two trajectories. Distance measures can easily be converted into similarity measures, and thus we also consider them as similarity measures.

Learned trajectory similarity measure. A learned measure f_l, e.g., T3S, involves a two-step process. First, an encoding function $f_e\colon D \to \mathbb{R}^d$, which is learned, is applied to map each trajectory into a d-dimensional embedding space. Second, the distance between the embeddings $\mathbf{h}$ and $\mathbf{h}'$ of trajectories $\mathcal{T}$ and $\mathcal{T}'$ is used to quantify the similarity between $\mathcal{T}$ and $\mathcal{T}'$, e.g., $f_l(\mathcal{T}, \mathcal{T}') = 1 - ||f_e(\mathcal{T}) - f_e(\mathcal{T}')||_1 = 1 - ||\mathbf{h} - \mathbf{h}'||_1$, using the Manhattan distance.

Trajectory similarity query. Given a trajectory dataset D, a query trajectory $\mathcal{T}_q$, a trajectory similarity measure f, and a positive integer k, a *trajectory similarity query* returns a set $S \subset D$ with $|S| = k$ such that $\forall \mathcal{T} \in S, \mathcal{T}' \in D \setminus S \;\; (f(\mathcal{T}_q, \mathcal{T}) \geqslant f(\mathcal{T}_q, \mathcal{T}'))$. This query is also called a *trajectory kNN query*.

Trajectory clustering. Given a trajectory dataset D and a trajectory similarity measure f, *trajectory clustering* groups the trajectories in D into subsets (i.e., clusters) based on their similarity.

Both non-learned and learned measures can be applied in trajectory similarity queries and clustering. A learned measure typically approximates some non-learned measures. The approximation error is referred to as the *inaccuracy* of the learned measure.

3 TRAJECTORY SIMILARITY MEASURES

Next, we cover existing studies on spatial / spatio-temporal *trajectory similarity measures*, including both *non-learned* and *learned* ones, and trajectory queries, including *trajectory similarity queries*, and *trajectory clustering*. Figure 3 summarizes the representative measures based on Euclidean space that we focus on. As the figure shows, the learned measures dominate the recent literature.

3.1 Non-learned Trajectory Similarity Measures

Non-learned measures are generally based on point matching [7, 8, 14, 18, 19, 54]. The similarity between two trajectories is derived from the distances between the matched point pairs. Based

Table 1: Categorization of representative trajectory similarity measures in Euclidean space

Category	Methodology	Measure	Time Complexity	Space Complexity
Non-learned measures	Linear scan	SPD [6], CDDS [14], SAX [9]	$O(n)$	$\Theta(1)$
	Dynamic programming	EDR [19], ERP [18], EDwP [54], LCSS [64], DTW [38], Discrete Fréchet [27], STEDR [60], STLCSS [60]	$O(n^2)$	$\Theta(n)$
	Enumeration	OWD [44], Hausdorff [7]	$O(n^2)$	$\Theta(1)$
Learned measures	Recurrent neural network	t2vec [43], NEUTRAJ [78], Play2vec [70], At2vec [46], Traj2SimVec [84], Tedjopurnomo [60], Chen [20], CL-Tsim [24], TMN [75], RSTS [21]	$\Omega(nd^2)$	$\Omega(d^2)$
	Self-attention neural network	T3S [76], CSTRM [47], TrajCL [13]	$\Omega(n^2d)$	$\Omega(d^2 + nd + n^2)$
	Convolutional neural network	TrjSR [12]	$\Omega(mk^2n_kc)$	$\Omega(k^2n_kc + mc)$
	Graph neural network	TrajGAT [79]	$\Omega(nn_ed)$	$\Omega(d^2 + nd + nn_e)$

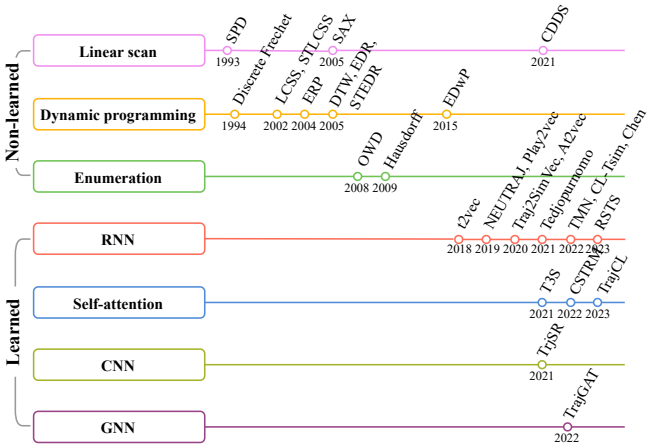


Figure 3: Representative trajectory similarity measures

on how the point matches are computed, we categorize the non-learned measures into three classes: (i) *linear scan-based*, (ii) *dynamic programming-based* and (iii) *enumeration-based* measures.

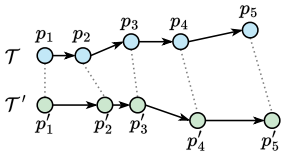


Figure 4: Computation of linear scan-based measures

Linear scan-based measures. Linear scan-based measures [6, 9, 14] take only a single scan over two trajectories (cf. Figure 4, where each gray dotted line denotes a pair of matched points). Such measures take $O(n)$ time to compute, assuming n points per trajectory, and $\Theta(1)$ space, to store the partial similarity results, excluding the $O(n)$ space to hold the trajectories (same below).

For example, *Sum-of-Pair Distance* (SPD) [6] simply matches the points on two trajectories following the order of the points and sums up the pairwise point distances. It assumes trajectories of the same number of points. *Close-Distance Duration Similarity* (CDDS) [14] further considers the time dimension. It scans the points on two trajectories and sums up the time span when the points on both trajectories are within a given spatial distance threshold. The resulting sum is used as the similarity. *Symbolic Aggregate Approximation* (SAX) representation, which was designed for time series [45], can be extended to compute spatio-temporal trajectory similarity [9], as trajectories are multivariate time series. SAX divides a time series (a trajectory in our case, same below) into segments and uses a symbol to represent the average of the points within each segment. Two time series are aligned by the time dimension. The differences between each pair of aligned symbol values are summed up as the distance between two time series. SAX essentially discretizes time series and computes their shape similarity.

Dynamic programming-based measures. The linear scan-based measures may compute sub-optimal point matches and hence result in falsely large trajectory similarity values. For example, SPD simply matches the points on two trajectories following the order that the points appear in the respective trajectories. It does not match points based on their spatial distances. To address the issue, dynamic programming (DP)-based measures are proposed to explore a larger point-matching space while confining the computation costs. As Figure 1 shows, such measures examine all point pairs incrementally with DP in $O(n^2)$ time, taking $\Theta(n)$ space to store the intermediate similarity (distance) values.

For example, *Dynamic Time Warping* (DTW) [38], which was designed for time series analysis, e.g., speech recognition, has been extended to trajectories by using the Euclidean distance function, exploiting the property that it allows many-to-one point matching to cope with trajectories with different travel speeds or sampling rates. DTW computes a set of point alignments between two trajectories that achieves a global minimum sum of point-to-point distances. It does not require trajectories of the same length. *Discrete Fréchet* [27] also allows many-to-one point matching. It returns the maximum distance between any matched point pairs.

Longest Common Sub-Sequence (LCSS) [64] adapts a string similarity measure to reveal the longest common sub-sequence of two trajectories. Here, a common sub-sequence refers to consecutive pairs of points that are within a given spatial distance threshold. Another series of studies [18, 19, 54] adapt the edit distance which is also a string similarity measure. They compute the cost to "edit" (insert, delete, or substitute) points on a trajectory to match (i.e., be within a predefined distance threshold) those in the other trajectory. A large distance suggests that more edits are needed to create a match and hence less similar trajectories. For example, *Edit Distance on Real sequence* (EDR) [19] considers a same unit cost for each edit. Next, *Edit Distance with Real Penalty* (ERP) [18] factors the point

distances into the edit costs. Further, *Edit Distance with Projections* (EDwP) [54] uses an interpolation-style insertion operation. It adds points on the line between two adjacent points of a trajectory when point insertions are needed to form matches. A few other measures consider both spatial and temporal distances, such as STEDR and STLCSS [60], which extend EDR and LCSS by adding a temporal distance threshold when matching the points, respectively.

Enumeration-based measures. These measures compute all pairwise point distances directly and aggregate them to form a trajectory similarity (cf. Figure 1). They take $O(n^2)$ time and $O(1)$ space, as they do not need to store intermediate results.

For example, the *One Way Distance* (OWD) [44] uses the average point-to-trajectory distance as the distance between two trajectories. The point-to-trajectory distance here is the minimum distance between a point to any point on a trajectory. *Hausdorff* [7], which was designed for image matching, computes the maximum point-to-trajectory distance. A small Hausdorff distance means each point in a trajectory to have a close match in another trajectory. Thus, the two trajectories form a close match.

Discussion. Some non-learned measures have approximate algorithms (e.g., *Approx-DTW* [81] and *aprxFréchetI* [26]) with lower running times. The comparison between non-learned and learned approximations is not our focus and is left for future work.

3.2 Learned Trajectory Similarity Measures

Studies in the past five years focused on deep learning models to encode trajectories and subsequently learn trajectory similarity [12, 13, 43, 75, 76, 78, 79, 84]. These studies can be categorized into two classes according to their design purposes: (i) to learn and approximate existing non-learned measures [76, 78, 79] with supervised learning, where some non-learned measure is used to provide the supervision signals, and (ii) to learn latent similarity measures independent from any non-learned measures [12, 13, 43]. For the latter class, self-supervised learning is used to learn trajectory embeddings. The similarity between two trajectories are calculated based on their embeddings, e.g., using the L_1 distance.

A core component in these studies is the *backbone trajectory encoder*, which takes a trajectory as input and outputs an embedding. The encoders play a central role in the learned measures. Thus, we review the learned measures based on the encoders used.

Recurrent neural network (RNN)-based measures. Since trajectories are sequences, RNNs form a natural backbone trajectory encoder. RNNs encode each trajectory point recurrently by considering both the historical states (i.e., aggregated information from preceding points) and the current state (i.e., the current point to be encoded). When an RNN terminates, the final output state entails aggregated information from all points on a trajectory, which is used as the trajectory embedding, i.e., $\mathbf{h}$ in Figure 5.

RNN-based measures (using Long Short-Term Memory, LSTM [35] or Gated Recurrent Unit, GRU [22]) take at least $\Omega(nd^2)$ time to encode a trajectory because they need to compute the hidden representation of n states (for the n points on a trajectory), while each hidden representation takes $\Theta(d^2)$ time to compute (i.e., hidden feature mapping). Here, d is the embedding dimensionality. Some approaches may exceed $\Theta(nd^2)$ time, e.g., $\Theta(nd^2 + n^2d)$ time for TMN [75]. RNN-based measures take $\Omega(d^2)$ space for the model

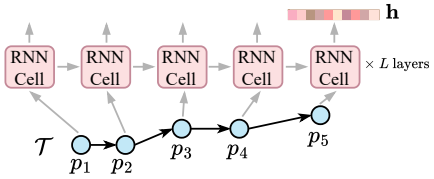


Figure 5: RNN-based learned measures

parameters (i.e., $d \times d$ weight matrices) and another $\Omega(d)$ space to store the intermediate results during the encoding process.

Most RNN-based studies use a grid cell-based input representation. They partition the data space with a grid and transform a trajectory into a sequence of grid cells enclosing the trajectory. This transformation has two benefits: (1) The cell-based representation reduces the input space from points in a continuous space to a small number of discrete cells. This creates an input data distribution that is easier to be learned. (2) As a side effect, the cell-based representation alleviates the impact of GPS errors and varying trajectory sampling rates, leading to more robust embeddings.

The first RNN-based model, *t2vec* [43], adapts GRU by introducing a spatial proximity aware loss function, which penalizes the model when it generates large similarity prediction errors on spatially close trajectories. *NEUTRAJ* [78] uses GRU (according to its released code) and adds a spatial attention memory unit, such that the embedding learning for a trajectory can refer to spatially close trajectories seen by the GRU before. Further, NEUTRAJ has a weighted ranking loss function to encourage model learning from the most similar trajectory pairs. *Traj2SimVec* [84] improves upon NEUTRAJ in two aspects: (1) Traj2SimVec leverages a k-d tree [11] to select a set of most similar trajectories as the positive training samples, rather than randomly sampling as in NEUTRAJ. Hence, it achieves better training efficiency. (2) Traj2SimVec uses a sub-trajectory-based loss to learn the detailed alignment and distances between points, while the loss function of NEUTRAJ is based on the embeddings of full trajectories. Chen et al. [20] also build upon NEUTRAJ. They use an interpolation-based trajectory calibration process to generate smoother trajectories for model training. *CL-Tsim* [24] adopts contrastive learning to help generate more diverse training samples so as to obtain more robust embeddings.

Unlike the models above that learn to encode each trajectory separately, *TMN* [75] introduces a dual-branch model to learn trajectory embeddings and point matches at the same time. Its matching module aims to simulate the computation process of the non-learned measures. TMN requires to input two trajectories together. It cannot be used to encode individual trajectories. Thus, this model cannot be used for the kNN query experiments in Section 4.3.

Besides, Tedjopurnomo et al. [60] and Li et al. (i.e., *RSTS*) [21] adapt t2vec to measure spatio-temporal trajectory similarity. RSTS simply introduces a three-dimensional grid cell where the third dimension models the time. Tedjopurnomo et al. introduce three loss functions to jointly learn trajectory similarity at trajectory, point, and pattern levels. Several studies adopt RNN models to learn similarity for special types of trajectories, e.g., point of interest (POI, *At2vec*) [46] and sports play (*Play2vec*) [69, 70] trajectories.

Self-attention-based measures. *Multi-head Self-attention* ("self-attention" in short) [62] is a more recent sequential model that

addresses the *catastrophic forgetting* issue of the RNNs. It learns the hidden correlation between every two elements in an input sequence (cf. Figure 6). It takes $\Omega(n^2 d)$ time to encode a trajectory. While this seems to be higher than the time taken by RNNs, self-attention models may run much faster than RNNs on GPU. This is because self-attention models run in only one round to compute a sequence embedding, which is highly parallelizable. In contrasts, RNNs need to run n rounds iteratively due to their recurrent structures. Self-attention-based measures take $\Omega(d^2 + nd + n^2)$ space, where the model parameters (i.e., weight matrices) take $\Omega(d^2)$ space, and the intermediate results for computing the attention coefficients (between the n^2 pairs of points) take $\Omega(nd + n^2)$ space.

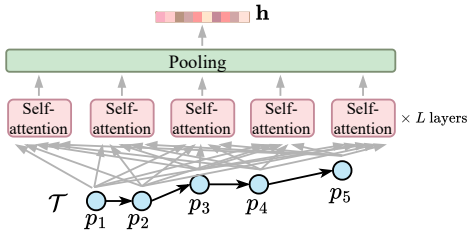


Figure 6: Self-attention-based learned measures

A few studies have used self-attention. *CSTRM* [47] leverages Masked Language Modeling [25] to learn trajectory embeddings in a self-supervised manner. The trained trajectory encoder can be further fine-tuned to approximate non-learned measures. *T3S* [76] combines self-attention and LSTM to capture the topological and spatial features of trajectories. Later, *TrajCL* [13] introduces a fully self-attention-based encoder that adaptively learns the topological and spatial features. It lifts the dependence on RNN models.

Convolutional neural network (CNN)-based measures. CNN models are widely used in image representation learning. They stack convolution kernel and pooling layers to capture image features. To convert a trajectory to a fixed-size image, a blank image corresponding to the data space enclosing the trajectory is created. Then, the points on the trajectory are mapped to pixels of the image, where a pixel value indicates the number of points mapped to the pixel (cf. Figure 7). Such a conversion resembles the grid-cell based representation described earlier and shares similar benefits.

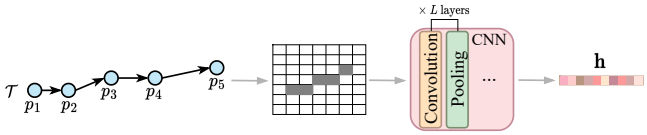


Figure 7: CNN-based learned measures

TrjSR [12] is the only CNN-based model. It is trained by reconstructing a super-resolution trajectory image from a low-resolution one. It encodes a trajectory in $\Omega(mk^2 n_k c)$, where k is the side length of the convolution kernels, n_k is the number of kernels, c is the channel size, and $m \gg d$ is the number of pixels. It takes $\Omega(k^2 n_k c + mc)$ space, where the model parameters take $\Omega(k^2 n_k c)$ space, and the intermediate results take $\Omega(mc)$ space.

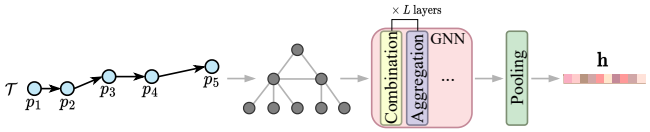


Figure 8: GNN-based learned measures

An issue with the CNN-based measures is that they lose the sequence information of the trajectory points. They cannot distinguish two trajectories traveling towards opposite directions.

Graph neural network (GNN)-based measures. GNNs are designed for graph embedding. In a GNN layer, every node receives and aggregates node embeddings from its neighbors (aggregation). Then, the aggregated information is combined with the embedding of the node to form its updated embedding (combination).

TrajGAT [79] is the only GNN-based measure (cf. Figure 8). It builds a multi-level quadtree [55] that partitions the space into cells with different sizes to help create graphs with multi-granularity views of trajectories. To construct a graph from a trajectory, it queries each trajectory point in the quadtree and adds the tree nodes on the traversed path into the graph as graph nodes. The edges between the added tree nodes are kept as graph edges. To encode a trajectory graph, TrajGAT takes $\Omega(nn_e d)$ time, where n_e is the number of neighbors per node in the graph. TrajGAT takes $\Omega(d^2 + nd + nn_e)$ space, where the model parameters take $\Omega(d^2)$ space, and the intermediate results for computing the node embeddings and the attention coefficients between nodes take $\Omega(nd + nn_e)$ space.

Discussion. We have focused on trajectories in Euclidean space. A closely related topic, trajectory similarity measures for road network space, has also attracted a strong interest [15, 29, 30, 33, 39, 56, 77, 82, 87]. Both non-learned and learned measures have been proposed. The learned measures generally adopt GNNs to exploit the network connectivity patterns and learn embeddings for (intersections or road segments of) the road networks. A trajectory is represented by aggregating the embeddings of the intersections and/or road segments passed by it. Due to the rich works on this topic, we leave an empirical analysis for future work.

3.3 Trajectory Similarity Queries

Trajectory similarity queries typically refer to trajectory k nearest neighbor (kNN) queries (Section 2). All existing trajectory kNN query algorithms are designed for the non-learned measures. Basically, they query trajectories from tree-based indies (e.g., R-trees [32]), with spatial distance-based pruning to speed up the search.

Most of the non-learned measures have their dedicated indices and query algorithms. LCSS comes with a 1-NN query algorithm [64]. It computes a distance lower bound between the query trajectory and the data trajectories in an index node. If this lower bound is greater than the distance between the query trajectory and the currently found nearest trajectory, the node (and all associated trajectories) can be safely pruned. EDR introduces several string encoding-based pruning techniques [19], including Q-gram (i.e., sub-string abstracts), string embedding, and triangle inequality based pruning [36]. ERP computes a distance lower bound based on the summation of trajectory points to prune dissimilar trajectories [18], while EDwP computes the distance between trajectories and bounding boxes of trajectories for pruning [54].

There is also a generic R-tree-based structure for trajectory kNN queries [63] that supports LCSS, DTW, and SPD. Its query algorithm creates a minimum bounding envelope (MBE) for the query trajectory $\mathcal{T}_q$, which bounds the search space by adding spatial and temporal offsets to $\mathcal{T}_q$. Trajectories outside the MBE are pruned. *DFT* [73] is a generic, distributed segment-based R-tree-like index that supports Hausdorff and Fréchet. The kNN algorithm of DFT leverages a small set of sampled data trajectories (more than k) to obtain a distance threshold ϵ that bounds the largest distance between $\mathcal{T}_q$ and its k-th NN. Then, DFT leverages ϵ to filter out dissimilar data segments of $\mathcal{T}_q$. *DITA* [57] is a third R-tree-based index, and it supports DTW and ERP. It indexes points sampled at equal intervals on each data trajectory, for space efficiency. Its kNN algorithm also starts by computing a k-th NN distance threshold ϵ. Unlike DFT, DITA can shrink ϵ during its search process, by subtracting the distance between the currently matched points of a data trajectory and the query trajectory. This is because DTW and ERP compute trajectory distance by summing up the distances between the matched points. In contrast, Fréchet and Hausdorff compute the global maximum distance of the matched points, such that ϵ cannot be updated progressively. We use DFT and DITA for indexing in the experiments due to their generalizability.

3.4 Trajectory Clustering

Earlier studies on trajectory clustering [17, 23, 40, 42] apply classic clustering algorithms, e.g., k-medoids, with non-learned measures (e.g., LCSS), For example, Lee et al. [40] presents a partition-and-group framework for trajectory clustering. Li et al. [42] focus on finding top-k clusters considering the cluster cardinality. Besides, a series of studies [17, 23, 42] consider online trajectory clustering.

Recent studies [16, 28, 31, 51, 80, 83] exploit deep learning to improve clustering efficiency and effectiveness. Yao et al. [80] leverage an RNN auto-encoder. They first learn trajectory embeddings and then cluster the embeddings by the k-medoids algorithm. This clustering paradigm is followed by the later studies. *DETECT* [83] and *Trip2Vec* [16] further consider POIs on trajectories to study the mobility pattern of trajectories. *E2DTC* [28] adopts t2vec [43] as the encoder and fine-tunes it with a multi-task loss function that considers both trajectory similarity and cluster distribution. Besides, several studies [51, 68] extend deep trajectory clustering to different data domains, e.g., aircraft or vessel trajectories.

4 EXPERIMENTS

We study the performance of both the non-learned and the learned measures empirically, for three representative tasks: trajectory similarity computation, clustering, and similarity querying.

4.1 Experimental Setup

Datasets. We use five real-world trajectory datasets, which are used in recent studies [12, 13, 24, 75, 76, 78, 79]. We pre-process the datasets by discarding short trajectories with less than 20 points, as done in previous studies [12, 13, 21, 43, 75, 78]. Table 2 summarizes dataset statistics after pre-processing.

Porto [2] contains taxi trajectories collected from Porto, Portugal, between July 2013 and June 2014. The trajectory points do not come with timestamps. The average number of points per trajectory is 50, which is the smallest among the five datasets.

Table 2: Dataset statistics

	Porto	Germany	Geolife	Chengdu	Xi'an
#traj.	1,380,777	243,417	15,972	1,259,639	1,009,693
#points per traj.	20 ~ 3,836	20 ~ 200	20 ~ 56,780	20 ~ 1,891	20 ~ 8,730
Avg. #points per traj.	50	84	1,201	142	262
Traj. length (m)	8 ~ 110,227	10,805 ~ 1.4e8	1 ~ 556,814	36 ~ 51,789	23 ~ 219,824
Avg. traj. length (m)	6,445	338,622	25,175	4,586	6,204
Spatial area (km^2)	16.0 × 20.1	1,023 × 1,187	235.9 × 232.5	9.7 × 9.7	9.9 × 9.6
Time span	-	-	five years	a week	a week

Germany [1] contains user trajectories collected mainly within Germany from OpenStreetMap without timestamps. This dataset has the longest average trajectory length, i.e., 338 km, and covers the largest spatial area, i.e., over 10^6 km^2.

Geolife [86] contains user trajectories collected in Beijing, China, from April 2007 to August 2012. We have further discarded the trajectories recorded outside Beijing, which take up a very small portion (6%) of the data. Geolife records trajectories of different types of user movements, e.g., walking, cycling, and driving. We use all trajectories together due to the limited dataset size.

Chengdu [3] contains ride-hailing trajectories from the Second Ring road of Chengdu, China, which is a densely populated area, in the first week of November 2016. It covers the smallest area among the five datasets and has the shortest average trajectory length.

Xi'an [3] contains ride-hailing trajectories from the Second Ring Road of Xi'an, China, in the first week of October 2018. This is the most current dataset and we use it by default. Its sampling rate is the lowest, i.e., roughly 30 meters per sampled point.

Similarity measures. For each category of measures in Section 3, we study the state-of-the-art as well as measures that have unique computation strategies, such as NEUTRAJ and TMN, which are both RNN-based but use single- and dual-branch models, respectively. As there is no released code for SAX on trajectories, we follow the proposal [9] and compute the duration for which two trajectories stay less than a given distance threshold based on their symbolic representations. We denote this measure as *SAR*. The similarity measures tested are listed below, where "ST" refers to measures that consider both spatial distances and time differences.

(1) Non-learned: ① Linear scan: **CDDS** (ST) and **SAR** (ST) ② DP: **DTW**, **ERP**, **Fréchet**, and **STEDR** (ST) ③ Enumeration: **Hausdorff**.

(2) Learned: ① RNN: **NEUTRAJ** (single-branch), **TMN** (dual-branch), **RSTS** (ST) ② Self-attention: **T3S** (RNN + self-attention), **TrajCL** (self-attention) ③ CNN: **TrjSR** ④ GNN: **TrajGAT**.

Implementation details. All experiments are run on a virtual machine with an 8-core Intel Xeon 4214 CPU (2.2 GHz), 128GB RAM and an NVIDIA V100 GPU (5,120 FP32 cores and 16GB VRAM). We use these due to their ease of access for reproducibility reasons. We repeat each experiment five times and report the average results.

For the non-learned measures, we use *traj-dist* [4], a commonly used CPU-based sequential implementation of trajectory similarity computation in Python. It supports DTW, ERP, Fréchet, and Hausdorff, while we add support for STEDR, CDDS, and SAR.

We generalize the non-learned measures into three meta-GPU-based algorithms, i.e., linear scan-based, DP-based, and enumeration-based algorithms, to enable GPU-based implementation. We implement these algorithms on CUDA cores of GPUs in Python with the

Numba 0.53.1 library. We leave the details of the meta-GPU-based algorithms in the technical report [5]. We note that parallel implementations of some of the non-learned measures exist (e.g., [72]). We use our meta-GPU-based algorithms instead because these do not contain special optimizations, enabling us to capture better the speedups achievable by a simple GPU-based adaptation.

For the learned measures, we use their released source code that are written in PyTorch, which can be run on either CPU or GPU, except for T3S and RSTS for which no code is available. We implement T3S and RSTS with PyTorch 1.8.1 following their papers.

Following previous studies [75, 76, 78], we set the embedding dimensionality d to 128. The side length of the grid cells is 100 meters for the grid-based learned measures. The image size m of TrjSR is 162×128 following its paper. The number of encoder layers stacked in each learned measure is 2, except for TrjSR, which has 10 CNN layers following its paper. We set the number of GPU cores, n_c, for computing the similarity between a pair of trajectories to 64. The batch size is 512 by default, to fit every measure in memory.

4.2 Trajectory Similarity Computation

4.2.1 Setup. For each dataset, we randomly sample 100,000 pairs of trajectories and compute their similarity using each measure. Each trajectory is limited to at most 200 points, following previous studies [13, 21, 75, 78, 79], as the learned measures may fail on longer trajectories due to out-of-memory errors (cf. Section 4.2.3). *Each trajectory is used only once, and no embeddings can be reused in Section 4.2 by default.* This experiment setting aims to evaluate the efficiency of computing the similarity of a pair of trajectories arriving online. We vary the data size of each run (i.e., **single** or **batched**) and the computation unit (i.e., **CPU** or **GPU**). Here, "single" refers to computing for a single pair of trajectories (with a single core), while "batched" refers computing for multiple pairs (i.e., 512) of trajectories using multiple cores of a computation unit.

Note that, when we compute trajectory similarity in batches, different "batched" computation paradigms are applied on GPU and CPU, respectively. On GPU, we use n_c cores for parallel processing of each trajectory pair following our meta-GPU-based algorithms. On CPU, we simply use a computation core for each trajectory pair, i.e., no parallel processing is done on individual trajectory level. This is because there are much more cores on GPU than on CPU [71]. In batch processing mode, all CPU cores can be fully utilized by the trajectory pairs in a batch already, while the cores on GPU can be shared at the individual trajectory level.

We run on both CPU and GPU for two reasons: (1) to show the speedups achievable using GPUs for the *same* trajectory similarity measure and guide the choice between CPU (less expensive but slower) and GPU (faster but more expensive) for different applications; and (2) to compare the speedups achievable using GPUs for *different* trajectory similarity measures, to reveal the measures that can better exploit the power of GPU parallelization.

We report the **elapsed time** (in seconds) and the **space cost** (in GB). All input data is aligned to the same form, i.e., raw 2-dimensional trajectory points, and the outputs are similarity scores. The cutoff running time is 7,200 seconds. We use "**OT**" to denote overtime errors and "**OOM**" to denote out-of-memory errors.

Table 3: Elapsed time of trajectory similarity computation (best results are in bold)

Dataset	Measure		Single		Batched	
			GPU	CPU	GPU	CPU
Porto	Non-learned	DTW	146.23	36.04	**3.96**	10.07
		ERP	190.76	78.46	4.09	12.88
		Fréchet	146.90	32.59	4.00	7.44
		Hausdorff	**135.71**	**26.64**	4.07	**6.57**
	Learned	NEUTRAJ	4907.14	3088.88	94.69	229.21
		TMN	389.63	535.48	54.67	254.23
		T3S	465.75	861.54	49.49	649.17
		TrajCL	561.94	666.82	59.49	276.30
		TrjSR	584.21	OT	301.74	4409.75
		TrajGAT	3319.11	2580.19	494.34	603.32
Germany	Non-learned	DTW	132.21	75.12	8.75	21.38
		ERP	163.34	191.38	**8.33**	35.03
		Fréchet	130.49	67.14	8.37	18.32
		Hausdorff	**113.22**	**39.21**	8.36	**14.42**
	Learned	NEUTRAJ	OT	5893.80	164.62	367.51
		TMN	429.18	776.21	113.06	432.96
		T3S	428.82	979.93	74.93	392.11
		TrajCL	636.30	945.13	93.91	436.90
		TrjSR	914.14	OT	165.69	1909.11
		TrajGAT	2440.17	2194.09	588.12	1775.44
Geolife	Non-learned	DTW	165.68	130.19	**5.68**	24.70
		ERP	212.14	283.55	5.94	48.91
		Fréchet	165.91	91.40	5.82	23.02
		Hausdorff	**140.97**	**47.29**	5.76	**16.95**
	Learned	NEUTRAJ	4182.19	2682.02	118.14	306.60
		TMN	524.18	608.16	68.64	414.70
		T3S	618.42	1242.94	93.35	833.46
		TrajCL	613.15	711.60	88.28	425.05
		TrjSR	588.39	OT	299.55	4417.78
		TrajGAT	4375.38	3799.01	1335.27	1815.51
	Non-learned (ST)	STEDR	165.46	221.35	**6.50**	43.94
		CDDS	**138.17**	**24.74**	6.65	**17.35**
		SAR	158.97	30.45	6.85	17.53
	Learned (ST)	RSTS	2970.58	3745.92	805.44	939.33
Chengdu	Non-learned	DTW	139.91	112.59	9.87	29.34
		ERP	168.70	298.32	9.55	53.43
		Fréchet	138.66	102.87	**9.50**	25.56
		Hausdorff	**116.74**	**50.56**	9.55	**18.78**
	Learned	NEUTRAJ	5235.88	3306.81	108.83	222.66
		TMN	251.70	409.24	78.08	220.05
		T3S	477.05	1188.18	89.10	428.20
		TrajCL	617.49	749.70	81.66	229.45
		TrjSR	947.35	OT	197.99	1950.42
		TrajGAT	5713.50	5387.81	2700.47	5682.02
	Non-learned (ST)	STEDR	137.57	233.34	12.11	47.82
		CDDS	**114.57**	30.18	**11.85**	20.20
		SAR	160.68	**29.79**	12.69	**17.53**
	Learned (ST)	RSTS	1964.53	4409.20	1087.39	1212.71
Xi'an	Non-learned	DTW	168.24	133.42	6.21	28.11
		ERP	215.28	362.92	6.34	54.04
		Fréchet	169.60	117.20	6.49	25.66
		Hausdorff	**145.41**	**70.63**	**5.83**	**17.75**
	Learned	NEUTRAJ	4226.71	2609.39	102.85	215.03
		TMN	394.89	463.79	62.08	230.18
		T3S	639.53	1958.81	83.69	778.61
		TrajCL	622.56	733.51	71.37	298.11
		TrjSR	625.54	OT	318.97	4629.79
		TrajGAT	4439.46	4114.52	1112.02	1560.24
	Non-learned (ST)	STEDR	171.03	280.66	7.25	48.09
		CDDS	**142.51**	**33.77**	**7.06**	22.66
		SAR	162.54	34.27	7.22	**19.40**
	Learned (ST)	RSTS	3736.67	6030.93	1257.75	1473.71

4.2.2 Results under Online Computation Settings. Table 3 shows the elapsed times when all computation is done online, including the trajectory embeddings. *Overall, the non-learned measures take less time to compute than the learned ones when they are run online on the same computation units.* Even for batched computation on GPU, where the learned ones are supposed to be at their best, the non-learned measures are at least an order of magnitude faster.

A few detailed observations can be made from the table:

(1) **Both non-learned and learned measures take the least time for batched computation on GPU.** This setting best exploits the parallelization power of GPU. Such a setting thus should be used for one-off similarity computation of a large number of trajectories (e.g., for offline trajectory mining tasks such as contact tracing).

(2) **When computing the similarity between trajectories on per pair basis ("single"), e.g., for an ad hoc similarity computation, the non-learned measures prefer CPU while the learned ones prefer GPU.** The non-learned measures are lightweight, such that the savings achieved by GPU for a single trajectory pair is not worth the data transfer costs between CPU and GPU. The learned measures, on the other hand, take less time on GPU, where matrix multiplications in the trajectory encoders are better suited.

(3) **Among the non-learned measures, the enumeration-based one, Hausdorff, is the fastest in general** (excluding the spatio-temporal measures), due to its simple computation rules. When computed on GPU, Hausdorff further benefits from its independent and fully parallelized calculation of the pairwise point similarity scores. Among the DP-based measures, ERP generally takes the most time, as it has the most complex computation rules. We note that all non-learned measures are very fast and have similar elapsed times under the GPU-batched mode, while DTW reported marginally faster elapsed time on Porto and Geolife.

Among the spatio-temporal measures, the linear scan-based CDDS is generally the fastest on GPU for its simple calculation. SAR performs similar to CDDS which is also a linear-time measure. On CPU, SAR can run faster than CDDS, because it can early terminate the computation when two trajectories do not align in time. This early termination does not help on GPU as the trajectories are distributed to each streaming core for processing anyway.

(4) **Among the learned measures, the attention-based ones T3S and TrajCL are more efficient in time** – T3S is the slower among the two as it also uses an RNN. This is because the attention computation can be fully paralellized (cf. Section 3). An RNN-based measure, TMN, is the fastest among the learned measures, due to its simple model (a vanilla LSTM). NEUTRAJ and RSTS are also RNN-based, while they suffer in efficiency especially on the "single" mode. NEUTRAJ has an expensive spatial module to compute the attention coefficients between the current and the seen training trajectories (cf. Section 3), while RSTS has an expensive input pre-processing step (cf. Table 4). The CNN-based measure TrjSR and the GNN-based measure TrajGAT are also slow. TrjSR suffers in the large amount of computation for its CNN, while TrajGAT spends much time on converting a trajectory into a graph.

Computation time decomposition. We further "zoom in" on the time costs. The elapsed time of a similarity measure reported in Table 3 (**Total** in Table 4) mainly consists of three parts: (1) input pre-processing time (**Pre.**) to convert a raw trajectory into the format required by a measure (e.g., a graph for TrajGAT), (2) embedding time (**Emb.**) to compute trajectory embeddings (inapplicable to the non-learned measures), and (3) similarity computation time (**Cmp.**), i.e., the time to compute point matches and distances for the non-learned measures, or to compute the embedding distances for the learned measures. There is also time for transferring the results and other minor inter-step processing, which is very small and hence omitted from the table. We only show the results on Xi'an in Table 4, as similar patterns are observed on the other datasets.

Table 4: Detailed time and space costs of batched trajectory similarity computation on Xi'an

Measure	Time on GPU				Time on CPU				Space
	Pre.	Emb.	Cmp.	Total	Pre.	Emb.	Cmp.	Total	(GB)
DTW	5.64	-	0.56	6.21	0.65	-	27.45	28.11	**0.02**
ERP	5.72	-	0.62	6.34	0.52	-	53.51	54.04	**0.02**
Fréchet	5.91	-	0.58	6.49	0.50	-	25.16	25.66	**0.02**
Hausdorff	5.28	-	0.55	**5.83**	0.44	-	17.30	**17.75**	**0.02**
NEUTRAJ	68.15	34.69	1e-4	102.85	67.83	147.20	2e-3	215.03	0.13
TMN	56.10	5.97	1e-4	62.08	56.18	174.00	2e-3	230.18	6.37
T3S	59.73	23.95	1e-4	83.69	58.27	720.34	2e-3	778.61	3.02
TrajCL	60.38	10.99	1e-4	71.37	57.79	240.30	2e-3	298.11	5.81
TrjSR	86.56	232.40	1e-4	318.97	90.08	4539.71	2e-3	4629.79	9.12
TrajGAT	1035.67	76.35	1e-4	1112.02	1028.66	531.58	2e-3	1560.24	13.89
STEDR (ST)	6.72	-	0.53	7.25	0.83	-	47.26	48.09	**0.02**
CDDS (ST)	6.49	-	0.57	**7.06**	0.60	-	22.06	22.66	**0.02**
SAR (ST)	6.60	-	0.62	7.22	0.68	-	18.72	**19.40**	0.03
RSTS (ST)	1239.22	18.53	1e-4	1257.75	1245.93	227.78	2e-3	1473.71	0.76

Overall, although the non-learned measures take more time than the learned measures on similarity computation, they have better efficiency on input pre-processing and do not require trajectory embeddings, which explains for their smaller total elapsed times. The non-learned measures take more time for input pre-processing on GPU than on CPU. This is because when preparing data on GPU, we need to pad the trajectories in a batch to the same length and group them into a matrix, which is required by CUDA.

Table 4 also suggests that the learned measures can be faster (in Cmp.), when the embeddings can be pre-computed and reused.

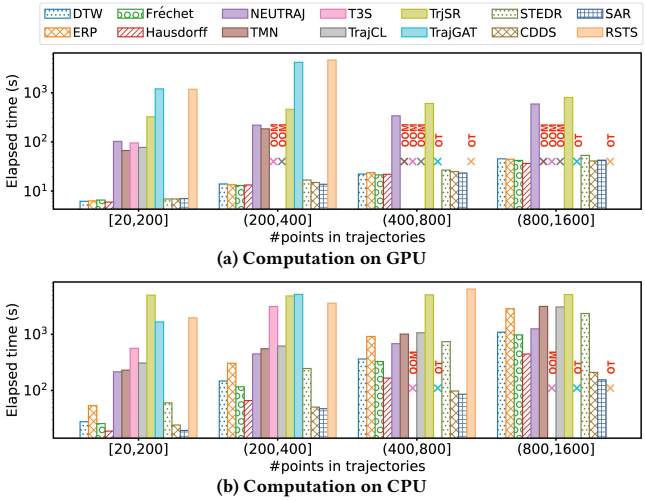
Memory costs. Table 4 further shows that the non-learned measures take less memory, as they do not need large weight matrices.

We also study the maximum batch size for each measure. Overall, the non-learned measures allow a larger batch size than the learned ones except NEUTRAJ which also has a good space efficiency for its simple RNN. Detailed results are included in [5].

4.2.3 Impact of the Number of Points in Trajectories n. We vary the number of points on trajectories, n, from 20 to 1,600. Most existing studies on learned measures only used trajectories with up to 200 points. Ours is the first set of results with over 1,000 points.

We focus on the batched setting on GPU and CPU as this is a more realistic setting in practice, and we omit the "single"-mode results as similar result patterns are observed as before. We again present the results on the largest dataset, Xi'an, for brevity.

Figure 9 shows the results, where each configuration, e.g., "[20, 200]" means to compute trajectory similarity for 100,000 pairs of randomly sampled trajectories each with $n = 20$ to 200. All measures except TrjSR have increasing elapsed times as n increases. The non-learned measures (denoted by empty bars with hatches) show clear advantage on GPU, while both types of measures perform



(a) Computation on GPU

(b) Computation on CPU

Figure 9: Elapsed time vs. #points in trajectories

closer on CPU when n becomes larger. TrjSR uses fixed-size images to represent trajectories, which is independent from n. TMN, T3S, and TrajCL trigger out of memory errors when n becomes large, especially on GPU. This is because TMN computes the attention coefficients between all pairs of points on two trajectories, while T3S and TrajCL compute the coefficients between every two points on each trajectory. These coefficient computations lead to a quadratic memory space overhead with respect to n.

4.2.4 Impact of the Trajectory Embedding Dimensionality d. We vary d from 32 to 256 following the literature and report the results in Figure 10. The non-learned measures are not impacted by d. The elapsed times of the learned measures present only a slightly increasing trend, as their matrix operations have been well parallelized by the PyTorch package. The learned measures are still slower even when $d = 32$, which is the smallest d value used in the literature [79]. When d increases to 256, TrajGAT has an out of memory error, because it takes more space to compute the graph-based trajectory embeddings. We show only the results on GPU for conciseness as similar patterns are observed on CPU.

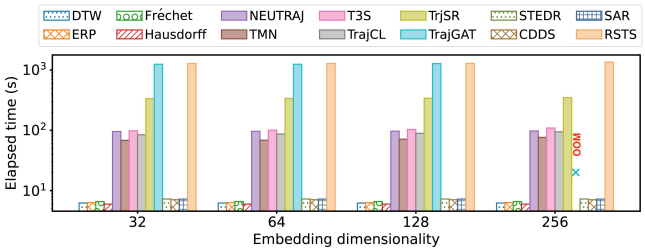
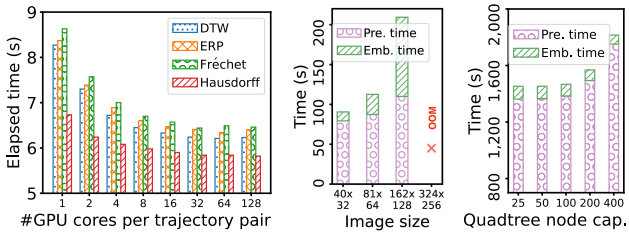


Figure 10: Elapsed time vs. embedding dimensionality (GPU)

4.2.5 Impact of the Number of Trajectory Pairs. We further vary the number of trajectory pairs from 1,000 to 1,000,000. Again, the computation time grows with the number of trajectory pairs, and the non-learned measures outperform the learned ones. These results reinforce the advantage of the non-learned measures in time efficiency. We omit the result figures due to space limit.

4.2.6 Impact of Computation Hardware. We also study the impact of different computation hardware, e.g., NVIDIA A100 which is a more advanced GPU. The results confirm that the similarity measures benefit from such more advanced hardware, which are included only in the technical report [5] due to the space limit.

4.2.7 Impact of the Number of GPU Cores n_c. We vary the number of GPU cores n_c for computing the similarity between a pair of trajectories from 1 to 128. Figure 11a reports the results on Xi'an for the non-learned measures (the learned measures are implemented based on PyTorch which does not offer control on the number of GPU cores for such computation). Overall, when n_c increases, the elapsed times first decrease before stabilizing at around $n_c = 32$. This is because when n_c grows, the number of trajectory points assigned to each GPU core decreases, which helps reduce the elapsed time. When n_c grows further, the benefit of reducing the number of points per core shrinks, while the overhead on scheduling more cores increases, such that the elapsed time does not decrease further.



(a) #GPU cores per trajectory pair

(b) TrjSR image size

(c) Quadtree node capacity of TrajGAT

Figure 11: Elapsed time vs. hardware and hyper-parameters

4.2.8 Impact of the Hyper-parameters of Learned Measures. Hyper-parameters of the learned measures, such as the image size m in TrjSR and the quadtree node capacity q_{nc} in TrajGAT, also impact the time efficiency. Particularly, the quadtree node capacity determines the quadtree structure in TrajGAT, which is inversely proportional to the number of neighbors of a graph node in TrajGAT, n_e – the time complexity of TrajGAT is linear in n_e (cf. Section 3.2).

We vary m and q_{nc}, and we report the detailed time costs for batched similarity computation on GPU in Figures 11b and 11c, respectively. When m increases, TrjSR takes more time to run, since the input size grows. For TrajGAT, when q_{nc} increases, the pre-processing time increases, because its hash computation to identify quadtree node candidates becomes more complex during graph construction. The embedding time decreases, since each node corresponds to a larger area, and each trajectory can be represented with fewer nodes, and n_e also decreases. Overall, the total time increases with q_{nc}, which is dominated by the pre-processing time.

4.2.9 Results with Embedding Reuse. Existing studies [13, 76, 78] assume two sets of trajectories and report the time to compute the similarity between each pair of trajectories, one from each dataset. *They compute the embedding of each trajectory once and reuse it for all similarity computations*, which saves the overall elapsed time

substantially. We repeat such a set of experiments to show that, when the embeddings can be pre-computed, the learned measures indeed can delivery their promised higher computation efficiency.

On each dataset, we randomly sample and form two sets of 1,000 and 100 trajectories, respectively, such that we have 100,000 trajectory pairs to compute as before. We report the results of batched computation on Xi'an in Table 5. Now the learned measures TrajCL, T3S, and NEUTRAJ outperform the non-learned ones, benefiting from the one-off input pre-processing and embedding times. TrjSR and TrajGAT also benefit substantially. However, they are still slower than the non-learned ones because even the one-off computations are too expensive. These results are consistent with the literature [13]. TMN does not apply in this experiment, as it is a pairwise model and cannot pre-compute for individual trajectories.

Table 5: Detailed time and space costs of batched trajectory similarity computation between two trajectory sets on Xi'an

Measure	Time on GPU				Time on CPU				Space
	Pre.	Emb.	Cmp.	Total	Pre.	Emb.	Cmp.	Total	(GB)
DTW	0.10	-	0.58	0.68	0.01	-	28.32	28.33	**0.02**
ERP	0.10	-	0.70	0.80	0.01	-	55.10	55.11	**0.02**
Fréchet	0.10	-	0.57	0.67	0.01	-	26.03	26.04	**0.02**
Hausdorff	0.10	-	0.63	0.73	0.01	-	18.71	18.72	**0.02**
NEUTRAJ	0.49	0.46	1e-4	0.96	0.51	0.78	2e-3	**1.29**	0.13
TMN	-	-	-	-	-	-	-	-	-
T3S	0.48	0.03	1e-4	0.51	0.49	5.09	2e-3	5.63	3.02
TrajCL	0.47	0.02	1e-4	**0.49**	0.46	1.75	2e-3	2.21	5.81
TrjSR	0.62	0.90	1e-4	1.52	0.63	36.89	2e-3	37.52	9.12
TrajGAT	17.96	1.81	1e-4	19.77	18.23	17.30	2e-3	35.53	13.89
STEDR (ST)	0.14	-	0.59	0.73	0.02	-	41.85	41.87	**0.02**
CDDS (ST)	0.14	-	0.50	**0.64**	0.02	-	20.97	20.99	**0.02**
SAR (ST)	0.21	-	0.79	1.00	0.02	-	24.54	24.57	0.03
RSTS (ST)	4.98	0.08	1e-4	5.06	5.03	1.18	2e-3	**6.30**	0.76

4.3 Trajectory Similarity kNN Queries

4.3.1 Setup. The default trajectory dataset D contains 100,000 randomly sampled trajectories each with 20 to 200 points. The query set Q contains 1,000 trajectories in the same length range randomly sampled outside D. The query parameter k is 50 by default.

We deploy *dedicated indices* for the non-learned and the learned measures, respectively, to provide a fairer comparison. This differs from the existing studies [43, 78] that use R-trees [10] to index the raw trajectories for query pruning and compute the similarity by the embeddings. For the non-learned measures, we use two recent indices, i.e., DITA [57] for DTW and ERP, and DFT [73] for Hausdorff and Fréchet (cf. Section 3.3). For the learned measures, we use generic vector similarity search indices for trajectory embedding-based kNN queries, as there are *no existing indices designed specifically* for trajectory embeddings. We use Faiss [37] which is a generic, widely used similarity search library for vectorized data. The *Inverted file index* **(IVF)** from this library is used by default for its good balance between query efficiency and accuracy.

We report results on both query efficiency and effectiveness (i.e., accuracy). The learned measures are inaccurate. Results on both aspects together guide the choice of learned measures for different application scenarios. Following the literature [76, 78, 79], we use hit ratios (**HR@50**) to measure the accuracy of the learned measures. We also report the cost of index construction. We use all learned measures expect TMN which cannot be applied in index-based kNN queries because it does not produce embeddings in advance. Like before, we focus on batched processing on Xi'an.

4.3.2 Overall Results. We report the elapsed times of index building and query processing in Table 6. The *index building time* in the table includes the time to build the indices and – for the learned measures – the time to compute embeddings for the data trajectories. Likewise, the *query time* for the learned measures includes the time to compute an embedding for the query trajectory (as the query trajectories may arrive online). Overall, the learned measures take more time on index building, for faster query processing.

Table 6: kNN index building and query performance results

Measure	Index building		Query time (GPU)	Query time (CPU)
	Time	Space (GB)		
DTW	0.71	**1.60**	1193.12	5441.06
ERP	**0.69**	**1.60**	1304.40	6792.25
Fréchet	28.46	2.63	878.89	1783.95
Hausdorff	28.46	2.63	903.41	3410.28
NEUTRAJ	49.15	2.55	0.98	14.30
T3S	58.39	2.55	0.96	8.58
TrajCL	47.88	2.55	**0.89**	**4.33**
TrjSR	171.15	2.55	2.42	23.43
TrajGAT	661.69	2.55	6.62	60.45

Index building costs. The index build times of the learned measures are higher because they need to first encode raw trajectories into embeddings. In comparison, the non-learned measures index raw trajectory points or segments, and their indices are faster to build. The DITA indices are the fastest to build, because they only index a few pivot points of each trajectory. Their space costs (i.e., the index size) are hence also the smallest. DFT indexes all trajectory segments which has the highest space costs.

kNN query costs. As for kNN querying, the learned measures outperform the non-learned ones significantly, by some two orders of magnitude. The fastest learned measure TrajCL is 984 times and 410 times faster than the fastest non-learned measure Fréchet when querying on GPU and CPU, respectively. Both DFT and DITA are spatial indices, which suffer in their pruning capability when indexing objects with highly skewed aspect ratios, such as trajectories which are long and thin. The trajectories that cannot be pruned require expensive similarity computations with the non-learned measures. In contrast, the learned measures use the vector index IVF. The embeddings are pre-computed, and the similarity scores are now computed by simple embedding scans, which is highly efficient. This set of results confirms an important advantage of the learned measures, i.e., their embeddings can be indexed that enable fast kNN queries, even using just generic vector indices. Such results motivate further development of dedicated indices for even faster embedding-based kNN query processing.

Overall kNN query accuracy. In Table 7, the HR@50 values indicate how accurately the top 50 trajectories returned by the learned measures approximate those returned by the non-learned measures

Table 7: kNN query accuracy (HR@50)

	DTW	ERP	Fréchet	Hausdorff
NEUTRAJ	**0.629**	0.417	0.671	0.678
T3S	0.514	**0.756**	0.756	0.657
TrajCL	0.528	0.421	**0.814**	**0.831**
TrjSR	0.521	0.377	0.630	0.757
TrajGAT	0.561	0.287	0.320	0.286

(recall that the learned measures are trained to approximate the non-learned ones). Overall, none of the learned measures return fully accurate results, and their HR@50 are lower than 0.7 in most cases, suggesting opportunities to explore models that can better approximate the non-learned measures. While there are exceptions, the models appear to achieve higher HR@50 scores approximating Fréchet and Hausdorff than approximating DTW and ERP. This suggests that DTW and ERP are more difficult to be approximated (by the learned measures tested) than Fréchet and Hausdorff – similar observations have been reported for DTW [76, 79]. TrajCL, in particular, achieves HR@50 > 0.8 for these two measures.

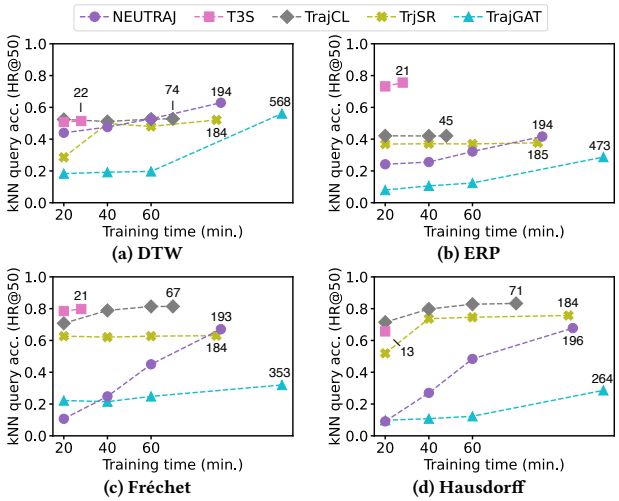


Figure 12: Training time of the learned measures vs. kNN query accuracy (the numbers are model convergence times)

***k*NN query accuracy vs. trajectory similarity model training time.** Figure 12 reports HR@50 of the learned measures, where the model training time is constrained from 20 to 60 minutes. Note that the trajectory similarity model training time is excluded from the index build times reported above (and is excluded from all the other sets of results), i.e., the last set of experiments assumes trained similarity measures. This set of results further provides guidance on model selection for applications with time constraints on system preparation from scratch or model retraining.

The figure shows that when the model training time is very limited, e.g., 20 minutes, T3S is generally the best option, except that TrajCL is 6% more accurate than T3S when approximating Hausdorff. When the training time increases to 40 minutes, TrajCL performs the best to approximate the non-learned measures except for the ERP (for which T3S remains the best). This is because T3S uses a vanilla attention model that is faster to train, while TrajCL uses a multi-view attention model that takes more time to train but may produce higher accuracy given enough time. *These results also confirm the superiority of using attention models to learn trajectory similarity, especially when the training time is limited.*

4.3.3 Impact of Indices for Learned Measures. We compare the default IVF index with the flat index (**FLAT**) and the hierarchical navigable small world index (**HNSW**) [48] for trajectory kNN queries. FLAT stores the full trajectory embeddings of the input dataset D and performs full scans for similarity searches. HNSW creates a navigable small world graph based on the embeddings and uses hierarchical skip lists to link graph nodes to speed up the search. Both HNSW and IVF return approximate query results. We repeat the experiments of batched kNN queries on Xi'an. We report the index build time, the query time on GPU, and the query accuracy to approximate the Hausdorff measure.

Table 8 presents the results. Overall, the three indices have similar build times, which include and are dominated by the embedding computation times. HNSW takes the longest to build because of its complex graph structure, which in turn helps it to achieve the fastest query times while sacrificing the accuracy. On the opposite, FLAT is the fastest to build, while it is the slowest for query processing because of its brute-force query strategy. FLAT is still inaccurate, because the embeddings are approximations of the raw trajectories. IVF has the best balanced performance. It builds as fast as FLAT, is fast at query processing, and has a good accuracy.

Table 8: Index comparison for the learned measures

Measure	Building time			Query time			Query accuracy		
	FLAT	HNSW	IVF	FLAT	HNSW	IVF	FLAT	HNSW	IVF
NEUTRAJ	48.87	53.22	49.15	4.09	0.61	0.98	0.717	0.661	0.678
T3S	57.69	69.99	58.39	3.91	0.51	0.96	0.694	0.628	0.657
TrajCL	46.92	58.31	47.88	3.90	0.43	0.89	0.879	0.819	0.831
TrjSR	162.63	183.42	171.15	6.32	1.33	2.42	0.796	0.731	0.757
TrajGAT	660.74	667.25	661.69	10.13	6.19	6.62	0.327	0.259	0.286

4.3.4 Impact of the Size of the Trajectory Dataset $|D|$. We vary the size of the trajectory dataset $|D|$ from 10^4 to 10^6. The indices of the non-learned measures are consistently faster to build, while the learned measures are faster at query processing. We report detailed results in the technical report [5] due to space limit here.

4.4 Trajectory Clustering

4.4.1 Setup. We follow the commonly used trajectory clustering paradigm [80, 83]. For the non-learned measures, we apply the k-medoids algorithm [52] to cluster the raw trajectories. For the learned measures, we apply k-medoids on the embeddings.

The default dataset D consists of 1,000 randomly sampled trajectories from a dataset, where the number of points on trajectories n is between 20 and 200. The number of clusters k is 10 by default.

Trajectory clustering based on learned measures is also an approximation of that based on non-learned measures. We follow the literature and use the *rand index* (RI) to evaluate clustering accuracy. RI computes the percentage of the ground-truth similar trajectory pairs (derived based on non-learned measures) that are assigned to the same cluster. We again report results on Xi'an.

Table 9: Detailed time and space costs of batched trajectory clustering on Xi'an

Measure	Time on GPU				Time on CPU				Space
	Pre.	Emb.	Clst.	Total	Pre.	Emb.	Clst.	Total	(GB)
DTW	0.60	-	3.22	3.82	0.24	-	226.09	226.33	**0.02**
ERP	0.60	-	3.70	4.29	0.25	-	436.48	436.73	**0.02**
Fréchet	0.60	-	3.26	3.86	0.24	-	223.68	223.92	**0.02**
Hausdorff	0.61	-	3.37	3.98	0.25	-	167.60	167.85	**0.02**
NEUTRAJ	0.26	0.17	0.27	**0.70**	0.35	0.87	0.21	**1.43**	0.88
T3S	0.40	0.02	0.31	0.74	0.47	2.26	0.34	3.06	2.74
TrajCL	0.36	0.01	0.33	**0.70**	0.40	1.40	0.31	2.11	2.35
TrjSR	0.52	0.05	0.41	0.97	0.58	17.12	0.31	18.01	9.12
TrajGAT	6.82	0.35	0.22	7.39	7.56	6.87	0.32	14.75	2.08
STEDR	0.79	-	3.64	4.42	1.09	-	386.26	387.34	**0.02**
CDDS	0.82	-	3.35	**4.17**	1.08	-	171.42	172.50	**0.02**
SAR	0.81	-	3.92	4.73	1.09	-	167.04	168.13	0.03
RSTS	6.22	0.05	0.31	6.58	6.37	0.97	0.31	**7.65**	0.38

4.4.2 Overall Results. Tables 9 and 10 report the results. *Overall, the learned measures are faster than the non-learned ones, while they take more memory space and serve inaccurate results.*

Clustering costs. We use the clustering time (**"Clst."**) to measure the time to cluster the raw trajectories with the non-learned measures or to cluster the embeddings for the learned measures. We only compute the embeddings once and reuse them throughout the clustering process. Thus, the learned measures offer faster clustering (while they require higher space costs for embeddings), as shown in Table 9. For the same reason, the embedding times (**"Emb."**) reported in Table 9 are much lower than those reported in Table 4 where the embeddings are recomputed every time.

Table 10: Clustering accuracy (RI)

	DTW	ERP	Fréchet	Hausdorff
NEUTRAJ	**0.885**	0.811	0.811	**0.874**
T3S	0.832	0.816	0.823	0.820
TrajCL	0.816	**0.817**	**0.827**	0.811
TrjSR	0.708	0.718	0.746	0.799
TrajGAT	0.808	0.646	0.712	0.433

Clustering accuracy. Table 10 shows the accuracy results. Similar to the kNN results in Table 7, NEUTRAJ is strong for DTW, while TrajCL approximates ERP and Fréchet well, and no learned measures can approximate the non-learned ones fully accurately.

4.4.3 Impact of the Size of Trajectory Dataset $|D|$. We vary $|D|$ from 100 to 10,000. Like above, clustering with non-learned measures generally consumes more time than that with the learned ones. We report detailed results in the technical report [5] due to space limit.

5 FUTURE WORK

The empirical study reveals that: (1) The learned measures are slower than the non-learned ones when computing trajectory similarity on the fly. (2) The learned measures are faster than the non-learned ones for kNN queries and clustering when the embeddings can be pre-computed, although there is no accuracy guarantee. (3) Self-attention network-based measures are relatively fast to train to obtain a high accuracy of trajectory similarity.

These results motivate important questions for future work: **(RQ1)** Can learned measures also outperform the non-learned ones for online similarity computation? **(RQ2)** Can a learned measure approximate a wide range of different non-learned measures all with a high accuracy? **(RQ3)** Can learned measures approximate the non-learned ones with accuracy guarantees? **(RQ4)** Can non-learned measures be indexed to achieve higher kNN query efficiency?

Here, we present an attempt to answer the first two questions.

To answer RQ1, we make use a *feedforward neural network* (FFN) as the trajectory encoder, which is arguably the simplest and most efficient deep learning model. We use a two-layer FFN, where the inputs are raw two-dimensional coordinates of the points on a trajectory, for efficiency considerations. We find that such a model can compute trajectory similarity online as fast as Hausdorff, while its HR@50 drops to 0.591 (0.831 for TrajCL) for kNN queries and RI drops to 0.502 (0.811 for TrajCL) for clustering when approximating Hausdorff (full results in the technical report [5]).

Such results show the potential of learned measures to obtain a high efficiency for trajectory similarity computation, thus meeting their original promise, while they also highlight the challenges in obtaining high query and clustering accuracy at the same time.

For RQ2, we argue that *large language models* (LLM) [53, 61] have a strong potential to learn a generic measure for approximating different non-learned measures. LLMs are trained on large text corpora to learn sequential patterns of texts. Since trajectories are also sequences, it would be interesting to study how LLMs can be adapted for numeric sequences. When an LLM is trained with a large trajectory corpus, it could be instruction-tuned to compute different trajectory similarity measures given different input prompts.

In addition, our study has focused on trajectory similarity in a low-dimensional Euclidean space. There are rich studies on trajectories over road networks [29, 33, 56, 66] as well as multi-dimensional time series (high dimensional trajectories) [41, 49, 50, 74]. An empirical study with such data will also be an interesting future work.

6 CONCLUSION

We revisited both non-learned and learned trajectory similarity measures and studied their empirical efficiency comprehensively. The learned measures outperform the non-learned measures as promised in literature, only when trajectory embeddings can be pre-computed. Meanwhile, such measures lack accuracy when approximating the non-learned measures. Among the learned measures, the self-attention ones are the fastest to train and offer the highest accuracy. The non-learned measures do not require pre-computation and are more suitable for one-off similarity computation. These results open up research opportunities in designing advanced learned measures with even higher efficiency and accuracy.

ACKNOWLEDGMENTS

This work is partially supported by Australian Research Council (ARC) Discovery Projects DP230101534 and DP240101006. Gao Cong is supported in part by a Singapore MOE AcRF Tier-2 grant MOE-T2EP20221-0015 and a Singapore MOE AcRF Tier-1 project RT6/23.

REFERENCES

[1] 2013. OpenStreetMap Planet. https://wiki.openstreetmap.org/wiki/Planet.gpx.
[2] 2015. Porto Taxi Trajectory Dataset. https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i.
[3] 2018. DiDi GAIA Open Dataset. https://outreach.didichuxing.com/.
[4] 2020. Trajectory Distance Library. https://github.com/bguillouet/traj-dist.
[5] 2024. Technical Report. https://github.com/changyanchuan/TrajSimiMeasures/blob/master/paper_technical_report.pdf.
[6] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. 1993. Efficient Similarity Search in Sequence Databases. In *International Conference on Foundations of Data Organization and Algorithms*. 69–84.
[7] Helmut Alt. 2009. The Computational Geometry of Comparing Shapes. In *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*. 235–248.
[8] Helmut Alt and Michael Godau. 1995. Computing the Fréchet Distance between Two Polygonal Curves. *International Journal of Computational Geometry & Applications* 5, 01n02 (1995), 75–91.
[9] Petko Bakalov, Marios Hadjieleftheriou, Eamonn Keogh, and Vassilis J Tsotras. 2005. Efficient Trajectory Joins using Symbolic Representations. In *MDM*. 86–93.
[10] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*. 322–331.
[11] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975), 509–517.
[12] Hanlin Cao, Haina Tang, Yulei Wu, Fei Wang, and Yongjun Xu. 2021. On Accurate Computation of Trajectory Similarity via Single Image Super-resolution. In *IJCNN*. 1–9.
[13] Yanchuan Chang, Jianzhong Qi, Yuxuan Liang, and Egemen Tanin. 2023. Contrastive Trajectory Similarity Learning with Dual-Feature Attention. In *ICDE*. 2933–2945.
[14] Yanchuan Chang, Jianzhong Qi, Egemen Tanin, Xingjun Ma, and Hanan Samet. 2021. Sub-Trajectory Similarity Join with Obfuscation. In *SSDBM*. 181–192.
[15] Yanchuan Chang, Egemen Tanin, Xin Cao, and Jianzhong Qi. 2023. Spatial Structure-Aware Road Network Embedding via Graph Contrastive Learning. In *EDBT*. 144–156.
[16] Chao Chen, Chengwu Liao, Xuefeng Xie, Yasha Wang, and Junfeng Zhao. 2019. Trip2Vec: A Deep Embedding Approach for Clustering and Profiling Taxi Trip Purposes. *Personal and Ubiquitous Computing* 23 (2019), 53–66.
[17] Lu Chen, Yunjun Gao, Ziquan Fang, Xiaoye Miao, Christian S. Jensen, and Chenjuan Guo. 2019. Real-time Distributed Co-movement Pattern Detection on Streaming Trajectories. *PVLDB* 12, 10 (2019), 1208–1220.
[18] Lei Chen and Raymond Ng. 2004. On the Marriage of LP-norms and Edit Distance. In *PVLDB*. 792–803.
[19] Lei Chen, M Tamer Özsu, and Vincent Oria. 2005. Robust and Fast Similarity Search for Moving Object Trajectories. In *SIGMOD*. 491–502.
[20] Yuanyi Chen, Peng Yu, Wenwang Chen, Zengwei Zheng, and Minyi Guo. 2021. Embedding-based Similarity Computation for Massive Vehicle Trajectory Data. *IEEE Internet of Things Journal* 9, 6 (2021), 4650–4660.
[21] Ziwen Chen, Ke Li, Silin Zhou, Lisi Chen, and Shuo Shang. 2023. Towards Robust Trajectory Similarity Computation: Representation-based Spatio-temporal Similarity Quantification. *World Wide Web* 26 (2023), 1271–1294.
[22] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In *NIPS Workshop on Deep Learning*.
[23] Ticiana L Coelho Da Silva, Karine Zeitouni, and José AF de Macêdo. 2016. Online Clustering of Trajectory Data Stream. In *MDM*. 112–121.
[24] Liwei Deng, Yan Zhao, Zidan Fu, Hao Sun, Shuncheng Liu, and Kai Zheng. 2022. Efficient Trajectory Similarity Computation with Contrastive Learning. In *CIKM*. 365–374.
[25] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*. 4171–4186.
[26] Anne Driemel, Sariel Har-Peled, and Carola Wenk. 2010. Approximating the Fréchet Distance for Realistic Curves in Near Linear Time. In *SoCG*. 365–374.
[27] Thomas Eiter and Heikki Mannila. 1994. *Computing Discrete Fréchet Distance*. Technical Report. Technical University of Vienna.
[28] Ziquan Fang, Yuntao Du, Lu Chen, Yujia Hu, Yunjun Gao, and Gang Chen. 2021. E2DTC: An End to End Deep Trajectory Clustering Framework via Self-training. In *ICDE*. 696–707.
[29] Ziquan Fang, Yuntao Du, Xinjun Zhu, Lu Chen, Yunjun Gao, and Christian S. Jensen. 2022. Spatio-temporal Trajectory Similarity Learning in Road Networks. In *KDD*. 347–356.
[30] Tao-Yang Fu and Wang-Chien Lee. 2020. Trembr: Exploring Road Networks for Trajectory Representation Learning. *ACM Transactions on Intelligent Systems and Technology* 11, 1 (2020), 10:1–25.
[31] Maxime Gariel, Ashok N Srivastava, and Eric Feron. 2011. Trajectory Clustering and an Application to Airspace Monitoring. *IEEE Transactions on Intelligent Transportation Systems* 12, 4 (2011), 1511–1524.
[32] Antonin Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. 47–57.
[33] Peng Han, Jin Wang, Di Yao, Shuo Shang, and Xiangliang Zhang. 2021. A Graph-based Approach for Trajectory Similarity Computation in Spatial Networks. In *KDD*. 556–564.
[34] Huajun He, Ruiyuan Li, Sijie Ruan, Tianfu He, Jie Bao, Tianrui Li, and Yu Zheng. 2022. TraSS: Efficient Trajectory Similarity Search Based on Key-Value Data Stores. In *ICDE*. 2306–2318.
[35] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.
[36] Yu Jiang, Guoliang Li, Jianhua Feng, and Wen-Syan Li. 2014. String Similarity Joins: An Experimental Evaluation. *PVLDB* 7, 8 (2014), 625–636.
[37] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
[38] Eamonn Keogh and Chotirat Ann Ratanamahatana. 2005. Exact Indexing of Dynamic Time Warping. *Knowledge and Information Systems* 7, 3 (2005), 358–386.
[39] Satoshi Koide, Chuan Xiao, and Yoshiharu Ishikawa. 2020. Fast Subtrajectory Similarity Search in Road Networks under Weighted Edit Distance Constraints. *PVLDB* 13, 12 (2020), 2188–2201.
[40] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory Clustering: A Partition-and-group Framework. In *SIGMOD*. 593–604.
[41] Seok-Lyong Lee, Seok-Ju Chun, Deok-Hwan Kim, Ju-Hong Lee, and Chin-Wan Chung. 2000. Similarity Search for Multidimensional Data Sequences. In *ICDE*. 599–608.
[42] Xiaohui Li, Vaida Ceikute, Christian S. Jensen, and Kian-Lee Tan. 2012. Effective Online Group Discovery in Trajectory Databases. *IEEE Transactions on Knowledge and Data Engineering* 25, 12 (2012), 2752–2766.
[43] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. 2018. Deep Representation Learning for Trajectory Similarity Computation. In *ICDE*. 617–628.
[44] Bin Lin and Jianwen Su. 2008. One Way Distance: For Shape based Similarity Search of Moving Object Trajectories. *GeoInformatica* 12 (2008), 117–142.
[45] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. 2003. A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In *SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. 2–11.
[46] An Liu, Yifan Zhang, Xiangliang Zhang, Guanfeng Liu, Yanan Zhang, Zhixu Li, Lei Zhao, Qing Li, and Xiaofang Zhou. 2022. Representation Learning with Multi-level Attention for Activity Trajectory Similarity Computation. *IEEE Transactions on Knowledge and Data Engineering* 34, 5 (2022), 2387–2400.
[47] Xiang Liu, Xiaoying Tan, Yuchun Guo, Yishuai Chen, and Zhe Zhang. 2022. CSTRM: Contrastive Self-Supervised Trajectory Representation Model for Trajectory Similarity Computation. *Computer Communications* 185 (2022), 159–167.
[48] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and Robust Approximate Nearest Neighbor Search using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2018), 824–836.
[49] Pierre-François Marteau. 2008. Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31, 2 (2008), 306–318.
[50] Michael D Morse and Jignesh M Patel. 2007. An Efficient and Accurate Method for Evaluating Time Series Similarity. In *SIGMOD*. 569–580.
[51] Xavier Olive, Luis Basora, Benoit Viry, and Richard Alligier. 2020. Deep Trajectory Clustering with Autoencoders. In *International Conference for Research in Air Transportation*.
[52] Hae-Sang Park and Chi-Hyuck Jun. 2009. A Simple and Fast Algorithm for K-medoids Clustering. *Expert Systems with Applications* 36, 2 (2009), 3336–3341.
[53] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. *OpenAI* (2018).
[54] Sayan Ranu, Padmanabhan Deepak, Aditya D. Telang, Prasad Deshpande, and Sriram Raghavan. 2015. Indexing and Matching Trajectories under Inconsistent Sampling Rates. In *ICDE*. 999–1010.
[55] Hanan Samet. 1984. The Quadtree and Related Hierarchical Data Structures. *Comput. Surveys* 16, 2 (1984), 187–260.
[56] 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.
[57] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. DITA: Distributed In-memory Trajectory Analytics. In *SIGMOD*. 725–740.
[58] Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. 2020. A Survey of Trajectory Distance Measures and Performance Evaluation. *The VLDB Journal* 29 (2020), 3–32.
[59] Yaguang Tao, Alan Both, Rodrigo I. Silveira, Kevin Buchin, Stef Sijben, Ross S. Purves, Patrick Laube, Dongliang Peng, Kevin Toohey, and Matt Duckham. 2021. A Comparative Analysis of Trajectory Similarity Measures. *GIScience & Remote Sensing* 58, 5 (2021), 643–669.
[60] David Alexander Tedjopurnomo, Xiucheng Li, Zhifeng Bao, Gao Cong, Farhana Choudhury, and A. Kai Qin. 2021. Similar Trajectory Search with Spatio-temporal Deep Representation Learning. *ACM Transactions on Intelligent Systems and*

Technology 12, 6 (2021), 77:1–26.

[61] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971* (2023).

[62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *NIPS*. 6000–6010.

[63] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. 2003. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. In *KDD*. 216–225.

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

[65] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, and Gao Cong. 2021. A Survey on Trajectory Data Management, Analytics, and Learning. *Comput. Surveys* 54, 2 (2021), 39:1–36.

[66] 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.

[67] Tingting Wang, Shixun Huang, Zhifeng Bao, J. Shane Culpepper, and Reza Arablouei. 2022. Representative Routes Discovery from Massive Trajectories. In *KDD*. 4059–4069.

[68] Taizheng Wang, Chunyang Ye, Hui Zhou, Mingwang Ou, and Bo Cheng. 2021. AIS Ship Trajectory Clustering based on Convolutional Auto-encoder. In *Intelligent Systems and Applications*. 529–546.

[69] Zheng Wang, Cheng Long, and Gao Cong. 2023. Similar Sports Play Retrieval with Deep Reinforcement Learning. *IEEE Transactions on Knowledge and Data Engineering* 35, 4 (2023), 4253–4266.

[70] Zheng Wang, Cheng Long, Gao Cong, and Ce Ju. 2019. Effective and Efficient Sports Play Retrieval with Deep Representation Learning. In *KDD*. 499–509.

[71] Phillip GD Ward, Zhen He, Rui Zhang, and Jianzhong Qi. 2014. Real-time Continuous Intersection Joins Over Large Sets of Moving Objects Using Graphic Processing Units. *VLDBJ* 23 (2014), 965–985.

[72] Limin Xiao, Yao Zheng, Wenqi Tang, Guangchao Yao, and Li Ruan. 2013. Parallelizing Dynamic Time Warping Algorithm Using Prefix Computations on GPU. In *IEEE International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing*. 294–299.

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

[74] Kiyoung Yang and Cyrus Shahabi. 2004. A PCA-based Similarity Measure for Multivariate Time Series. In *ACM International Workshop on Multimedia Databases*. 65–74.

[75] 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.

[76] Peilun Yang, Hanchen Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2021. T3S: Effective Representation Learning for Trajectory Similarity Computation. In *ICDE*. 2183–2188.

[77] Sean Bin Yang, Jilin Hu, Chenjuan Guo, Bin Yang, and Christian S. Jensen. 2023. LightPath: Lightweight and Scalable Path Representation Learning. In *KDD*. 2999–3010.

[78] Di Yao, Gao Cong, Chao Zhang, and Jingping Bi. 2019. Computing Trajectory Similarity in Linear Time: A Generic Seed-guided Neural Netric learning approach. In *ICDE*. 1358–1369.

[79] Di Yao, Haonan Hu, Lun Du, Gao Cong, Shi Han, and Jingping Bi. 2022. TrajGAT: A Graph-based Long-term Dependency Modeling Approach for Trajectory Similarity Computation. In *KDD*. 2275–2285.

[80] Di Yao, Chao Zhang, Zhihua Zhu, Jianhui Huang, and Jingping Bi. 2017. Trajectory Clustering via Deep Representation Learning. In *IJCNN*. 3880–3887.

[81] Rex Ying, Jiangwei Pan, Kyle Fox, and Pankaj K. Agarwal. 2016. A Simple Efficient Approximation Algorithm for Dynamic Time Warping. In *SIGSPATIAL*. 21:1–10.

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

[83] Mingxuan Yue, Yaguang Li, Haoze Yang, Ritesh Ahuja, Yao-Yi Chiang, and Cyrus Shahabi. 2019. DETECT: Deep Trajectory Clustering for Mobility-behavior Analysis. In *IEEE International Conference on Big Data*. 988–997.

[84] Hanyuan Zhang, Xingyu Zhang, Qize Jiang, Baihua Zheng, Zhenbang Sun, Weiwei Sun, and Changhu Wang. 2020. Trajectory Similarity Learning with Auxiliary Supervision and Optimal Matching. In *IJCAI*. 11–17.

[85] Bolong Zheng, Lianggui Weng, Xi Zhao, Kai Zeng, Xiaofang Zhou, and Christian S. Jensen. 2021. REPOSE: Distributed Top-k Trajectory Similarity Search with Local Reference Point Tries. In *ICDE*. 708–719.

[86] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. Geolife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Engineering Bulletin* 33, 2 (2010), 32–39.

[87] 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.