E^{2}DTC: An End to End Deep Trajectory Clustering Framework via Self-Training

Ziquan Fang[†], Yuntao Du[†], Lu Chen[†], Yujia Hu[†], Yunjun Gao[†♯], Gang Chen[†]
[†]*College of Computer Science, Zhejiang University, Hangzhou, China*
[♯]*Alibaba–Zhejiang University Joint Institute of Frontier Technologies, Hangzhou, China*
{*zqfang, ytdu, luchen, CharlieHu, gaoyj, cg*}*@zju.edu.cn*

2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00066

***Abstract*—Trajectory clustering has played an essential role in trajectory mining tasks. It serves in a wide range of real-life applications, including transportation, location-based services, behavioral study, and so on. To support trajectory clustering analytics, a plethora of trajectory clustering methods have been proposed, which mainly extend traditional clustering algorithms by using spatio-temporal characteristics of trajectories. However, existing traditional trajectory clustering approaches based on raw trajectory representation highly rely on hand-craft similarity metrics, and can not capture hidden spatial dependencies in trajectory data, which is inefficient and inflexible for clustering analysis. To this end, we propose an end-to-end deep trajectory clustering framework via self-training, termed as E^{2}DTC, inspired by the data-driven capabilities of deep neural networks. E^{2}DTC does not require any additional manual feature extraction operations, and can be easily adapted for trajectory clustering analytics on any trajectory dataset. Extensive experimental evaluations on three real-life datasets show that our framework E^{2}DTC achieves superior accuracy and efficiency, compared with classical clustering methods (i.e., K-Medoids) and state-of-the-art neural-network based approaches (i.e., t2vec).**

I. Introduction

With the proliferation of GPS-enable devices and mobile computing services, massive volumes of trajectory data are collected to capture the mobility of vehicles, people, and animals [41]. Analyzing the trajectory data has provoked various research topics across different fields, such as behavioral study [17], transportation [32], location-based services [40], and urban computing [42]. Trajectory clustering, an essential and popular trajectory data analytics task to discover similar trajectory groups, plays a vital role in many real-world applications including animal tracking [25], hot-area detection [6], mobility pattern mining [2], and abnormal activity prediction [7], to name just a few.

A plethora of trajectory clustering methods has been proposed, which mainly extend standard clustering algorithms by using spatio-temporal properties. More specifically, they first adopt an existing or modified distance metric (e.g., LCSS) to compute the similarities between trajectories, and then apply classic clustering technologies (e.g., k-means) to cluster trajectories. However, existing methods are far from enough due to the following three issues.

First, existing trajectory clustering methods are limited to raw-trajectory-based representations. The raw-trajectory-based representations (i.e., trajectory points, trajectory segments, and the entire trajectory) are not able to well capture the spatial characteristics of trajectories, which results in a negative effect on the clustering results. A trajectory of a moving object is actually a continuous curve in the 2D (or 3D) space. However, it is usually collected in the form of sample GPS coordinates [41] in our daily life. When the sampling rate is low or non-uniform, it is hard to represent the true behaviors of a moving object with limited GPS points, as the underlying routes between discrete GPS points are unknown [16]. Besides, the raw-trajectory-based representations can be sensitive to noise, which could arise in urban canyons due to low positioning signals. To address the above issues, trajectory via deep learning representation based on neural networks can be used [30], which aims to embed raw trajectories to vectors and capture the hidden information of trajectory data. Based on the deep representations, trajectory mining tasks (such as classification, similarity computation, and traffic flow prediction) can be implemented in the embedded vector space. Nonetheless, little attention has been put on deep representations for trajectory clustering analytics. Hence, we target at learning a deep cluster-oriented representation and performing unsupervised trajectory clustering simultaneously.

Second, the result quality of existing trajectory clustering approaches mostly rely on the distance (or similarity) metrics. However, it is difficult to find a proper distance metric for miscellaneous trajectory datasets. On one hand, trajectories may have various properties such as different lengths or sampling rates, which brings challenges when computing trajectory similarities using existing pair-matching metrics, including point-based (e.g., EDR [3], LCSS [27], and DTW [34]) or shape-based (e.g., Hausdorff) similarity metrics. On the other hand, all above similarity metrics either focus on the local spatial dependencies (i.e., point-based) or emphasize global relations between trajectories (i.e., shape-based), making it more challenging to define a proper similarity metric to consider both local and global spatial characteristics. In contrast, we choose to embed the raw trajectories into the vector space and conduct deep clustering simutaniously. In the embedded vector space, the Euclidean metric can be easily adopted for computation. As illustrated in the experimental evaluations, distance metric based classic clustering approaches may have totally contrast performance across different trajectory datasets, which exposes the high dependencies to distance measures of traditional clustering approaches. Nevertheless, the deep clustering method always shows superior performance to them.

978-1-7281-9184-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDE51399.2021.00066

Last but not the least, the traditional trajectory clustering processing pipeline is inflexible to support various trajectory datasets. The processing pipeline usually consists of several steps, including (i) data pre-processing, (ii) similarity definition and computation, and (iii) applying clustering algorithms. The traditional clustering pipeline that established for one trajectory dataset may be not optimal for another, as different trajectory datasets have different spatial properties [41]. Thus, the above procedures have to be adapted and re-designed for different trajectory datasets, which is inefficient and inflexible. In contrast, we aim to propose an end-to-end deep trajectory clustering framework trained by neural networks. Once the deep clustering model has been finely established and trained, trajectory clustering analytics tasks can be efficiently executed to response clustering requests across various datasets. In this paper, we realize an unsupervised clustering by utilizing the data-driven capabilities of deep neural networks without handcrafted feature extraction operations.

In summary, for the long-established traditional clustering methods, we need to manually define a similarity metric and choose a particular clustering algorithm in the raw-trajectory-based data space, which prevents it from automatically capturing deep spatial characteristics hidden in trajectory data. Inspired by the success of RNNs at capturing hidden dependencies for sequential data in natural language processing [5], speech recognition [39], and deep trajectory representations [16], we revisit the trajectory clustering problem, and propose an end-to-end deep trajectory clustering framework in this paper, termed as E^{2}DTC. Here, the end-to-end means no separations and manual feature extractions during the entire trajectory clustering pipeline. Given an input of a pure trajectory dataset, our goal is to embed each raw trajectory into a cluster-oriented representation while jointly doing trajectory clustering at the same time.

To address the above issues, we present an end-to-end deep trajectory clustering framework, i.e., E^{2}DTC, via self-training, which clusters trajectories in the embedded latent feature space. Specifically, our framework E^{2}DTC mainly contains three phases: (i) trajectory embedding of raw data, (ii) pre-training for initializing trajectory representations, and (iii) self-training that jointly learns the cluster-oriented representation and performs clustering analytics. To sum up, this paper makes the following major contributions:

- *Deep trajectory clustering.* We propose the first end-to-end deep trajectory clustering framework E^{2}DTC via self-training without manual feature extractions, which fully utilizes the data mining capabilities of neural networks. Once the E^{2}DTC has been finely trained, it can be efficiently adopted for trajectory clustering requests on different datasets, which avoids processing from scratch like traditional clustering approaches.
- *Three jointly training loss.* To further enhance the clustering performance of the neural network, we present the three-jointly-training loss in the self-training stage, including the reconstruction loss L_r, the clustering loss L_c, and the triplet loss L_t.
- *Ground-truth providing.* Since there are no public trajectory datasets with spatial labels for clustering quality evaluation, we stand forward and develop a ground-truth generation algorithm for real-life trajectory datasets. The evaluated datasets with ground-truth are released for further research studies.
- *Extensive experiments.* We conduct massive experimental evaluations on three real-world trajectory datasets to offer deep insights that E^{2}DTC outperforms the existing traditional and neural-network based clustering approaches in terms of both accuracy and efficiency. As for the E^{2}DTC model itself, we also provide extensive evaluations on its learning process and model robustness from the aspects of both quantitative and qualitative.

The rest of the paper is organized as follows. Section II reviews the related work, and Section III covers the background. Section IV presents the problem definitions. Sections V and VI detail our framework and ground truth generation. The experimental results are reported in Section VII. Finally, Section VIII concludes the paper and offers potential research directions. All implementation codes and corresponding datasets have been released for further studies.

II. Related Work

In this section, we briefly review the related work on raw-trajectory-based clustering and deep trajectory representation learning, respectively.

A. Raw-Trajectory-Based Clustering

Trajectory clustering has always been playing a foundational role in trip classification [13], moving pattern detection [2], abnormal prediction [37], and many other trajectory-based analysis tasks [1]. Trajectory clustering aims at grouping trajectories into different clusters, where trajectories in the same cluster exhibit similar movement characteristics and different from those trajectories in other clusters [23]. A lot of studies based on raw trajectory representations (i.e., the entire trajectory [8], trajectory segments [14], or trajectory points [2]) has been proposed to support trajectory clustering analytics. They first define proper distance metrics and then apply classic clustering algorithms, such as k-means. For example, Gaffney et al. [8] develop the EM algorithm to cluster trajectories based on the overall Euclidean distance between two entire trajectories, resulting into overlooking significant local clusters that might exist only for portions of their lifespan. To this end, Lee et al. [14] first partition trajectories into trajectory segments and then execute clustering process based on Hausdorff distance between trajectory segments. Recently, Chen et al. [2] study the moving pattern detection in the streaming environment and run DBSCAN algorithm based on L1-norm distance in each snapshot (i.e., a set of trajectory points) to detect dynamic clusters. More detailed trajectory clustering approaches that can be found in [35] are omitted here, due to the limited space. The aforementioned trajectory clustering methods mainly focus on defining the proper distance metrics used for trajectory similarity computations and then performing clustering analysis in the raw data space. As discussed in Section I, the raw-trajectory-based clustering methods often suffer from complex and various properties of trajectories, such as variable sampling rates, variable trajectory lengths, and inevitable GPS noise.

In addition, there also exist trajectory clustering methods [1] that do not rely on the spatial distance metrics of trajectories, but detect trajectory clusters based on similar destinations, similar sources, and similar duration. However, this type of methods is not our focus. In contrast, we aim to investigate the spatial dependence based similarities (i.e., local points and global shape), based on which the clustering is performed.

B. Deep Trajectory Representation Learning

The essential problem behind traditional clustering methods is that both the similarity definition and clustering processing are based on raw data representations, which tend to ignore potential features of trajectories. Motivated by this, we aim to develop a deep trajectory representation instead of the raw trajectory representation, and then, we execute trajectory clustering analytics in the latent feature space. Actually, representation learning for specific tasks has been widely studied in machine learning field [15]. Inspired by the success of RNN-based models in capturing order information emerging in time-serial data [22], developing deep representations for specific trajectory mining tasks has attracted much attention, such as deep-based trajectory classification [13], deep-based traffic flow prediction [28], deep-based trajectory similarity computation [16], etc. Li et al. [16] develop the Seq2Seq-based method (i.e., t2vec) to learn trajectory representations for similarity computation, which has been proved to be robust to variable sampling rates and noise. As the similarity computation is essential to trajectory clustering tasks, we extend t2vec for trajectory clustering by applying the k-means method based on the t2vec representation, termed as t2vec + k-means, which is used as one of baselines for experimental comparison. Next, we mainly detail related work about trajectory clustering based on deep representation learning models.

Yao et al. [16] study trajectory clustering via Seq2Seq Auto-Encoder model. Specifically, they employ a sliding window algorithm to capture moving features (i.e., spatial location, speed, direction, etc.) of trajectories and then apply classic k-means clustering method. Wang et al. [31] annotate trajectories with movement labels (such as “Enter”, “Leave”, “Stop”, and “Move”), and use these labeled data to detect events from trajectories. Nevertheless, they both need a careful and manual feature-extraction operation and use these features as inputs to feed into neural networks, and thus, they not only have higher requirements for domain-specific knowledge but also can loss potential information in the manual extraction process. More recently, Yue et al. [36] explore the deep trajectory clustering in terms of extracted POIs in trajectories. In contrast, our E^{2}DTC focus on a more general spatial-dependency based deep trajectory clustering on raw trajectory data, which has better plasticity and scalability. As a summary, E^{2}DTC is the first data-driven model, and utilizes raw trajectories as inputs for neural networks to avoid hidden data information loss. In addition, we aim to build an end-to-end deep trajectory clustering framework without pre-processing (e.g., labeling) or manual feature extractions.

TABLE I
SYMBOLS AND DESCRIPTION

Notation	Description
x	An input sequence of trajectory tokens
x_t	The token at position t of x
y	An output sequence of tokens
y_t	The token at position t of y
v_x	The embedded (feature) vector of x
V	Vocabulary
h_t	The hidden state (vector) at position t
p	Local probability at a certain position
$P(y\|x)$	The conditional probability of y given x
T	A raw trajectory generated by a moving object
v_T	The embedded (feature) vector of trajectory T
O	The raw trajectory data space that contains all T
Z	The latent feature space that contains all v_T
f_θ	A target deep neural network with parameters θ
$\{C_j\}_{j=1}^{k}$	The learned k clusters $C_j \in Z$
β	The weight of clustering loss
γ	The effect of triplet loss

III. Background

The Seq2Seq (i.e., sequence to sequence) framework has shown its success to learn general representation of texts, time-series and trajectories. And therefore, we adopt this architecture for deep trajectory clustering with three carefully designed loss functions. In this section, we introduce the Seq2Seq framework and deep clustering, respectively. For ease of understanding, the frequently used notations are summarized in Table I.

A. Seq2Seq Model

The Seq2Seq framework is widely used to capture the hidden information of sequential data such as text and time series. Taking machine translation as an example, given an English text sequence “Trajectory clustering is important” as the input, and the model will output a French text sequence “Le regroupement des trajectoires est important”. The Seq2Seq model is composed of two components, i.e., an encoder and a decoder, as depicted in Figure 1. Both of the two components are essentially based on Recurrent Neural Networks (RNNs), where the encoder is used to compress and encode the input data sequence, and the decoder is used to generate the output data sequence conditioned on the encoded representation. Given the input data sequence $x = \left\{x_1, x_2, \ldots, x_{|x|}\right\}$ and the output data sequence $y = \left\{y_1, y_2, \ldots, y_{|y|}\right\}$, where x_t and y_t denote the t-th element in the corresponding data sequence, respectively. The underlying problem of translation example shown in Figure 1 is to predict the conditional probability $P(y|x)$, i.e., the probability of the output y given an input x. In this subsection, we will illustrate how the Seq2Seq via

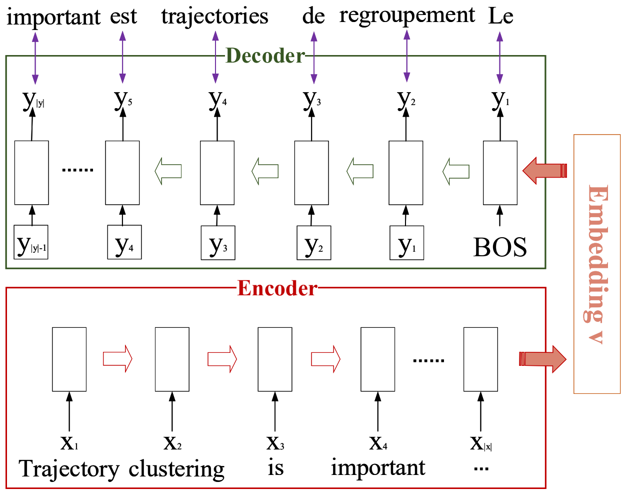

important est trajectories de regroupement Le
Decoder
$y_{|y|}$ y_5 y_4 y_3 y_2 y_1
......
$y_{|y|-1}$ y_4 y_3 y_2 y_1 BOS
Embedding v
Encoder
......
x_1 x_2 x_3 x_4 $x_{|x|}$
Trajectory clustering is important ...

Fig. 1. Seq2Seq Architecture with A Translation Example

encoder-decoder based framework works through encoding, decoding, and training procedures. Note that, $|x|$ and $|y|$ represent data sequence length.

1) Encoding: The encoder is used to encode the input data sequence x into a fixed-length representative vector v_x, which preserves the sequential information of x. Recurrent neural networks such as RNN, LSTM, and GRU [4] can be used as the encoder. Given the input sequence data x, the encoder reads x sequentially and the current hidden state h_t of RNNs is updated as follows, where $1 \leq t \leq |x|$, $\boldsymbol{f}(\cdot,\cdot)$ indicates the RNNs forward computation. Note that, the superscript e and d in the hidden state are utilized to distinguish encoding and decoding phases.

$$h_t^e = \boldsymbol{f}\left(h_{t-1}^e, x_t\right) \tag{1}$$

According to Formula 1, RNNs update the hidden state h_t based on the current input token x_t and the previous hidden state h_{t-1}. The hidden state h_t acts as an internal memory at position t that enables the network to capture sequential dependencies of input data $x = \{x_1, x_2, \ldots, x_t\}$. After the last token x_t is processed, the hidden state h_t is obtained and can be used as the hidden representation v_x for the data sequence x. Therefore, the encoder has compressed the sequential hidden information of the input data sequence x into a latent vector v_x.

2) Decoding: The decoder is used to decode the latent representation v_x and reconstruct another output sequence y. As depicted in Figure 1, the decoder first generates the output y_1 by taking v_x as the initialized hidden state, and then continuously generates $y_2, y_3, \ldots, y_t$ in order. Given the hidden representation v_x, the previous hidden state h_{t-1}^d, and the output token y_{t-1}, the current hidden state h_t^d of RNNs-based decoder is updated by:

$$h_t^d = \boldsymbol{f}\left(h_{t-1}^d, y_{t-1}\right) \tag{2}$$

The conditional probability of decoding y_t depends on the previous output sequence $\{y_1, y_2, \ldots, y_{t-1}\}$ and the latent representation v_x. With a customized output layer and softmax operation $\boldsymbol{g}()$, the probability distribution of y_t of output sequence y at position t is calculated as follows.

$$p\left(y_t \mid y_1, \ldots, y_{t-1}, v_x\right) = \boldsymbol{g}\left(h_t^d, y_{t-1}, v_x\right) \tag{3}$$

Given the single word vector output probability at position t shown above, we can get the probability of the entire data sequence with the multiplication of the probability of occurrence of each word vector:

$$\begin{aligned} P(y \mid x) &= P\left(y_1, \ldots, y_{|y|} \mid x_1, \ldots, x_{|x|}\right) \\ &= p\left(y_1 \mid x\right) \prod_{t=2}^{|y|} p\left(y_t \mid y_1, \ldots, y_{t-1}, v_t\right) \end{aligned} \tag{4}$$

3) Training: The training goal is to minimize the divergence between y and x, as denoted in purple in Figure 1. Taking the translation as an example, the divergence between "Trajectory clustering is important" and "Le regroupement de trajectoires n'est pas important" is larger than that between "Trajectory clustering is important" and "Le regroupement de trajectoires est pas important". To train such a Seq2Seq model, a loss function is required to describe the optimization objective. When the sequence encoder-decoder model is employed in natural language processing, e.g., in neural machine translation [26], Negative Log Likelihood (NLL) loss is widely used to minimize the negative log likelihood function for tokens in the target sentence as follows:

$$\begin{aligned} L &= -\log P(y \mid x) \\ &= -\log \prod_{t=1} p\left(y_t \mid y_1, \ldots, y_{t-1}; x\right) \end{aligned} \tag{5}$$

Equation 5 is the loss function designed for one input data sequence x. Given a training dataset, we aim to maximize the sum of the probabilities of all data sequences.

B. Deep Clustering

Based on the learned representation v, clustering algorithms are applied in the latent dimensional space instead of on the raw data representation x. The existing deep clustering methods can be divided into two categories [20]. The methods in the first category first extract features to learn deep embedded representations and then perform clustering. Since the representation learning and clustering analysis are separated, the extracted features during representation learning may not be suitable for subsequent clustering. The second category jointly optimizes the embedding v and clustering results $\{C_j\}_{j=1}^k$, and in this way, a cluster-oriented representation v is naturally developed. In this paper, we follow the second category, and adopt the Seq2Seq based architecture. In addition, the training objective of general deep clustering is:

$$L_{deep\ clustering} = L_r + \beta L_c \tag{6}$$

where L_r is the reconstruction loss used for deep representation learning (such as Equation 5), L_c is the clustering loss, and β is hyper-parameters to control the weight of the clustering loss. The reconstruction loss ensures that the learned embeddings keep the original structure of the input data, preventing it from losing hidden information in the latent space. The clustering loss aims to make the embeddings in the latent space more discriminative in order to obtain a cluster-oriented representation. Although the deep clustering methods have been used for several unsupervised learning tasks, such as images clustering [33], and time series sequence clustering [20], little literature studies our problem (i.e., the deep trajectory clustering with spatial dependencies).

IV. Problem Definitions

In this section, we present formal definitions related to the deep trajectory clustering. A raw trajectory generated by a moving object is usually represented as a time ordered sequence of sample GPS points, i.e., $T = \{p_1, p_2, \ldots, p_{|T|}\}$. Each point $p_i \in T (1 \leq i \leq |T|)$ consists of a pair of spatial coordinates (i.e., latitude and longitude) and its observed timestamp. Here, $|T|$ denotes the length of a trajectory, i.e., the number of sampled points.

The raw-trajectory-based representations usually are not able to capture the spatio-temporal dependencies of trajectories due to the complex and various properties (such as variable sampling rates and noise), as discussed in Section I. Therefore, we aim to study trajectory clustering analytics based on deep trajectory representations.

Definition 1: **(Trajectory Representation Learning)**. *Trajectory representation learning* trains a deep neural network (i.e., $f_\theta : O \rightarrow Z$) to transform each raw trajectory $T \in O$ to the trajectory representation $v_T \in Z$, where θ denotes the parameters for neural network, O represents a set of raw trajectories, and Z denotes a set of trajectory representations.

Note that, the learned trajectory representations must be robust to complex trajectories with non-uniform, low sampling rates and noisy sample points. Based on this, deep trajectory clustering has an essential clustering quality assurance. We proceed to define deep trajectory clustering.

Definition 2: **(Deep Trajectory Clustering)**. Given a set of trajectories O, *deep trajectory clustering* simultaneously learns a mapping from raw trajectory to deep representations while iteratively optimizing a clustering objective.

Here, we denote each learned cluster by using its centroid $C_j (1 \leq j \leq k)$. The goal of deep trajectory clustering is to minimize the similarity between trajectories in different clusters and to maximize the similarity between trajectories in the same cluster. It is worth mentioning that, our E^{2}DTC model **jointly** learns trajectory representations, and conducts trajectory clustering in a unified deep neural network, rather than doing these tasks with separated models.

V. Our Framework E^{2}DTC

In this section, we first give an overview of our framework E^{2}DTC, and introduce the detailed methods respectively.

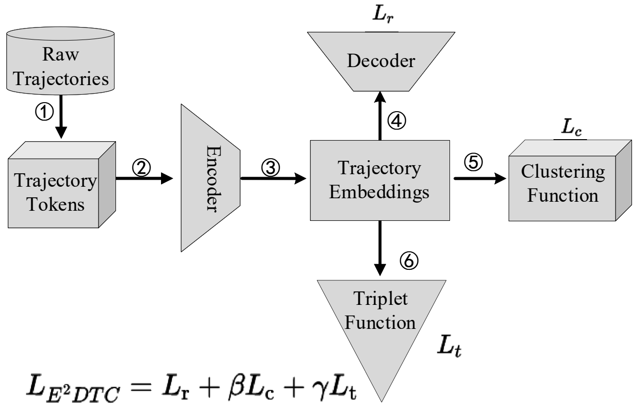


Fig. 2. An Overview of E^{2}DTC Framework

A. Overview

As depicted in Figure 2, the framework of E^{2}DTC is established on the encoder-decoder architecture, which consists of three main processing phases: (i) trajectory embedding; (ii) pre-training; and (iii) self-training. In the first trajectory embedding phase (including ① in Fig. 2), each raw trajectory T is transformed into a sequence of discrete "tokens" as data input. In the second pre-training phase (including ②, ③, and ④ in Fig. 2), inspired by t2vec method [16], we utilize Seq2Seq model to learn initial trajectory representation v_T, which is robust to variable sampling rates and noisy points of raw trajectory data. During the last deep trajectory clustering stage (including ②, ③, ④, ⑤, and ⑥ in Fig. 2), we apply self-training using soft cluster assignments to jointly tune the deep neural networks θ considering both deep representation v_T and deep clustering $C_j (1 \leq j \leq k)$. Note that, we only show the forward propagation of E^{2}DTC in Figure 2.

The entire training of E^{2}DTC can provide a robust encoder, which is finely tuned for cluster-oriented representations and can be used for unsupervised trajectory clustering tasks on other trajectory datasets without additional manual feature extraction operations. In other words, E^{2}DTC is an end-to-end deep trajectory clustering framework to fully liberate the data-driven capabilities of trajectories. Next, we detail the above three processing phases, respectively.

B. Trajectory Embedding

The data input of E^{2}DTC should be sequences of discrete tokens (e.g., like a sentence of words in Fig. 1). Hence, we first convert each trajectory T with continuous GPS points to a sequence of discrete tokens, where we call this process as trajectory discretization. A common discretization strategy in trajectory data analysis is to divide the entire spatial space into disjoint equal-sized grid cells. All the grid cells form the space vocabulary V. Each grid cell is treated as a token and labeled with a vocabulary Id. Based on this, each GPS point is converted to the Id of a grid cell where it locates in. Thus, the raw trajectory T has been transformed into a sequence of vocabulary Ids for subsequent trajectory embedding.

Based on these transformed discrete spatial points, we utilize the popular skip-gram model to learn their representations that maximizes $\frac{1}{N}\sum_{t=1}^{N}\sum_{-c\leq j\leq c, j\neq 0}\log P\left(g_{t+j}\mid g_t\right)$, where g_t denotes the current grid cell, g_{t+j} denotes a neighbor cell of g_t, and c is the number of neighbor cells. Two neighbor cells should have similar representations. The probability can be calculated using the softmax function as:

$$P\left(g_{t+j}\mid g_t\right)=\frac{\exp\left(v_{g_{t+j}}^{T}v_{g_t}\right)}{\sum_{g_i\in V}\exp\left(v_{g_t}^{T}v_{g_t}\right)} \tag{7}$$

where v_{g_t} represents the vector representation of g_t. Consequently, the raw trajectory T can be transformed to a spatial embedding v_T: $T\rightarrow v=\{v_{g_1},\cdots,v_{g_t},\cdots,v_{g_{|T|}}\}$.

C. Pre-Training

In the second phase, E^{2}DTC iteratively encodes and reconstructs trajectories to form an initial trajectory representation v_T. Specifically, E^{2}DTC first uses an encoder to transform trajectories into the latent space (i.e., $T\rightarrow v_T$), and then, it uses a decoder to reconstruct the deep trajectory representation to natural data space (i.e., $v_T\rightarrow T'$). Here, T and T' represent the input and learning target, respectively.

We follow the t2vec method [16] that drops points and adds noise during training process, which has been proved effective to capture the spatial dependencies of complex trajectories. Given a trajectory T_a, we first randomly drop sample points from T_a with a dropping rate r_1 to get a down-sampled trajectory T_c. Next, we further distort each down-sampled T_c with a distorting rate r_2 to get T'_a, i.e., we sample points in T_c with a rate r_2 and then add a Gaussian noise to each sampled point. Thereafter, we can get a collection of trajectory pairs (T'_a,T_a), where T_a is the original trajectory and T'_a is obtained by randomly dropping points and adding noise from T_a. In our experiments, we set the dropping rate $r_1\in(0,0.2,0.4,0.6)$ and the distorting rate $r_2\in(0,0.2,0.4,0.6)$. Hence, given a trajectory T_a, we can get 16 pairs of (T'_a,T_a).

The representation learned from the low-sampling trajectory is expected to be able to recover the high-sampling trajectory. Consequently, we use T'_a as the input and T_a as the learning target. Simply adopting the loss function in Equation 5 is not enough as it ignores spatial relationship of trajectory data. To this end, we utilize a widely used cell weight based spatial proximity aware loss [16], [18], [38] as follow.

$$L_r=-\sum_{t=1}^{|T|}\sum_{g_t\in V}w_{g_t}\log\frac{\exp\left(W_{g_t}^{\top}h_t\right)}{\sum_{v\in V}\exp\left(W_v^{\top}h_t\right)} \tag{8}$$

where W^T is the projection matrix that projects h_t from the hidden state space into the vocabulary space, and the cell weight w_{g_t} equals to $\frac{\exp\left(-\left\|v_{g_t}-v_{g'_t}\right\|_2/\alpha\right)}{\sum_{v\in V}\exp\left(-\left\|v-v_{g'_t}\right\|_2/\alpha\right)}$. Here, g_t and g'_t denote the predicted Cell and the ground truth Cell, v_{g_t} and $v_{g'_t}$ are vector representations of g_t and g'_t, respectively. According to the definition of w_{g_t}, a larger weight is attached to a predicted cell that is close to the true target, and a small α penalizes far away cells heavily. When $\alpha\rightarrow 0$, the loss function degrades to NLL loss function in Equation 5. To reduce the computational cost without scarifying the performance, we use the k nearest neighbors of the target cell g'_t, rather than all cells in vocabulary V for avoiding unnecessary computation, whose size is far more smaller than V but collects the most relevant neighbors around g'_t.

Therefore, we can get an initial estimate of the non-linear mapping (i.e., f_θ) that forms a deep trajectory representation v. Based on the initial trajectory embeddings, a standard k-means clustering algorithm is applied in the feature space Z to obtain k initial cluster centroids $\{C_j\}_{j=1}^{k}$.

D. Self-Training

Given an initial estimate of the non-linear mapping f_θ and the cluster centroids $\{C_j\}_{j=1}^{k}$, we can simultaneously refine the trajectory cluster assignments to make clusters more discriminative via self-clustering technology.

Self Clustering. The design of the self-clustering is based on the assumption "in the initial clusters, points very close to the centroid are likely to be correctly predicted/clustered" [33], i.e., points close to the centroid are likely with high confidence predictions. The model improves the overall clustering iteratively, by aligning the low confidence counterparts. In particular, the clustering refinement iteratively minimizes the difference between the current cluster distribution Q and an auxiliary target cluster distribution P, which is a distribution derived from high confidence predictions of Q. Intuitively, Q describes the probability of an augmented trajectory belonging to the k clusters. Whereas, P is generated by re-enforcing the probability of high-confidence trajectories. Although the self-clustering mechanism has shown its success in many deep clustering literature [10], [20], [22], [33], little work has applied it for unsupervised deep trajectory clustering.

E^{2}DTC uses the unsupervised self-clustering to conduct deep trajectory clustering in three steps: (i) computing the probability of assignments (i.e., Q) between the embedded trajectories v_T and the cluster centroids (i.e., $\{C_j\}_{j=1}^{k}$); (ii) calculating the auxiliary target probability distribution from Q; and (iii) updating the deep mapping parameters f_θ and the cluster centroids based on Kullback-Leibler(*KL*) divergence between P and Q. The above processes are iteratively executed until a convergence criterion is satisfied. We detail three steps respectively, namely soft assignment, target calculating, and *KL*-divergence minimization, as below.

(i) Soft Assignment. In the first step, we compute a soft cluster assignment for each deep embedded trajectory v_T. Here, we use the Student's t-distribution [11] as a kernel to measure the similarity between each embedded trajectory v_T and the cluster centroid C_j:

$$q_{ij}=\frac{\left(1+\|v_i-C_j\|^2\right)^{-1}}{\sum_{j'}\left(1+\|v_i-C_{j'}\|^2\right)^{-1}} \tag{9}$$

where v_i denotes the embedded trajectory T_i, q_{ij} can be interpreted as the probability of assigning trajectory T_i to cluster j. In other words, q_{ij} can be used as a soft assignment of embeddings to centroids. The encoder is then fine-tuned to match embedded trajectory v_i to a target cluster C_j.

(ii) Target Calculating. In the second step, we calculate the auxiliary target distribution P that has "stricter" probabilities compared to the assignment probability distribution Q. It aims to improve cluster purity and put more emphasis on data points assigned with high confidence. This also prevents large clusters from distorting the hidden feature space. The probabilities p_{ij} in P are calculated as:

$$p_{ij} = \frac{q_{ij}^2 / \sum_{i'} q_{i'j}}{\sum_{j'} \left(q_{ij'}^2 / \sum_{i'} q_{i'j'} \right)} \tag{10}$$

Note that ${q_{ij}}^2$ is normalized by the soft cluster frequencies $(\sum_{i'} q_{i'j})$.

(iii) KL-Divergence Minimization. In the last step, we iteratively refine the clusters by learning from their high confidence assignments with the help of an auxiliary target distribution. Specifically, E^{2}DTC is trained by matching the soft assignment to the target distribution. To this end, the KL-divergence between the soft assignments Q and the auxiliary distribution P is then used as the clustering objective loss L_c, as defined:

$$L_{\mathrm{c}} = KL(P \| Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{11}$$

Recall that we simultaneously refine the cluster-oriented representation and deep clustering. Thus, the overall training loss of E^{2}DTC is defined as below:

$$L_{E^2DTC} = L_{\mathrm{r}} + \beta L_{\mathrm{c}} \tag{12}$$

where L_r is the modified reconstruction loss in Equation 8, and L_c is the clustering loss in Equation 11.

E^{2}DTC jointly optimizes the cluster centroids C_j and neural-network parameters θ using Adam stochastic gradient descent [12] to minimize the clustering loss in Equation 12.

(iv) Improved Deep Trajectory Clustering. To generate more separated clusters while making the elements in the same cluster much tighter, we further apply a Triplet loss to improve the quality of deep trajectory representations for clustering. The idea of Triplet loss is simple. Given three samples (i.e., an anchor, a positive, and a negative), the feature distance between the similar identities should be as small as possible (an anchor vs. a positive), and the feature distance between different identities (an anchor vs. a negative) should be as large as possible. The Triplet loss has been used widely in image classification and clustering analysis [19], [29]. To the best of our knowledge, this is the first attempt to utilize Triplet loss in trajectory clustering tasks. In our E^{2}DTC, the Triplet loss is defined as:

$$L_t = \sum_i^N \left[\left\| f\left(v_i^a\right) - f\left(v_i^p\right) \right\|_2^2 - \left\| f\left(v_i^a\right) - f\left(v_i^n\right) \right\|_2^2 + \alpha \right]_+ \tag{13}$$

Algorithm 1: Deep Trajectory Clustering

```
Input: discretized trajectory dataset $\mathcal{T}$, number of clusters $k$,
       maximum number of iterations $MaxIter_1$ of
       pre-training, maximum number of iterations $MaxIter_2$
       of self-training, stopping threshold $\delta$
Output: clustering results $\mathcal{C} = \{C_j\}_{j=1}^{k}$
foreach $iter \in \{0, 1, ..., MaxIter_1\}$ do
    initialize embeddings $\{v_{T_i} = f_\theta(T_i)\}_{i=1}^{|\mathcal{T}|}$ // Equation 8
foreach $iter \in \{0, 1, ..., MaxIter_2\}$ do
    update embeddings $\{v_{T_i} = f_\theta(T_i)\}_{i=1}^{|\mathcal{T}|}$ // Equation 8
    update $Q$ and $P$ // Equations 9 and 10
    $\mathcal{C}' = \mathcal{C}$ // save previous cluster assignments
    compute new cluster assignments $\mathcal{C}$ // Equation 11
    if $\sum_{i=1}^{k}(C_i' \neq C_i) \leq \delta$ then
        break // Stop Training
    update $\theta$ of E$^2$DTC // Equation 14
return $\mathcal{C}$
```

where v_i^a is an embedded trajectory that used as an anchor, v_i^p is the noisy point around v_i^a (v_i^a and v_i^p tend to belong to the same cluster), and the remaining points are negative v_i^n.

E. Overall Loss Function

Finally, we propose the overall training loss function of E^{2}DTC by summing up the reconstruction loss L_r (i.e., Equation 8), the clustering L_c loss (i.e., Equation 11), and the Triplet loss L_t (i.e., Equation 13).

$$L_{E^2DTC} = L_{\mathrm{r}} + \beta L_{\mathrm{c}} + \gamma L_{\mathrm{t}} \tag{14}$$

where β and γ are coefficients. Specifically, L_r captures the hidden features of the input, L_r encourages the representations to form cluster structures, and L_t enhances the clustering ability of the encoder. The detailed training process of E^{2}DTC is presented in Algorithm 1.

VI. Ground Truth Generation

Existing trajectory clustering methods mostly focus on improving the efficiency and scalability, while little attention has been put on evaluating the clustering qualities due to the lack of ground truth clusters. To the best of our knowledge, there are no public trajectory datasets with spatial cluster labels, which makes it challenging for trajectory clustering evaluation. Hence, we carefully design a ground-truth generation algorithm, based on which three real-world trajectory datasets with labeling are public and provided to encourage more researches in this trajectory clustering field.

In this section, we detail how we generate ground truth based on a pure trajectory dataset without any spatial cluster labels. The generation can be divided into two steps including cluster center selection and cluster label assignment.

Step I: Cluster center selection. We first visualize all trajectories by georeferencing all GPS points on the map, and carefully select certain POIs that most frequently visited as the cluster centers, which considers both urban and rural areas.

Step II: Cluster label assignment. Mobile behavior-based GPS trajectories can be very different at different times,

Algorithm 2: Ground Truth Generation

```
   Input: trajectory dataset $\mathcal{T}$, the radius ratio $\sigma$ $(0 < \sigma \leq 1)$,
          the fallen threshold $\lambda$ $(0<\lambda \leq 1)$
   Output: trajectory dataset with cluster labels $\mathcal{T}'$
 1 initialize $k$ POIs as cluster centers $\mathcal{C} = \{C_j\}_{j=1}^{k}$ on the map
 2 $radius = min\{Distance(C_i, C_j)\}, (1 \leq j \leq k \ \& \ j \neq i)$
 3 for each $C_j \in \mathcal{C}$ do
 4     $C_j.r = radius \times \sigma$
 5 for each $C_j \in \mathcal{C}$ do
 6     for each $T_i \in \mathcal{T}$ do
 7         $fallenPoints = rangeQuery(T_i, C_j)$
 8         $T_i.fallenRate = fallenPoints/|T_i|$
 9         if $T_i.fallenRate \geq \lambda$ then
10             insert $(T_i, C_j)$ in $\mathcal{T}'$
11             break
12 return $\mathcal{T}'$
```

locations, and regions. Thus, it is not sensible to generate all clusters with the same size and shape. To this end, we set two essential parameters to control the cluster inner property for better ingratiating the real complex situation.

- Radius ratio σ, which controls the boundary/area of a cluster. If σ is too large, two neighboring clusters may be overlapped with each other. However, if σ is too small, there will be a lot of outliers.
- Fallen threshold λ, which is used to decide whether a trajectory belongs to a cluster. Given a trajectory T_i, if the percentage of T_i located in a cluster area C_j exceeds λ, we assign the trajectory T_i to the cluster C_j.

The corresponding pseudo-code is depicted in Algorithm 2. It first initializes k cluster centers $\mathcal{C} = \{C_j\}_{j=1}^{k}$ using k POIs selected on the map (line 1). Then, the radius of every cluster $C_j.r$ is set according to the minimum distance among all the cluster centers based on σ (lines 2-4). For each trajectory T_i, the algorithm traverses all cluster centers C_j $(1 \leq j \leq k)$ to calculate the percentage T_i.fallenRate trajectory points that fall into the coverage of C_j. If T_i.fallenRate exceeds the threshold λ, T_i is assigned to C_j (lines 5-11). Finally, we can get a ground truth dataset with cluster labels $\mathcal{T}'$ (line 12).

VII. Experiments

In this section, we experimentally evaluate the performance, scalability, deep clustering process, and stability of E^{2}DTC on real-world trajectory datasets. We first present our experimental settings, and introduce a detailed evaluation metrics and training parameters. Then, we explore the clustering performance of E^{2}DTC, including scalability performance, the learning process and the loss function. Last but not the least, we conduct experiments for robustness analysis in terms of k selection and imbalanced data distribution.

A. Experimental Settings

Datasets. We verify the performance of E^{2}DTC model based on three real-world trajectory datasets, where the detailed dataset descriptions are as below.

TABLE II
Statics of Generated Ground-truth Datasets

Attributes	GeoLife	Porto	Hangzhou
Trajectories	85,987	86,113	80,016
Trajectory Points	1,587,320	3,320,622	5,371,406
Number of clusters	12	15	7

- **GeoLife**[1] keeps the GPS records of each user during a period of more than three years. The GPS information is collected periodically, and 91% of the trajectories are sampled every 5 seconds.
- **Porto**[2] contains 1.7 million taxi trips with a 15-seconds sampling rate over eight months in Porto, Portugal.
- **Hangzhou** is generated by taxis in Hangzhou, China. Each trajectory represents the trace of a taxi during three months with a sampling rate of every 5 seconds.

Ground-truth Preparation. As no cluster labels exist in these original datasets, we need to generate ground-truth datasets for measuring the effectiveness of clustering processing. The generation details can refer to Section VI. Here, we set the radius ratio σ to 0.6 and the fallen threshold λ to 0.7. Statistics of the generated ground-truth datasets used for evaluation are summarized in Table II.

Compared methods. We compare our proposed E^{2}DTC model with both classic and neural-network based approaches. For classic methods used for comparison, we choose K-Medoids clustering methods by considering different distance metrics (i.e.., EDR + KM, LCSS + KM, DTW + KM, and Hausdorff + KM), since they are widely used in spatial data analysis. T2vec [16] is the state-of-the-art trajectory similarity metric by using deep learning technologies, based on which k-means is used. Hence, t2vec + k-means is a neural-network based method for comparison. Overall, we compare E^{2}DTC with five existing trajectory clustering approaches.

B. Evaluation Metrics and Training Parameters

Evaluation metrics. We utilize three widely used popular evaluation metrics in unsupervised clustering [20] when exploring the performance of E^{2}DTC and other comparison methods. The evaluation metrics are the unsupervised clustering accuracy (*UACC*), the normalized mutual information (*NMI*), and the rand index (*RI*).

(i) ***UACC*** measures the difference between clustering results and ground-truth, as defined bellow:

$$UACC = \max_{m} \frac{\sum_{i=1}^{n} \mathbf{1}\left\{C_i' = m\left(C_i\right)\right\}}{n} \tag{15}$$

where C_i is the clustering assignment for T_i, C_i' is the ground truth label of T_i, and m() transforms C_i to its ground-truth label by the Hungarian algorithm [24].

(ii) ***NMI*** denotes the information shared between the predicted cluster C and ground truth C', as defined below:

$$NMI(C, C') = \frac{I(C, C')}{\sqrt{H(C)H(C')}} \tag{16}$$

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

[2] http://www.geolink.pt/ecmlpkdd2015-challenge/dataset.html

TABLE III
CLUSTERING PERFORMANCE (I.E., UACC, NMI, AND RI) OF ALL APPROACHES

Methods		**GeoLife**			**Porto**			**Hangzhou**		
		UACC	NMI	RI	UACC	NMI	RI	UACC	NMI	RI
classic K-Medoids	EDR + KM	0.582	0.678	0.402	0.645	0.698	0.474	0.732	0.715	0.672
	LCSS + KM	0.683	0.807	0.554	0.672	0.795	0.579	0.735	0.789	0.670
	DTW + KM	0.583	0.821	0.572	0.678	0.828	0.63	0.729	0.818	0.682
	Hausdorff + KM	0.667	0.796	0.633	0.743	0.883	0.747	0.688	0.735	0.629
t2vec + k-means		0.85	0.81	0.72	0.805	0.895	0.835	0.798	0.825	0.745
E^{2}DTC		**0.915**	**0.857**	**0.812**	**0.872**	**0.909**	**0.877**	**0.932**	**0.872**	**0.918**
max impro. vs. K-Medoids		33.9%	4.2%	28.2%	17.2%	2.8%	17.2%	26.6%	6.5%	34.5%
max impro. vs. t2vec + k-means		7.6%	5.9%	12.7%	8.3%	1.5%	14.6%	16.8%	5.7%	23.2%

where I is the mutual information and H is the entropy. *NMI* close to 1 indicates high quality clustering.

(iii) ***RI*** measures the clustering accuracy, i.e., the percentage of correct predictions of clusters, which is defined as below:

$$RI = \frac{TP + TN}{N(N-1)/2} \tag{17}$$

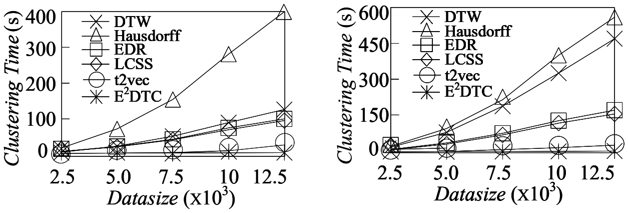
where *TP* is the number of pairs of trajectories that are correctly put in the same cluster, *TN* is the number of trajectory pairs that are correctly put in different clusters, and N is the cardinality of the dataset.

Training parameters. The default spatial grid length is set to 300 meters. The E^{2}DTC model adopts a 3 layers-based GRU [4] as the basic computational unit, because it has a better embedding performance compared with the LSTM network. In addition, we clip the gradients by enforcing a maximum gradient norm constraint [9], which is set to 5 in our experiments. Moreover, we adopt Adam stochastic gradient descent with an initial learning rate of 0.0001 and the number of iterations to 500 for training the model. To investigate how the co-efficients β and γ of clustering loss (i.e., Equation 14) affect the performance of E^{2}DTC, we conduct experiments on every dataset by sampling β and γ in the range $[10^{-3}, 10]$, and report the highest clustering accuracy. Last but not the least, EDR and LCSS based clustering methods also need a distance threshold to determine whether two trajectories are matched. Here, we utilize the grid search method to tune this distance threshold and report the best performance. Note that, for each comparison clustering method, we repeat it twenty times and report the average performance.

We implemented the framework with Python and Pytorch. All the experiments were conducted on a server with Intel Silver 4210R, 2.40GHz CPU, 16-GB RAM and a GeForce GTX-2080 Ti 11G GPU. All evaluated datasets and source codes of E^{2}DTC are publicly available here[3].

C. Performance Evaluation

In this section, we evaluate the performance of E^{2}DTC compared with both classic (i.e., K-Medoids variants) and neural-network based (i.e., t2vec + k-means) approaches on three trajectory datasets. The performance results are shown in Table III. The first observation is that, the performance of classic K-Medoids methods with the same distance function varies across different datasets. For example, Hausdorff + KM performs best on the Porto dataset but worst on the Hangzhou dataset. The reason is that different trajectory datasets may have different spatial properties, and a carefully designed clustering approach for one dataset may be not optimal for another. This observation also verified the issue analysis in Section I. The second observation is that both E^{2}DTC and t2vec-based models perform much better than classic clustering approaches, which demonstrates that deep-trajectory-representation based clustering can indeed capture the hidden information to improve the clustering quality. Last but not the least, E^{2}DTC has the best performance compared with other methods, i.e., nearly 34% improvement of *UACC* on GeoLife dataset, up to 6.5% improvement of *NMI* on Hangzhou dataset, and an average improvement of *RI* by 26% on Porto dataset. The superiority of E^{2}DTC compared with classic K-Medoids is due to the deep trajectory representations, while the superiority of E^{2}DTC compared against t2vec + k-means is mainly because of the simultaneously deep clustering mechanism. Specifically, t2vec + k-means uses a two-stage clustering, which first learns the embedded feature representations from raw trajectories and then performs k-means clustering algorithm. In the clustering tuning phase, the latent trajectory representations will not be updated for clustering tasks. In contrast, E^{2}DTC executes trajectory representation learning and clustering jointly, after which a cluster-oriented representation is developed and can be used for other trajectory clustering analysis. Overall, our E^{2}DTC model achieves the state-of-the-art clustering performance.



(a) Porto

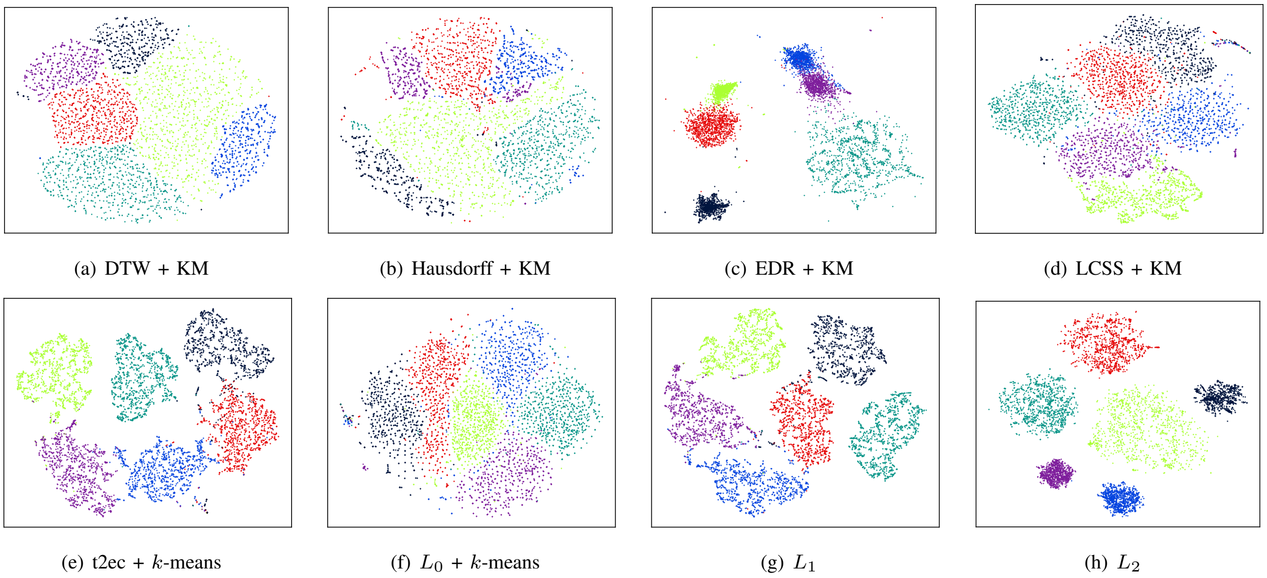
(b) Hangzhou

Fig. 3. Scalability Evaluation vs. Datasize

[3] https://github.com/Database-and-Big-Data-Analytics-Lab/E2DTC

TABLE IV
E^2DTC Performance vs. Different Loss Functions

Methods	GeoLife			Porto			Hangzhou		
	L_0	L_1	L_2	L_0	L_1	L_2	L_0	L_1	L_2
UACC	0.784	0.835	**0.915**	0.675	0.853	**0.871**	0.642	0.854	**0.932**
NMI	0.742	0.745	**0.858**	0.792	0.861	**0.908**	0.748	0.870	**0.872**
RI	0.601	0.753	**0.812**	0.621	0.818	**0.877**	0.754	0.881	**0.918**


(a) DTW + KM

(b) Hausdorff + KM

(c) EDR + KM

(d) LCSS + KM

(e) t2ec + k-means

(f) L_0 + k-means

(g) L_1

(h) L_2

Fig. 4. Visualization Comparisons of Clustering Analysis

D. Scalability Evaluation

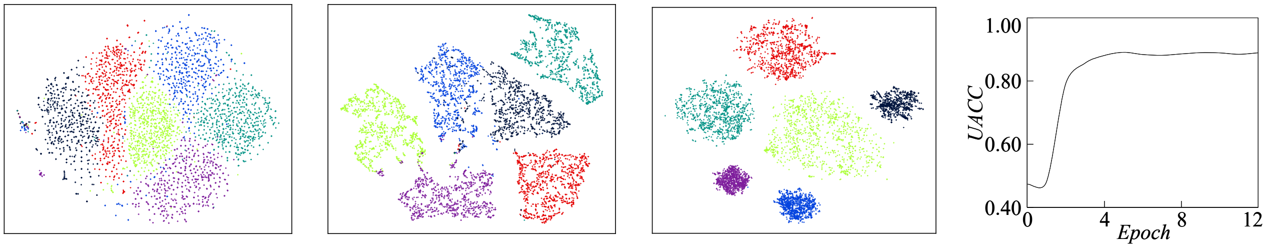
In order to verify the scalability of our E^2DTC model, we conduct the experiments by varying the cardinality of the trajectory datasets. Fig. 3 plots the corresponding results. Here, the clustering time is introduced to evaluate the performance of clustering methods, which refers to an end-to-end processing time from raw trajectory data to the obtained clustering results. For K-Medoids approaches, the clustering time denotes the total time of similarity computation and clustering processing. For our E^2DTC model, the clustering time is the total time of trajectory embedding and deep clustering. Here, we only report the results on the Porto and Hangzhou datasets, due to the space limitation and similar performance results on the GeoLife dataset. As observed, our E^2DTC method has two orders of magnitude performance improvement over the compared classic methods, and is slightly better compared with t2vec + k-means method. Moreover, the performance of both E^2DTC and t2vec + k-means is more stable with the growth of datasize, while the classic k-Medoids methods show a general sharp increment in terms of the clustering time. This is because, once the deep learning models have been trained offline, they can be efficiently utilized for trajectory clustering tasks on other datasets. However, the classic methods need to compute the distances between trajectories and perform clustering from scratch. Next, we proceed to give experimental deep insights into the E^2DTC model itself.

E. Loss Function Evaluation

To verify the effectiveness of the L_c (Equation 11) and L_t (Equation 13), we evaluate the contributions of each loss function from the view of both qualitative and quantitative. For ease of understanding, L_2 (i.e., Equation 14) denotes our E^2DTC model with full loss, L_1 (i.e., Equation 12) represents E^2DTC without triplet loss, L_0 (i.e., Equation 8) stands for E^2DTC without triplet loss and clustering loss. In other words, L_0 means that we apply classic clustering method after the pre-training phase directly.

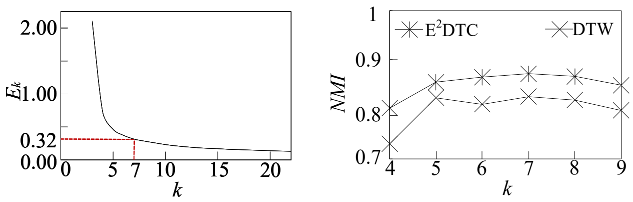
1) Qualitative: We study the effectiveness of the proposed loss functions on three trajectory datasets. Table IV depicts the performance (i.e., *UACC*, *NMI*, and *RI*) under different loss functions. The main observation is that the full loss equipped E^2DTC (i.e., L_2) performs the best, followed by L_1, and L_0 has the worst performance. It confirms the superiority of three jointly loss based deep clustering again.

2) Quantitative: To further investigate the benefits of the cluster-oriented representations with different loss functions, we also give a visualization analysis of the embedded feature space with deep-representation-based (i.e., t2vec, L_0, L_1, and L_2) and classic similarity based metrics (i.e., *DTW*, *Hausdorff*, *EDR*, and *LCSS*). Here, we adopt the t-SNE method [21] on a random subset of Hangzhou dataset with 1000 samples, as illustrated in Fig. 4. Figs. 4(a)∼ 4(d) represent raw-representation-based clustering, and Figs. 4(e)∼ 4(h) denote



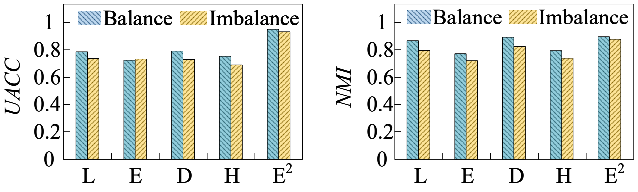
(a) Epoch 0 (b) Epoch 4 (c) Epoch 10 (d) UACC vs. epochs

Fig. 5. The Cluster-Oriented Representation Learning Process of E^{2}DTC



(a) Selection Evaluation of k (b) *NMI* vs. k

Fig. 6. Robustness Analysis vs. k



(a) Performance vs. *UACC* (b) Performance vs. *NMI*

Fig. 7. Robustness Analysis vs. *data distribution*

deep-representation-based clustering. As observed, the deep representations learned by the full E^{2}DTC (i.e., as shown in Fig. 4(h)) not only have the most separated clusters but also has the tightest relations in each cluster, compared with either classic or t2vec based clustering. In contrast, the results of $t2vec$+k-means and L_0 (i.e., Fig. 4(e) and Fig. 4(g)) exist overlapping among the clusters. For L_1 that denotes E^{2}DTC model without the triplet loss, the obtained representations are better than those obtained by L_0, but are still worse compared with the full loss L_2.

F. Learning Process of E^{2}DTC

In this subsection, we provide experimental insights into the learning process of the cluster-oriented representations of the proposed E^{2}DTC model. We also utilize the t-SNE method to visualize the iterations of learned cluster representations during training processing on the Hangzhou dataset. As shown in Figure 5, trajectory clusters are becoming significantly separated during the deep clustering process. Fig. 5(d) shows that the clustering accuracy improves correspondingly over SGD epochs. As observed, the accuracy increases rapidly in the beginning, and stays stable after Epoch 4.

TABLE V
STATICS OF DATA DISTRIBUTION

Attributes	Balanced dataset	Imbalanced dataset
Min cluster size	4,752	3,520
Max cluster size	5,424	25,088
Ave cluster size	5,055	11,430

G. Model Robustness Analysis

As we have studied the performance and processing of E^{2}DTC, we explore model stability by robustness analysis.

1) Robustness Analysis vs. k: We assumed that the number of clusters k is directly given when evaluating clustering performance. However, the actual k is not available in real-world applications, and it is necessary to determine the optimal k before clustering. We describe how we choose k as follows. We increase the number of k from 2 to 22 with step-size 1, and calculate the sum of distances from samples to their nearest centroid (denoted as E_k). The selection process on Hangzhou dataset is shown in Fig. 6(a), whose optimal k value corresponding to the elbow point is 7. After obtaining k, k-based clustering analysis can be executed. It is worth noting that the optimal k here equals to k used in our ground truth preparation, which further illustrates the credibility of our proposed ground-truth generation algorithm.

However, what if the selection of k is incorrect? To answer this question, we conduct a set of experiments on Hangzhou dataset to see whether the model performance is affected by varying k from 4 to 9. We choose *NMI* to compare clustering results with different number of clusters, which are shown in Figure 6(b). As observed, even if the chosen cluster number k is different from that in ground truth, our model can still achieve relatively high *NMI*, while traditional DTW + k-means method that achieves the highest *NMI* among other metrics always performs worse than E^{2}DTC under different k values.

2) Robustness Analysis vs. data distribution: To study the effect of data distribution on the model stability, we create two subsets from the Hangzhou dataset, where the detailed information is depicted in Table V. Figs. 7(a) and 7(b) show the performance (i.e., *UACC* and *NMI*) results, respectively. Note that, due to the space limitation, the method name is denoted using its first character. We can see that E^{2}DTC achieves the highest performance, and has a stable *UACC* and *NMI* in both balanced and imbalanced datasets. However, the traditional clustering methods cannot deal with imbalanced data distribution, and thus, the performance drops rapidly.

In addition, as real-world trajectories have more complex spatial dependencies, we also generate a variety of ground-truth datasets with different parameters σ and λ via Algorithm 2 for evaluation. The results show our algorithm achieves best performance in different ground-truth datasets, which are omitted due to the space limitation.

VIII. Conclusions

Traditional raw-representation-based trajectory clustering approaches with the inflexible clustering pipeline are not able to capture hidden spatial dependencies, and it is also difficult to choose a proper similarity functions for them. Motivated by these, we propose a new end-to-end deep trajectory clustering framework $\mathrm{E^2DTC}$ to learn deep trajectory representations and perform clustering simultaneously. Extensive experiments on three real-life datasets confirm that $\mathrm{E^2DTC}$ is an efficient and effective clustering framework with high quality results in terms of both accuracy and efficiency. $\mathrm{E^2DTC}$ consistently outperforms the baseline methods in all evaluated datasets. Moreover, we also present an effective ground-truth generation algorithm for spatial cluster labeling to encourage further researches on trajectory clustering. In the future, we plan to further speed up the deep clustering process. In addition, a context-based (e.g., road network, POIs, and semantic information) deep clustering is also attractive.

Acknowledgments. This work was supported in part by the NSFC under Grant No. 62025206 and 61972338. Lu Chen is the corresponding author of the work.

References

[1] J. Bian, D. Tian, Y. Tang, and D. Tao. A survey on trajectory clustering analysis. *arXiv preprint arXiv:1802.06971*, 2018.

[2] L. Chen, Y. Gao, Z. Fang, X. Miao, C. S. Jensen, and C. Guo. Real-time distributed co-movement pattern detection on streaming trajectories. *VLDBJ*, 12(10):1208–1220, 2019.

[3] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[4] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[5] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, pages 160–167, 2008.

[6] Z. Fang, Y. Gao, L. Pan, L. Chen, X. Miao, and C. S. Jensen. Coming: A real-time co-movement mining system for streaming trajectories. In *SIGMOD*, pages 2777–2780, 2020.

[7] T. Fernando, S. Denman, S. Sridharan, and C. Fookes. Soft+ hardwired attention: An lstm framework for human trajectory prediction and abnormal event detection. *NN*, 108:466–478, 2018.

[8] S. J. Gaffney, A. W. Robertson, P. Smyth, S. J. Camargo, and M. Ghil. Probabilistic clustering of extratropical cyclones using regression mixture models. *Climate dynamics*, 29(4):423–440, 2007.

[9] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[10] X. Guo, L. Gao, X. Liu, and J. Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI*, pages 1753–1759, 2017.

[11] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] J.-G. Lee, J. Han, X. Li, and H. Gonzalez. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. *VLDBJ*, 1(1):1081–1094, 2008.

[14] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, pages 593–604, 2007.

[15] S. Li and H. Zhao. A survey on representation learning for user modeling. In *AAAI*, pages 4997–5003, 2020.

[16] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei. Deep representation learning for trajectory similarity computation. In *ICDE*, pages 617–628, 2018.

[17] Z. Li, J. Han, M. Ji, L. Tang, Y. Yu, B. Ding, J. Lee, and R. Kays. Movemine: Mining moving object data for discovery of animal movement patterns. *TITS*, 2(4):37:1–37:32, 2011.

[18] A. Liu, Y. Zhang, X. Zhang, G. Liu, Y. Zhang, Z. Li, L. Zhao, Q. Li, and X. Zhou. Representation learning with multi-level attention for activity trajectory similarity computation. *TKDE*, 2020.

[19] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song. Sphereface: Deep hypersphere embedding for face recognition. In *CVPR*, pages 212–220, 2017.

[20] Q. Ma, J. Zheng, S. Li, and G. W. Cottrell. Learning representations for time series clustering. In *NIPS*, pages 3781–3791, 2019.

[21] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *JMLR*, 9(Nov):2579–2605, 2008.

[22] N. S. Madiraju, S. M. Sadat, D. Fisher, and H. Karimabadi. Deep temporal clustering: Fully unsupervised learning of time-domain features. *arXiv preprint arXiv:1802.01059*, 2018.

[23] J. D. Mazimpaka and S. Timpf. Trajectory data mining: A review of methods and applications. *JoSIS*, 2016(13):61–99, 2016.

[24] C. H. Papadimitriou and K. Steiglitz. Combinatorial optimization: algorithms and complexity. 1998.

[25] M. Teimouri, U. G. Indahl, H. Sickel, and H. Tveite. Deriving animal movement behaviors using movement parameters extracted from location data. *ISPRS*, 7(2):78, 2018.

[26] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, Ł. Kaiser, N. Kalchbrenner, N. Parmar, et al. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416*, 2018.

[27] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[28] D. Wang, J. Zhang, W. Cao, J. Li, and Y. Zheng. When will you arrive? estimating travel time based on deep neural networks. In *AAAI*, volume 18, pages 1–8, 2018.

[29] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu. Cosface: Large margin cosine loss for deep face recognition. In *CVPR*, pages 5265–5274, 2018.

[30] S. Wang, J. Cao, and P. Yu. Deep learning for spatio-temporal data mining: A survey. *TKDE*, 2020.

[31] X. Wang, G. Li, G. Jiang, and Z. Shi. Semantic trajectory-based event detection and event pattern mining. *KAIS*, 37(2):305–329, 2013.

[32] Y. Wang, Y. Zheng, and Y. Xue. Travel time estimation of a path using sparse trajectories. In *SIGKDD*, pages 25–34, 2014.

[33] J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML*, pages 478–487, 2016.

[34] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1998.

[35] G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, 2017.

[36] M. Yue, Y. Li, H. Yang, R. Ahuja, Y.-Y. Chiang, and C. Shahabi. Detect: Deep trajectory clustering for mobility-behavior analysis. In *ICBD*, pages 988–997, 2019.

[37] X. Zhang, F. Meng, and J. Xu. Perfinsight: A robust clustering-based abnormal behavior detection system for large-scale cloud. In *CLOUD*, pages 896–899, 2018.

[38] Y. Zhang, A. Liu, G. Liu, Z. Li, and Q. Li. Deep representation learning of activity trajectory similarity computation. In *ICWS*, pages 312–319, 2019.

[39] Z. Zhang, J. Geiger, J. Pohjalainen, A. E.-D. Mousa, W. Jin, and B. Schuller. Deep learning for environmentally robust speech recognition: An overview of recent developments. *TIST*, 9(5):1–28, 2018.

[40] Y. Zheng. Computing with spatial trajectories. pages 243–276. Springer, 2011.

[41] Y. Zheng. Trajectory data mining: An overview. *TITS*, 6(3):29:1–29:41, 2015.

[42] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: Concepts, methodologies, and applications. *TIST*, 5(3):38:1–38:55, 2014.