Detecting Loaded Trajectories for Hazardous Chemicals Transportation

Shuncheng Liu[1], Zhi Xu[1], Huimin Ren[2,3], Tianfu He[2,3], Boyang Han[2,3], Jie Bao[2,3,*], Kai Zheng[1,*], Yu Zheng[2,3]
[1]University of Electronic Science and Technology of China, Chengdu, China
[2]JD iCity, JD Technology, Beijing, China [3]JD Intelligent Cities Research, Beijing, China
{liushuncheng, zhixu023}@std.uestc.edu.cn, {renhuimin5, hetianfu3, hanboyang, baojie}@jd.com,
zhengkai@uestc.edu.cn, msyuzheng@outlook.com

***Abstract*—Hazardous chemicals transportation (HCT) brings significant financial, environmental, and health-related risks. It is imperative that a robust regulatory system is in place to reduce the risk of accidents occurring while such hazardous chemicals are being transported. Governments around the world use GPS sensors to monitor the raw trajectories of HCT trucks, but they have difficulty detecting the loaded trajectories, which is of utmost importance for the management of HCT processes. The loaded trajectory refers to the subtrajectory generated by tracking an HCT truck when it is loaded with hazardous chemical in an HCT process. The stay points in the raw trajectory provide some feasibility to detect the loaded trajectory as they reflect the potential loading and unloading actions of the HCT truck. However, directly using the stay points to detect the loaded trajectory usually leads to unsatisfactory results due to two challenges: (1) complex staying scenarios, and (2) numerous loading and unloading locations. To tackle the challenges, we propose a <u>L</u>oad<u>E</u>d tr<u>A</u>jectory <u>D</u>etection framework, called *LEAD*, to detect the loaded trajectory from the raw HCT trajectory accurately and efficiently. *LEAD* processes a raw trajectory into a set of candidate trajectories, encodes each candidate trajectory into a latent representation, and detects the loaded trajectory using the latent representations of candidate trajectories. Extensive experiments based on a real-world dataset from Nantong, China confirm the effectiveness of our framework. The results show that the detection accuracy of *LEAD* exceeds 83% which outperforms competing baselines by over 42%.**

***Index Terms*—Hazardous chemicals transportation, Loaded trajectories detection**

I. Introduction

Hazardous chemicals are chemical materials that may do harm to human health and/or environment, or are capable of damaging properties, such as flammable petroleum, corrosive acids, toxic carbon monoxide, etc. Hazardous chemicals are widely used in the workplace as raw materials, solvents, catalysts, and for a number of other functions; however, the improper operation of hazardous chemicals could cause severe accidents. For example, on June 13^{th}, 2020, a speeding fuel truck crashed and exploded in Wenling, China, causing 20 deaths, 175 injuries, and \$14.5 million loss in property damage [1]. As a result, the production, transportation, and storage processes of hazardous chemicals are strictly regulated by governments around the world [2]. Among them, Hazardous Chemicals Transportation (*HCT*) is the most uncontrollable process. According to the statistics of the Ministry of Emergency Management of China, 77% of the accidents related to hazardous chemicals have occurred during the HCT processes [3]. Due to the chemical instability, hazardous chemicals are allowed to be transported using only the HCT trucks. An HCT process has three ordered phases (shown in Figure 1): **(I)** the HCT truck goes to the loading location; **(II)** the HCT truck transports hazardous chemical from the loading location to the unloading location; and **(III)** the HCT truck leaves the unloading location. Generally, most of the HCT process can finish within a day [4], so each HCT process can be represented by a trajectory of the HCT truck within one day (namely raw trajectory), where each spatiotemporal point indicates a physical presence of the HCT truck in a location at a certain time. Due to the uncontrollability of the HCT process, governments install GPS sensors on all HCT trucks to monitor the raw trajectories.

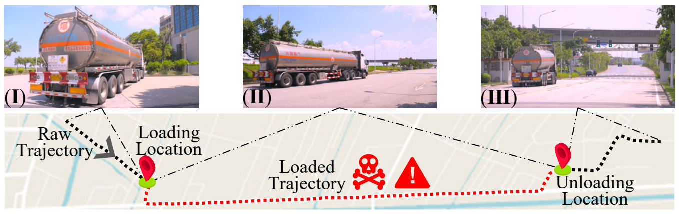


Fig. 1. Example of HCT process

In the raw trajectory, there is an especially important subtrajectory called loaded trajectory that indicates when an HCT truck is loaded with hazardous chemical (corresponding to the phase **(II)**). We find that the loaded trajectory is of utmost importance for the HCT process due to the following reasons: (1) The origin and destination of the loaded trajectory are important, as they represent the loading location and unloading location, respectively. The loading and unloading locations correspond to specific types of POIs (points of interests) such as chemical factories or fueling stations. Governments can utilize these information to promptly identify illegal loading and unloading locations [4]. (2) The complete loaded trajectory is important, as it can determine whether the driver has complied with the regulations during the HCT process. For example, the HCT truck loaded with hazardous chemical is prohibited from entering the main urban areas or moving on roads from 2:00 am to 5:00 am [5]. Once an HCT truck

* Corresponding authors: Kai Zheng and Jie Bao

is found to violate the regulations, further actions can be taken immediately. (3) The governments may improve the urban planning schemes by analyzing the loaded trajectories. For example, the driver often chooses detour routes to avoid entering the main urban areas, but these detour behaviors will affect the efficiency of the HCT process. Better understanding the route preferences of HCT trucks can improve the road network planning and urban planning of the city.

In practice, to obtain the loaded trajectory from the raw trajectory, governments require the driver to fill in the waybill which contains the time and location information of loading and unloading. Using the waybill to select a subtrajectory of the raw trajectory that matches the time and location information, the loaded trajectory could be determined. Unfortunately, the waybill is filled and submitted manually by the driver in the online system after the whole HCT process finishes. As a result, the collected waybills are usually of low quality in terms of both time information and location information: 1) the driver frequently uses the default time preset by the system for convenience (e.g., the loading time is 8:00 am and unloading time is 5:00 pm). This leads to inaccuracy in time information; 2) the loading and unloading addresses manually filled in by the driver are either coarse-grained or even incorrect. For example, the correct address is 'Zhongtian' chemical factory in Nantong, China, but the driver only fills in Nantong, or mistypes 'Zhongzhi' chemical factory. This leads to deviations in location information. Therefore, simply relying on manually filled waybills cannot get the accurate loaded trajectories, making it difficult for governments to assure the safety and legality of the HCT process.

In this work, *our objective is to accurately detect the loaded trajectory from the raw trajectory of the HCT truck*. With the loaded trajectories accurately detected, the governments can better prevent chemical accidents, regulate the drivers, and improve urban planning. In addition, high-quality waybill can be automatically generated from the loaded trajectory, which not only obtains reliable loading and unloading information, but also greatly eases the burden on drivers. Therefore, loaded trajectory detection is crucial for managing the HCT process.

In fact, the loaded trajectory has two important characteristics that can guide the detection. Firstly, when the HCT truck is loading/unloading hazardous chemical, it must stay somewhere for a sufficient period of time [4]. In other words, there are staying behaviors at the origin and destination of the loaded trajectory (defined as loading and unloading stay points). Secondly, there are specific types of POIs near the loading and unloading stay points, such as chemical factories, hospitals, etc. Therefore, analyzing the stay points in the raw trajectory is beneficial for detecting the loaded trajectory.

Intuitively, we can detect the loaded trajectory by estimating whether each stay point in the raw trajectory is a loading stay point or an unloading stay point. For instance, a model can be built to classify each stay point by recognizing the loading and unloading actions; or we can collect a white list with real loading and unloading locations utilizing the historical loaded trajectories, which can be used to search for the loading and unloading stay points. However, they cannot effectively detect the loaded trajectory due to the following reasons: (1) The scenarios where an HCT truck stays are complex. An HCT truck stays at a fuel station either because it is loading/unloading fuel, or simply because the driver is having a break while refueling the truck. These different scenarios share the same staying behavior, which are hard to be distinguished solely based on the stay points. (2) There are many locations for loading and unloading stay points, as they may appear in different chemical factories, hospitals, and even construction sites. It is difficult to collect all real loading and unloading locations for the white list that covers all potential loading and unloading stay points.

To tackle the challenges, we propose to detect the loaded trajectory by generating and identifying the candidate trajectories (i.e., a subtrajectory that starts with one stay point and ends with another stay point), based on the following insight. *A stay point indicates a staying behavior, and a trajectory that connects two consecutive stay points represents a moving behavior.* It is necessary to consider both of them to detect the loaded trajectory. The basic idea is to firstly extract all the stay points, and treat each ordered pair of stay points and all locations in-between as a candidate trajectory. Then we detect the loaded trajectory by identifying all candidate trajectories of the raw trajectory. In this way, the above two challenges could be remedied because: (1) Candidate trajectories contain not only the staying behaviors, but also the moving behaviors. The moving behaviors contain enriched information (e.g., speeds and routes), which can be leveraged to better classify complex staying scenarios. For example, after loading the fuel, the speed of HCT truck is lower than that of having a break in the fueling station. (2) We can build a deep learning-based framework to capture the general knowledge from historical loaded trajectories, and detect the loaded trajectory from an unseen raw trajectory instead of relying on pre-collected loading and unloading locations in the white list.

In this work, we propose a LoadEd trAjectory Detection framework, called *LEAD*, to accurately detect the loaded trajectory from the raw HCT trajectory. *LEAD* consists of three components: *1) raw trajectory processing*, which transforms a raw trajectory into a series of the candidate trajectories; *2) candidate trajectory encoding*, which encodes each candidate trajectory into a latent representation; and *3) loaded trajectory detection*, which detects the loaded trajectory using the latent representations of candidate trajectories. Our main contributions can be summarized as follows:

- To the best of our knowledge, it is the first work to propose and address the loaded trajectory detection problem, which can help better manage and monitor the HCT process.
- We propose *LEAD*, which can model both staying and moving behaviors of the HCT truck, and accurately detect the loaded trajectory from candidate trajectories.
- We conduct extensive experiments using a real-world dataset from Nantong, China, to evaluate the effectiveness of *LEAD*. The results show that the detection accuracy of *LEAD* exceeds 83% which outperforms competing baselines by at least 42%.

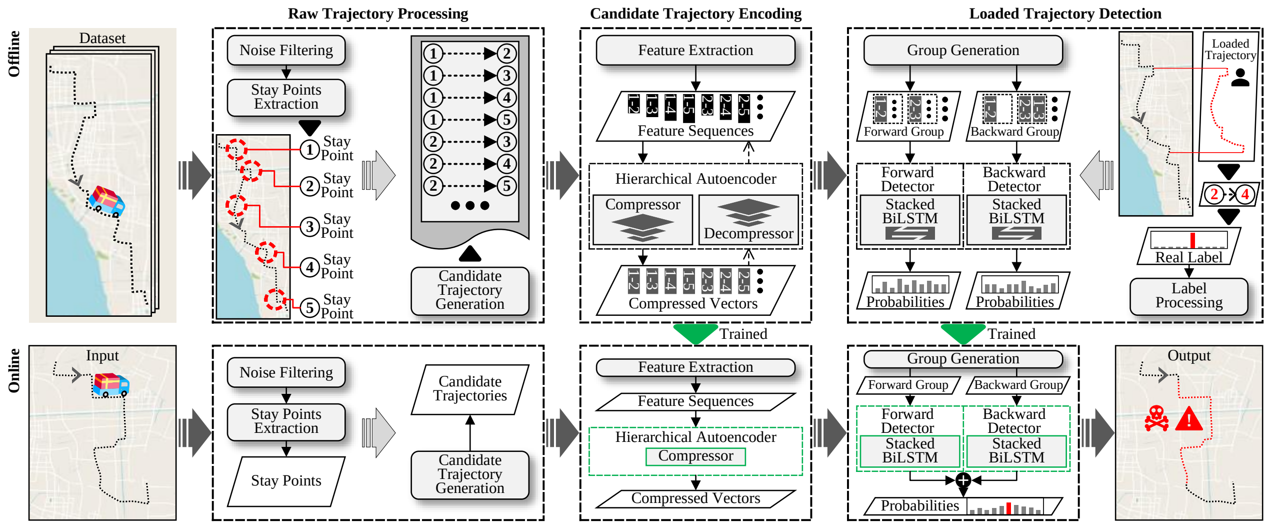


Fig. 2. *LEAD* Framework Overview

II. Overview

A. Preliminary Concepts

Definition 1 (Raw Trajectory): A raw trajectory is a sequence of GPS points, denoted as $tr^r = \langle p_1, p_2, \dots, p_n \rangle$, where each GPS point p consists of a location in latitude lat and longitude lng, and a timestamp t, i.e., $p = (lat, lng, t)$. GPS points in a raw trajectory are organized chronologically, i.e., $p_i.t < p_{i+1}.t (\forall i \in [1, n))$. In this work, a raw trajectory is generated by an HCT truck within one day, which indicates three ordered phases: *going* to a loading location $\rightarrow$ *transporting* hazardous chemical $\rightarrow$ *leaving* an unloading location.

Definition 2 (Stay Point): A stay point sp is a subtrajectory of tr^r which semantically means that an HCT truck stays in a geographic region for a while. Formally, given a distance threshold D_{max} and a time threshold T_{min}, $\langle p_i, p_{i+1}, \dots, p_j \rangle$ is a stay point sp if $distance(p_i, p_k) \leq D_{max} (\forall k \in [i+1, j])$, $distance(p_i, p_{j+1}) \geq D_{max} (j < n)$, and $|p_j.t - p_i.t| \geq T_{min}$.

Definition 3 (Loaded Trajectory): A loaded trajectory tr^l is a subtrajectory of tr^r indicating that an HCT truck loaded with hazardous chemical during the transporting phase. We note that an HCT truck still loaded with hazardous chemical and stays in a region for a while when loading and unloading hazardous chemical. Therefore, a loaded trajectory tr^l starts with the loading stay point sp_l and ends with the unloading stay point sp_u. Formally, given a loaded trajectory $\langle p_i, p_{i+1}, \dots, p_j \rangle$, the loading stay point sp_l refers to $\langle p_i, p_{i+1}, \dots, p_a \rangle$, and the unloading stay point sp_u is $\langle p_b, \dots, p_{j-1}, p_j \rangle$ $(i < a < b < j)$. Thus, a loaded trajectory $\langle p_i, p_{i+1}, \dots, p_j \rangle$ can be simplified as the ordered pair of loading and unloading stay points $\langle sp_l \dashrightarrow sp_u \rangle$.

Definition 4 (Candidate Trajectory): A candidate trajectory tr^c is a subtrajectory of tr^r that starts with one stay point and ends with another stay point. Given a candidate trajectory $\langle p_i, p_{i+1}, \dots, p_j \rangle$ that starts with $sp_{i'}$ and ends with $sp_{j'}$, it can be simplified as an ordered pair of stay points $tr^c = \langle sp_{i'} \dashrightarrow sp_{j'} \rangle$. We denote Tr^c as the set of all candidate trajectories. Given a raw trajectory tr^r, the loaded trajectory is one of the candidate trajectories, i.e., $tr^l \in Tr^c$.

Problem Statement. *Given an unseen raw trajectory tr^r, our objective is to detect the loaded trajectory tr^l from tr^r.*

B. Framework Overview

Figure 2 shows the architecture of our framework *LEAD*, which consists of three components: *raw trajectory processing*, *candidate trajectory encoding*, and *loaded trajectory detection*. *LEAD* is a two-stage framework, including the offline stage and the online stage. In the offline stage, *LEAD* learns the knowledge utilizing the historical raw trajectories with corresponding loaded trajectories. In the online phase, *LEAD* detects the loaded trajectory from the unseen raw trajectory. Next, we will briefly introduce each component.

Raw Trajectory Processing. This component takes the raw trajectory and performs three main tasks. The first task is noise filtering, which removes the outlier GPS points. The second task is the stay points extraction, which captures all stay points in the raw trajectory. The third task is the candidate trajectory generation, which produces a series of candidate trajectories by enumerating all the stay point pairs (detailed in Section III).

Candidate Trajectory Encoding. This component encodes candidate trajectories into compressed vectors, which is designed to obtain the latent representations of candidate trajectories. The component firstly extracts features from candidate trajectories and converts them to feature sequences. Then a hierarchical autoencoder learns to refine and restore the feature sequences equipped with a compressor and a decompressor. After training the hierarchical autoencoder, the compressor can be used to acquire the compressed vectors of candidate trajectories (detailed in Section IV).

Loaded Trajectory Detection. This component utilizes the compressed vectors to detect the loaded trajectory, which is designed to capture the potential relationships between different candidate trajectories, thus making detection more accurate. The component firstly organizes the compressed vectors in two ways to generate a forward group and a backward group respectively. Both groups consist of multiple subgroups, and each subgroup contains the compressed vectors of candidate trajectories with potential relationships. Secondly, the two groups are fed into forward and backward detectors, respectively. The two detectors output two discrete probability distributions, and each distribution represents probabilities of the candidate trajectories. Thirdly, the real labels derived from archived loaded trajectories are processed, and the processed labels are used to train the two detectors. After training, the outputs of two detectors are combined into one probability distribution, and the candidate trajectory which has the highest probability is the detection result (detailed in Section V).

III. Raw Trajectory Processing

This component takes a raw trajectory as input, cleans the raw trajectory and extracts the stay points from it. Finally, the candidate trajectories are generated by enumerating all the ordered pairs of stay points. Figure 3 shows an example of raw trajectory processing.

Noise Filtering. The raw trajectory generated by an HCT truck usually contains a few noise GPS points due to the shifts introduced by the GPS sensor. As shown in Figure 3(a), the error of p_{19} and p_{22} might be several hundred meters away from their true locations. Such noise GPS points would affect the performance of the subsequent tasks, e.g., stay point extraction. Thus, we utilize a heuristic approach [6] to filter noise GPS points. The approach sequentially calculates the traveling speed for each GPS point based on its precursor and itself. If the speed is larger than a speed threshold V_{max}, the current examined GPS point is removed from the raw trajectory. In Figure 3(a), p_{19} and p_{22} will be removed.

Stay Point Extraction. Acquiring all stay points in the raw trajectory can help construct candidate trajectories. We employ a rule-based algorithm [7] to extract the stay points in the cleaned raw trajectory. The algorithm firstly checks if the distance between an anchor point and its successors in a raw trajectory is larger than a distance threshold D_{max}. As shown in Figure 3(b), p_{11} is the current anchor point, and p_{12} to p_{14} are its successors within D_{max}. It then calculates the time interval between the anchor point and the last successor within D_{max} (p_{11} and p_{14}). If the duration is larger than a temporal threshold T_{min}, a stay point is extracted (from p_{11} to p_{14}), and the anchor point moves to the next GPS point after the current stay point (p_{15}). Otherwise, the anchor point moves forward by one (p_{12}). This process is repeated until the anchor point moves to the end of the raw trajectory. The algorithm can generate stay points that are temporally consecutive, which is convenient for stay points numbering.

Candidate Trajectory Generation. Based on the stay points of the raw trajectory, we can further generate a series of candidate trajectories. The candidate trajectories cover the search space of loaded trajectory detection, since each candidate trajectory starts with a stay point and ends with another stay point. To generate the candidate trajectories, we enumerate all the ordered pairs of stay points. As shown in Figure 3(c), 10 candidate trajectories are generated by traversing 5 stay points.

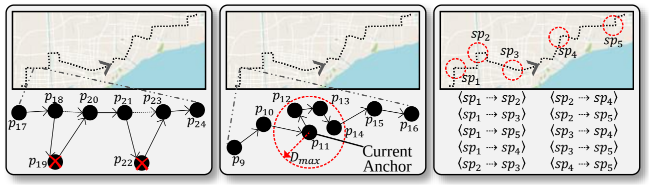

p_{17}
p_{18}
p_{19}
p_{20}
p_{21}
p_{22}
p_{23}
p_{24}
p_9
p_{10}
p_{11}
p_{12}
p_{13}
p_{14}
p_{15}
p_{16}
D_{max}
Current Anchor
sp_1
sp_2
sp_3
sp_4
sp_5
$\langle sp_1 \dashrightarrow sp_2 \rangle$
$\langle sp_1 \dashrightarrow sp_3 \rangle$
$\langle sp_1 \dashrightarrow sp_5 \rangle$
$\langle sp_1 \dashrightarrow sp_4 \rangle$
$\langle sp_2 \dashrightarrow sp_3 \rangle$
$\langle sp_2 \dashrightarrow sp_4 \rangle$
$\langle sp_2 \dashrightarrow sp_5 \rangle$
$\langle sp_3 \dashrightarrow sp_4 \rangle$
$\langle sp_3 \dashrightarrow sp_5 \rangle$
$\langle sp_4 \dashrightarrow sp_5 \rangle$
(a) Noise Filtering (b) Stay Point Extraction (c) Candidate Trajectory Generation

Fig. 3. Example of Raw Trajectory Processing

Generally, given n stay points, we can generate $n(n-1)/2$ candidate trajectories. According to the statistics, the number of stay points extracted from a raw trajectory within one day ranges from $3 \sim 14$, so the number of generated candidate trajectories is moderate ($3 \sim 91$).

IV. Candidate Trajectory Encoding

This component firstly converts candidate trajectories into high-dimensional feature sequences. Then a hierarchical autoencoder is proposed to compress them and acquire latent representations for each candidate trajectory.

A. Feature Extraction

A candidate trajectory is composed of GPS points, we need to extract the features of each GPS point. In addition to the acquired spatiotemporal features i.e., (lat, lon, t), the POI (point of interest) features reflect the spatial semantics that are beneficial for the detection. For example, if there are many factories near a GPS point, it means that the HCT truck has entered the industrial zone and might be loading/unloading hazardous chemical. Therefore, we extract both spatiotemporal and POI features for each GPS point in a candidate trajectory.

Specifically, given a candidate trajectory $\langle p_i, p_{i+1}, \ldots, p_j \rangle$, we vectorize each GPS point p as a feature vector $f = [p.lat, p.lng, p.t, poi]$, where $(p.lat, p.lng, p.t)$ is the spatiotemporal features and poi is the POI feature. For the spatiotemporal features, $p.lat$ and $p.lng$ form a spatial location, and $p.t$ is a timestamp. For the POI feature, we count the nearby POI categories of a GPS point within a radius of $100m$, forming a vector poi where each value refers to the number of a POI category existence. In this work, we select 29 typical POI categories, so f is a 32-dimensional feature vector. Moreover, to avoid the outlier issue, we normalize the above features using the Z-score strategy [8]. Finally, a candidate trajectory $\langle p_i, p_{i+1}, \ldots, p_j \rangle$ is converted to a sequence of feature vectors $\langle f_i, f_{i+1}, \ldots, f_j \rangle$, namely, a feature sequence.

B. Hierarchical Autoencoder

After extracting the features of all candidate trajectories, we get a series of feature sequences. The most straightforward approach is to use a recurrent neural network to learn the latent representation of each feature sequence and then make the detection. However, the high-dimensional feature sequence will suffer from the curse of dimensionality especially for the long-range trajectory. Moreover, the vectors in the feature sequence are sparse due to the usage of the POI feature. The sparse inputs will affect the convergence performance of the model, and even reduce the accuracy of the detection. In

TABLE I
SUMMARY OF ABBREVIATIONS IN HIERARCHICAL AUTOENCODER

Abbreviation	**Explanation**
f-seq	feature sequence
sp(mp)-f-seq	feature sequence of a stay point (move point)
SPs(MPs)-f-seq	feature sequence of stay points (move points)
sp(mp)-c-vec	compressed vector of a stay point (move point)
SPs(MPs)-c-vec-seq	sequence of compressed vectors of stay points (move points)
SPs(MPs)-c-vec	compressed vector of stay points(move points)
c-vec	compressed vector

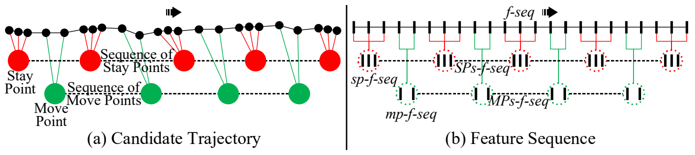


Fig. 4. Example of Candidate Trajectory and Feature Sequence

summary, a representation model that can compress the feature sequence into a low-dimensional dense vector is desired. Table I lists the abbreviations used throughout this subsection.

We introduce an autoencoder equipped with a compressor and a decompressor, to solve the aforementioned issues and learn the representation of each *f-seq*. The compressor can reduce the dimension of the *f-seq*. In contrast, the decompressor recovers the compressed vector to the *f-seq*. Furthermore, we analyze two potential characteristics of the candidate trajectories that inspire our autoencoder construction:

(1) *Spatiotemporal difference between stay points and move points.* A stay points indicates a staying behavior of an HCT truck, while the GPS points (called move point) that connect two consecutive stay points indicate a moving behavior of an HCT truck, as shown in Figure 4(a). Formally, a move point is defined as follows:

Definition 5 (Move Point): A move point $mp_{i'}$ is a subtrajectory of tr_r that connects two consecutive stay points[1], i.e., $sp_{i'}$ and $sp_{i'+1}$.

Accordingly, a candidate trajectory can be further regarded as a sequence that stay points and move points appear by turns, e.g., $tr_c = \langle sp_{i'}, mp_{i'}, sp_{i'+1}, \ldots, mp_{j'-1}, sp_{j'} \rangle$. Apparently, the spatiotemporal patterns of stay points and move points are different due to the different driving behaviors. Therefore, as shown in Figure 4(b), a feature sequence of a stay point (*sp-f-seq*) and a feature sequence of a move point (*mp-f-seq*) need to be compressed and decompressed separately, avoiding the wrong parameters sharing in the autoencoder.

(2) *Spatiotemporal difference between sequence hierarchies.* As shown in Figure 4(a), all stay points in a candidate trajectory can be regarded as a sequence of stay points, and each stay point is a sequence of GPS points. Apparently, different sequence hierarchies have different spatiotemporal patterns, e.g., the spatial/temporal spans between stay points are larger than the spatial/temporal spans between GPS points. Therefore, as in Figure 4(b), a *f-seq* can be split into a feature sequence of stay points (*SPs-f-seq*) and a feature sequence of move points (*MPs-f-seq*), where *SPs-f-seq* is a sequence of *sp-f-seq*(s) and *MPs-f-seq* is a sequence of *mp-f-seq*(s).

[1]We note two special move points: mp_0 is a move point before the first stay point sp_1, and mp_n is a move point after the last stay point sp_n.

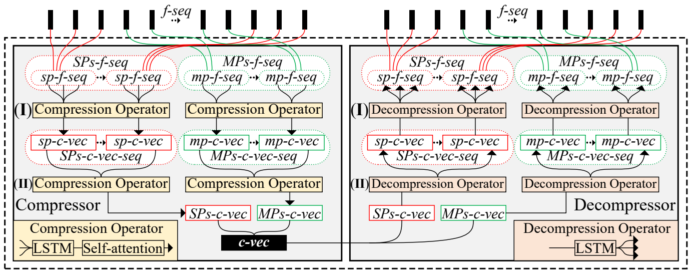


Fig. 5. Hierarchical Autoencoder

Each *sp-f-seq* or *mp-f-seq* consists of feature vectors of GPS points. The compressor should capture the hierarchical features from the *f-seq*, while the decompressor needs to recognize the hierarchical features from the compressed result.

To this end, we propose a hierarchical autoencoder, where the compressor and decompressor can separately process the stay points and move points in a hierarchical manner. Next, we introduce the compressor and decompressor, and then present the workflow in detail.

Compressor. As shown on the left side in Figure 5, our compressor has two phases. In the first phase, a compression operator compress each *sp-f-seq* (in *SPs-f-seq*) into a vector called *sp-c-vec*, and another operator compress each *mp-f-seq* (in *MPs-f-seq*) into a vector called a *mp-c-vec*. In the second phase, all *sp-c-vec*(s) and *mp-c-vec*(s) are compressed into a final compressed vector (*c-vec*) using two compression operators. A compression operator consists of an LSTM and a self-attention mechanism. The LSTM learns the latent representation of a sequence [9] and the self-attention mechanism [10], [11] aggregates a sequence into a vector.

We take a *f-seq* as an example to introduce the process of the compressor. The compressor firstly takes the *f-seq* $\langle f_i, f_{i+1}, \ldots, f_j \rangle$ of a candidate trajectory $tr_c = \langle p_i, p_{i+1}, \ldots, p_j \rangle$ (i.e., $\langle sp_{i'} \dashrightarrow sp_{j'} \rangle$, $\langle sp_{i'}, mp_{i'}, sp_{i'+1}, \ldots, mp_{j'-1}, sp_{j'} \rangle$) as an input, and divides it into *SP-f-seq* and *MP-f-seq* as follows:

$$\begin{aligned} \boldsymbol{f_{SP}} &= \langle \boldsymbol{f}_{sp_{i'}}, \boldsymbol{f}_{sp_{i'+1}}, \ldots, \boldsymbol{f}_{sp_{j'}} \rangle, \\ \boldsymbol{f_{MP}} &= \langle \boldsymbol{f}_{mp_{i'}}, \boldsymbol{f}_{mp_{i'+1}}, \ldots, \boldsymbol{f}_{mp_{j'-1}} \rangle, \end{aligned} \tag{1}$$

where $\boldsymbol{f}_{sp_{i'}}$ and $\boldsymbol{f}_{mp_{i'}}$ denote a *sp-f-seq* and *mp-f-seq*, respectively.

(I) In the first phase, two compression operators work for compressing each *sp-f-seq* and *mp-f-seq*, respectively. A *sp-f-seq*, $\boldsymbol{f}_{sp_{i'}} = \langle f_i, f_{i+1}, \ldots, f_{i+a} \rangle$ (with $a+1$ steps), is fed into an LSTM which outputs the hidden state vector at each step as follows:

$$h_\tau = LSTM(f_\tau, h_{\tau-1}; W_{l1}) \tag{2}$$

where $\tau \in [i, i+a]$, $h_{\tau-1}$ is the hidden state vector at the last step, and W_{l1} denotes the learnable parameters. Then a self-attention mechanism is used to aggregate hidden states along with the steps while different steps have different importance scores. Unlike the simple usage of the LSTM's hidden state vectors, we introduce a self-attention mechanism [10], [11] to enhance the memory ability of the operator, which can better deal with the long-range sequence. The last hidden state vector h_{i+a} of LSTM that contains the information of all the historical steps is used to calculate the importance score of each step. For example, to get the importance score of a step in the sequence, we calculate how much h_{i+a} pays attention to it. This attention indicates the weight assigned to this step during the aggregation. Following the standard procedure [10], we can obtain a query vector q of the last hidden state, and a key matrix K of all the hidden states, as follows:

$$q = h_{i+a} \times W_q + b_q, K = H \times W_K + b_K \tag{3}$$

where H refers to all hidden state vectors of the LSTM, i.e., $H = [h_i, h_{i+1}, \ldots, h_{i+a}]$, and W_q and W_K are the weights of the fully connected layers for h_{i+a} and H, respectively, b_q and b_K denote biases of W_q and W_K, respectively. It should be noted that we want to aggregate all the hidden state vectors, so the value matrix includes the hidden states output by LSTM. With q and K, we can calculate the importance scores $\boldsymbol{s}$ of all steps using $Softmax(\frac{q \times K}{\sqrt{d_k}})$, where d_k is dimension of the key vector in K. Thereafter, we can aggregate the hidden state vectors H into a vector $h = \sum_{\tau=i}^{i+a} s_i \cdot h_i$, where s_i ($s_i \in \boldsymbol{s}$) represents the importance score at each step. Finally, a *sp-c-vec* (i.e., $v^c_{sp_{i'}}$) can be obtained by using non-linear activation as follows:

$$v^c_{sp_{i'}} = Tanh((h \times W_{c1} + b_{c1}) \times W_{c2} + b_{c2}) \tag{4}$$

where W_{c1} and W_{c2} are the weights of two fully connected layers, and b_{c1} and b_{c2} denote biases of W_{c1} and W_{c2}, respectively.

Following the aforementioned process, every *sp-f-seq* in *SP-f-seq* is compressed into a *sp-c-vec*, thus forming a sequence as $\boldsymbol{v^c_{SP}} = \langle v^c_{sp_{i'}}, v^c_{sp_{i'+1}}, \ldots, v^c_{sp_{j'}} \rangle$ (called *SP-c-vec-seq*), and every *mp-f-seq* in *MP-f-seq* is compressed into a *mp-c-vec*, forming a sequence as $\boldsymbol{v^c_{MP}} = \langle v^c_{mp_{i'}}, v^c_{mp_{i'+1}}, \ldots, v^c_{mp_{j'}} \rangle$ (i.e., *MP-c-vec-seq*).

(II) Then in the second phase, the other two compression operators work for compressing *SP-c-vec-seq* and *MP-c-vec-seq*, respectively. The architecture of compression operators in the second phase is the same as those in the first phase. Therefore, $\boldsymbol{v^c_{SP}}$ is compressed into a vector v^c_{SP} (*SP-c-vec*), and $\boldsymbol{v^c_{MP}}$ is compressed into a vector v^c_{MP} (*MP-c-vec*). Finally, the *c-vec* of candidate trajectory $\langle sp_{i'} \dashrightarrow sp_{j'} \rangle$ (denoted as $v^c_{(i',j')}$) can be obtained by concatenating *SP-c-vec* and *MP-c-vec*, i.e., $v^c_{(i',j')} = [v^c_{SP}, v^c_{MP}]$.

Decompressor. As shown on the right side in Figure 5, our decompressor is roughly symmetrical to the compressor that has two phases. In the first phase, the *SP-c-vec* and *MP-c-vec* in the *c-vec* are decompressed to *SP-c-vec-seq* and *MP-c-vec-seq*, respectively. In the second phase, each *sp-c-vec* in *SP-c-vec-seq* is decompressed to a *sp-f-seq*, and each *mp-c-vec* in *MP-c-vec-seq* is decompressed to a *mp-f-seq*. Thereafter, all *sp-f-seq*(s) form a *SP-f-seq*, and all *mp-f-seq*(s) form a *MP-f-seq*. The decompressed result *f-seq* will be obtained by assembling the *SP-f-seq* and *MP-f-seq*. Similar to the compressor, there are 4 decompression operators inside the decompressor. A decompression operator is an LSTM that can utilize an input vector to recover a sequence with variable steps.

To be specific, the decompressor firstly takes $v^c_{(i',j')}$ as input, and divides it into v^c_{SP} and v^c_{MP}.

(I) In the first phase, two decompression operators work for decompressing v^c_{SP} and v^c_{MP}, respectively. For the *SP-c-vec* decompression, v^c_{SP} is fed into an LSTM, which outputs the hidden state vector at each step as follows:

$$h'_\tau = LSTM(v^c_{SP}, h'_{\tau-1}; W_{l2}) \tag{5}$$

where $\tau \in [i, i+a]$, $h^{de}_{\tau-1}$ is the hidden state vector at last step, and W_{l2} denotes the parameters of LSTM. This calculation repeats $a+1$ times to get a matrix $H' = [h'_i, h'_{i+1}, \ldots, h'_{i+a}]$. Finally, the *SP-c-vec-seq* is generated as follows:

$$\boldsymbol{v^{dec}_{SP}} = Tanh((H' \times W_{d1} + b_{d1}) \times W_{d2} + b_{d2}) \tag{6}$$

where $\boldsymbol{v^{dec}_{SP}}$ denotes the decompressed result of *SP-c-vec-seq*, W_{d1} and W_{d2} are the weights of two fully connected layers, and b_{d1} and b_{d2} denote biases of W_{d1} and W_{d2}, respectively. This non-linear activation can map the H' to between -1 to 1, matching the range of *f-seq*. Similarly, for decompressing the *MP-c-vec*, v^c_{MP} is decompressed to $\boldsymbol{v^{dec}_{MP}}$ that is the the decompressed result of *MP-c-vec-seq*.

(II) Then in the second phase, the other two operators work for decompressing each *sp-c-vec* in *SP-c-vec-seq*, and each *mp-c-vec* in *MP-c-vec-seq*, respectively. The decompression operators in the second phase are the same as those in the first phase. As a result, each $v^{dec}_{sp} \in \boldsymbol{v^{dec}_{SP}}$ is decompressed to a *sp-f-seq* (i.e., $\boldsymbol{f}^{dec}_{sp}$), and each $v^{dec}_{mp} \in \boldsymbol{v^{dec}_{MP}}$ is decompressed to a *mp-f-seq* (i.e., $\boldsymbol{f}^{dec}_{mp}$). Arranging all *sp-f-seq*(s) and *mp-f-seq*(s) can form the decompressed results of *SP-f-seq* and *MP-f-seq*, respectively, as follows:

$$\begin{aligned} \boldsymbol{f^{dec}_{SP}} &= \langle \boldsymbol{f}^{dec}_{sp_{i'}}, \boldsymbol{f}^{dec}_{sp_{i'+1}}, \ldots, \boldsymbol{f}^{dec}_{sp_{j'}} \rangle, \\ \boldsymbol{f^{dec}_{MP}} &= \langle \boldsymbol{f}^{dec}_{mp_{i'}}, \boldsymbol{f}^{dec}_{mp_{i'+1}}, \ldots, \boldsymbol{f}^{dec}_{mp_{j'-1}} \rangle, \end{aligned} \tag{7}$$

Finally, the decompressed *f-seq* i.e., $\langle f^{dec}_i, f^{dec}_{i+1}, \ldots, f^{dec}_j \rangle$, can be obtained by assembling the *SP-f-seq* and *MP-f-seq*.

Workflow. In the offline stage, given an archive of historical raw trajectories, the hierarchical autoencoder utilizes the *f-seq*(s) of all the candidate trajectories generated from the raw trajectories to train the compressor and decompressor in a self-supervised manner. In particular, all *f-seq*(s) derived from historical raw trajectories are shuffled for training the hierarchical autoencoder. For each *f-seq* e.g., $\langle f_i, f_{i+1}, \ldots, f_j \rangle$, the hierarchical autoencoder needs to minimize the MSE loss as follows:

$$L_{MSE} = \frac{1}{j-i+1} \sum_{\tau=i}^{j} (f_\tau - f^{dec}_\tau)^2 \tag{8}$$

where f^{dec}_τ denotes τ-th feature vector in the decompressed *f-seq*.

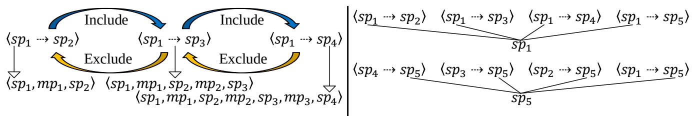
After training the hierarchical autoencoder, the decompressor is put aside, and the trained compressor is used to compress any *f-seq* of candidate trajectory into a *c-vec*. Furthermore, we note that the trained compressor can be used in both offline and online stages. For the offline stage, the *c-vec*(s) of a historical raw trajectory are fed into the detection component, thus training the detector. For the online stage, the *c-vec*(s) of an unseen raw trajectory are the inputs of the trained detection component, thus detecting the loaded trajectory.

V. Loaded Trajectory Detection

This component firstly organizes the compressed vectors in forward and backward ways and generates two groups. Then the two groups are input to two detectors, respectively. The two detectors output two discrete probability distributions in terms of forward and backward perspectives. Each distribution represents the probability that a candidate trajectory is the loaded trajectory. Furthermore, to effectively train the detectors, a label processing method is used to generate the training labels from real labels.

A. Group Generation

After encoding the candidate trajectories of a raw trajectory, we obtain a set of compressed vectors. Specifically, each $tr_r \in Tr_c$ ($tr_c = \langle sp_{i'} \dashrightarrow sp_{j'} \rangle$), is represented as a compressed vector $v^c_{(i',j')}$, via the candidate trajectory encoding. Intuitively, we can detect the loaded trajectory using a binary classifier, as a detector. The classifier utilizes all compressed vectors with corresponding labels (i.e., is loaded trajectory or not) to learn how to distinguish a loaded trajectory. Regardless of the offline or online stage, the classifier treats each candidate trajectory as an independent sample and calculates the probability using the compressed vector. However, this naive approach *ignores the relationships between candidate trajectories*, which will suffer from sub-optimal solutions. Next, we analyse three relationships between candidate trajectories, based on which we present our group generation method.


Include
Include
⟨sp₁ ⇢ sp₂⟩
⟨sp₁ ⇢ sp₃⟩
⟨sp₁ ⇢ sp₄⟩
Exclude
Exclude
⟨sp₁, mp₁, sp₂⟩
⟨sp₁, mp₁, sp₂, mp₂, sp₃⟩
⟨sp₁, mp₁, sp₂, mp₂, sp₃, mp₃, sp₄⟩
⟨sp₁ ⇢ sp₂⟩ ⟨sp₁ ⇢ sp₃⟩ ⟨sp₁ ⇢ sp₄⟩ ⟨sp₁ ⇢ sp₅⟩
sp₁
⟨sp₄ ⇢ sp₅⟩ ⟨sp₃ ⇢ sp₅⟩ ⟨sp₂ ⇢ sp₅⟩ ⟨sp₁ ⇢ sp₅⟩
sp₅

(a) Inclusion and Exclusion Relationships (b) Analogy Relationships

Fig. 6. Example of Relationships Between Candidate Trajectories

The candidate trajectories of a raw trajectory have visible *inclusion and exclusion* relationships. As show in Figure 6(a), a candidate trajectory $\langle sp_1 \dashrightarrow sp_3 \rangle$ ($\langle sp_1, mp_1, sp_2, mp_2, sp_3 \rangle$) includes another candidate trajectory $\langle sp_1 \dashrightarrow sp_2 \rangle$. Meanwhile, $\langle sp_1 \dashrightarrow sp_3 \rangle$ is a result of $\langle sp_1 \dashrightarrow sp_4 \rangle$ excluding mp_3 and sp_4. Therefore, when judging $\langle sp_1 \dashrightarrow sp_3 \rangle$, both $\langle sp_1 \dashrightarrow sp_2 \rangle$ and $\langle sp_1 \dashrightarrow sp_4 \rangle$ as a context information can guide the judgment. If the probabilities of $\langle sp_1 \dashrightarrow sp_2 \rangle$ and $\langle sp_1 \dashrightarrow sp_4 \rangle$ are very low, $\langle sp_1 \dashrightarrow sp_3 \rangle$ is almost impossible to be the loaded trajectory. In contrast, individually judging $\langle sp_1 \dashrightarrow sp_3 \rangle$ could easily make a wrong decision since it cannot be corrected by any context information.

In addition, it is necessary to consider the *analogy* relationships between candidate trajectories. Specifically, analogous candidate trajectories that share the same starting or ending stay points indicate the consistency of origin and destination. As shown in Figure 6(b), the candidate trajectories on the top are started with sp_1, and the candidate trajectories on the bottom are ended with sp_5. If sp_1 is not the loading stay point, each of the probability in the left group should be low. Similarly, if sp_5 is not the unloading stay point, each of the probability in the right group should be low. Therefore, the analogy relationships can help determine the loaded trajectory that starts with the loading stay point and ends with the unloading stay point.

TABLE II
Example of Group Generation

Compressed Vectors of Candidate Trajectories
$v^c_{(1,2)}, v^c_{(1,3)}, v^c_{(1,4)}, v^c_{(1,5)}, v^c_{(2,3)}, v^c_{(2,4)}, v^c_{(2,5)}, v^c_{(3,4)}, v^c_{(3,5)}, v^c_{(4,5)}$

Forward Group			
g_1	g_2	g_3	g_4
$\langle v^c_{(1,2)}, v^c_{(1,3)}, v^c_{(1,4)}, v^c_{(1,5)} \rangle$	$\langle v^c_{(2,3)}, v^c_{(2,4)}, v^c_{(2,5)} \rangle$	$\langle v^c_{(3,4)}, v^c_{(3,5)} \rangle$	$\langle v^c_{(4,5)} \rangle$

Backward Group			
$\overline{g}_2$	$\overline{g}_3$	$\overline{g}_4$	$\overline{g}_5$
$\langle v^c_{(1,2)} \rangle$	$\langle v^c_{(2,3)}, v^c_{(1,3)} \rangle$	$\langle v^c_{(3,4)}, v^c_{(2,4)}, v^c_{(1,4)} \rangle$	$\langle v^c_{(4,5)}, v^c_{(3,5)}, v^c_{(2,5)}, v^c_{(1,5)} \rangle$

Based on the analysis, it is necessary for a detector to capture the inclusion, exclusion and analogy relationships between candidate trajectories. However, the compressed vectors of candidate trajectories are disordered, which are impossible for a detector to directly capture the relationships. To raise awareness about the relationships between candidate trajectories, we propose a group generation method, which can wisely organize the compressed vectors using two grouping strategies, i.e., *forward grouping* and *backward grouping*.

Forward grouping is a strategy that groups the compressed vectors by the starting indexes of compressed vectors, and generates a forward group. The forward group consists of subgroups, and each subgroup contains compressed vectors with the same starting indexes. Specifically, compressed vectors with the same starting index i' form a subgroup $g_{i'}$ where compressed vectors are sorted in ascending order by ending indexes, i.e., $g_{i'} = \langle v^c_{(i',i'+1)}, v^c_{(i',i'+2)}, \ldots, v^c_{(i',n)} \rangle$, where n is the number of stay points and $1 \leq i' < n$.

Backward grouping is a strategy that groups the compressed vectors by the ending indexes, and generates a backward group. The backward group consists of subgroups, and each subgroup contains compressed vectors with the same ending indexes. Specifically, compressed vectors with the same ending index j' is a subgroup $\overline{g}_{j'}$ where compressed vectors are sorted in descending order by starting indexes, i.e., $\overline{g}_{i'} = \langle v^c_{(j'-1,j')}, v^c_{(j'-2,j')}, \ldots, v^c_{(1,j')} \rangle$ $(1 < j' \leq n)$.

For example, Table II gives 10 compressed vectors of candidate trajectories, and all compressed vectors are organized as a forward group and a backward group. There is an inclusion relationship between a compressed vector and its previous one, and there is an exclusion relationship between a compressed vector and its next one. Furthermore, in the forward group, each subgroup represents analogous candidate trajectories with the same starting stay point. And in the backward group, each subgroup indicates analogous candidate trajectories with the same ending stay point. Accordingly, the group generation method can organize the disordered compressed vectors into two groups with valuable relationships.

B. Forward and Backward Detectors

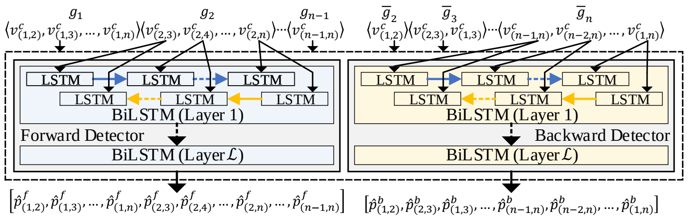
Based on the forward and backward groups, we need to construct a detector that not only captures the inclusion, exclusion and analogy relationships, but also calculates the probabilities of compressed vectors of candidate trajectories. Benefiting from the group generation, we can leverage two Bidirectional LSTM-based detectors to capture the relationships while calculating the probabilities. For the inclusion and exclusion relationships, if we regard each subgroup as a horizontal sequence (in Table II), the inclusion relationships are converted to the left-to-right relationships, and the exclusion relationships are converted to the right-to-left relationships. This interesting property inspires us to use a Bidirectional LSTM (BiLSTM) [12] to model the inclusion and exclusion relationships. For the analogy relationships, the forward and backward groups are input to two detectors sharing the same structure, and each subgroup will be separately calculated. In this way, each detector will focus on one type of analogy relationship (forward or backward), and different subgroups with different starting or ending indexes are independent. After that, two detectors output two discrete probability distributions w.r.t. forward and backward groups, and each probability corresponds to a compressed vector of candidate trajectory. In addition, we use a stacked BiLSTM network as a detector [13], which can better extract the sequential features at different timescale [14]. Next, we introduce two detectors, and then present the workflow in detail.

Forward Detector. As shown on the left side in Figure 7, the forward detector is a stacked BiLSTM network with $\mathcal{L}$ BiLSTM layers. It takes a forward group as input, and calculates the compressed vectors of each subgroup in a sequential manner. To be specifically, a subgroup $g_{i'} = \langle v^c_{(i',i'+1)}, v^c_{(i',i'+2)}, \ldots, v^c_{(i',n)} \rangle$ ($i' \in [1,n)$) is fed into the first BiLSTM layer equipped with a forward LSTM and a backward LSTM. The forward LSTM output sequence $\overrightarrow{\boldsymbol{h}}_{i'}$, is iteratively calculated using inputs of $g_{i'}$ from left to right, while the backward LSTM output sequence, $\overleftarrow{\boldsymbol{h}}_{i'}$, is obtained using the reversed inputs of $g_{i'}$ from right to left. Both the forward and backward LSTM outputs are calculated based on the standard LSTM equation, i.e., Equation 2. Then a hidden sequence $\boldsymbol{h}_{i'}$ is calculated by concatenating $\overrightarrow{\boldsymbol{h}}_{i'}$ and $\overleftarrow{\boldsymbol{h}}_{i'}$ as follows:

$$\boldsymbol{h}_{i'} = [\overrightarrow{\boldsymbol{h}}_{i'}, \overleftarrow{\boldsymbol{h}}_{i'}] \times W_{f1} + b_{f1} \tag{9}$$

where W_{f1} is the weights of fully connected layer, and b_{f1} denotes the biases of W_{f1}. To facilitate the calculation of subsequent BiLSTM layers, the length of $\boldsymbol{h}_{i'}$ is equals to those of $\overrightarrow{\boldsymbol{h}}_{i'}$ and $\overleftarrow{\boldsymbol{h}}_{i'}$. Formally, $\boldsymbol{h}_{i'} = \langle h_{(i',i'+1)}, h_{(i',i'+2)}, \ldots, h_{(i',n)} \rangle$, where $h_{(i',i'+1)}$ is the hidden vector corresponding to $v^c_{(i',i'+1)}$, and n is the number of stay points.

After obtaining the all the hidden sequences of subgroups from the first BiLSTM layer, the hidden sequences are recursively computed from the second BiLSTM layer to $\mathcal{L}$-th BiLSTM layer. Once the top-layer hidden sequences are



g_1
g_2
g_{n-1}
$\langle v^c_{(1,2)}, v^c_{(1,3)}, \ldots, v^c_{(1,n)} \rangle \langle v^c_{(2,3)}, v^c_{(2,4)}, \ldots, v^c_{(2,n)} \rangle \cdots \langle v^c_{(n-1,n)} \rangle$
$\overline{g}_2$
$\overline{g}_3$
$\overline{g}_n$
$\langle v^c_{(1,2)} \rangle \langle v^c_{(2,3)}, v^c_{(1,3)} \rangle \cdots \langle v^c_{(n-1,n)}, v^c_{(n-2,n)}, \ldots, v^c_{(1,n)} \rangle$
LSTM
BiLSTM (Layer 1)
Forward Detector
BiLSTM (Layer $\mathcal{L}$)
Backward Detector
$[\hat{p}^f_{(1,2)}, \hat{p}^f_{(1,3)}, \ldots, \hat{p}^f_{(1,n)}, \hat{p}^f_{(2,3)}, \hat{p}^f_{(2,4)}, \ldots, \hat{p}^f_{(2,n)}, \ldots, \hat{p}^f_{(n-1,n)}]$
$[\hat{p}^b_{(1,2)}, \hat{p}^b_{(2,3)}, \hat{p}^b_{(1,3)}, \ldots, \hat{p}^b_{(n-1,n)}, \hat{p}^b_{(n-2,n)}, \ldots, \hat{p}^b_{(1,n)}]$

Fig. 7. Forward and Backward Detectors

computed, the probability vector of each subgroup can be obtained as follows:

$$\hat{\boldsymbol{p}}^f_{i'} = Softmax(\boldsymbol{h}^{\mathcal{L}}_{i'} \times W_{f2} + b_{f2}) \tag{10}$$

where $\hat{\boldsymbol{p}}^f_{i'}$ is a probability vector of $g_{i'}$ subgroup ($\hat{\boldsymbol{p}}^f_{i'} = [\hat{p}^f_{(i',i'+1)}, \hat{p}^f_{(i',i'+2)}, \ldots, \hat{p}^f_{(i',n)}]$), $\boldsymbol{h}^{\mathcal{L}}_{i'}$ is the hidden sequence of $g_{i'}$ in $\mathcal{L}$-th layer, and W_{f2} is the weights of a fully connected layer and b_{f2} denotes biases of W_{f2}. Finally, the output of the forward detector is obtained by concatenating all the probability vector (from $\hat{\boldsymbol{p}}^f_1$ to $\hat{\boldsymbol{p}}^f_{n-1}$) as $\hat{\boldsymbol{\mathcal{P}}}^f = [\hat{p}^f_{(1,2)}, \hat{p}^f_{(1,3)}, \ldots, \hat{p}^f_{(n-1,n)}]$.

Backward Detector. As shown on the right side in Figure 7, the backward detector is a stacked BiLSTM network with $\mathcal{L}$ BiLSTM layers, which is the same as the forward detector. It takes the backward group as input, and calculates the probabilities of compressed vectors in each subgroup. Following the calculations of the forward detector, we can obtain the probability vector of each subgroup in the backward group, as $\hat{\boldsymbol{p}}^b_{j'} = [\hat{p}^b_{(j'-1,j')}, \hat{p}^b_{(j'-2,j')}, \ldots, \hat{p}^b_{(1,j')}]$ ($j' \in (1,n]$). And the output of the backward detector is obtained by flattening all the probability vector (from $\hat{\boldsymbol{p}}^b_2$ to $\hat{\boldsymbol{p}}^b_n$) as $\hat{\boldsymbol{\mathcal{P}}}^b = [\hat{p}^b_{(1,2)}, \hat{p}^b_{(2,3)}, \ldots, \hat{p}^b_{(1,n)}]$.

Workflow. In the offline stage, the forward and backward detectors are trained separately. The inputs are the forward and backward groups derived from the group generation. The labels are two discrete probability distributions derived from the label processing (detailed in the next subsection). For each forward group as input, the forward detector needs to minimize the Kullback-Leibler Divergence (KLD) loss as follows:

$$L^f_{KLD} = \sum_{p^f_{(i',j')} \in \boldsymbol{\mathcal{P}}^f, \hat{p}^f_{(i',j')} \in \hat{\boldsymbol{\mathcal{P}}}^f} p^f_{(i',j')} log\left(\frac{p^f_{(i',j')}}{\hat{p}^f_{(i',j')}}\right) \tag{11}$$

where $\boldsymbol{\mathcal{P}}^f$ is the label probability distribution from the label processing, and $\hat{\boldsymbol{\mathcal{P}}}^f$ is the probability distributions output from the forward detector. For each backward group as input, the backward detector needs to minimize the KLD loss as follows:

$$L^b_{KLD} = \sum_{p^b_{(i',j')} \in \boldsymbol{\mathcal{P}}^b, \hat{p}^b_{(i',j')} \in \hat{\boldsymbol{\mathcal{P}}}^b} p^b_{(i',j')} log\left(\frac{p^b_{(i',j')}}{\hat{p}^b_{(i',j')}}\right) \tag{12}$$

where $\boldsymbol{\mathcal{P}}^b$ is the label probability distribution and $\hat{\boldsymbol{\mathcal{P}}}^b$ is the probability distributions output from the backward detector.

Thereafter, in the online stage, the trained forward and backward detectors can be used to detect the loaded trajectory by merging the output probability distributions. Specifically, the output distributions of two detectors, i.e., $\hat{\boldsymbol{\mathcal{P}}}^f$ and $\hat{\boldsymbol{\mathcal{P}}}^b$, are firstly merged into one intermediate vector, where each element is obtained by adding the probabilities with the same index in both distributions. Then the intermediate vector is normalized by rescaling the range of all elements to $[0, 1]$, thus obtaining the merged probability distribution $\hat{\boldsymbol{\mathcal{P}}}$, and each probability indicates the likelihood that a candidate trajectory of being a loaded trajectory. Finally, the index of the maximum probability in $\hat{\boldsymbol{\mathcal{P}}}$ determines the detected loaded trajectory. For example, if $\hat{p}_{(i',j')}$ is the maximum probability in $\hat{\boldsymbol{\mathcal{P}}}$, the detected loaded trajectory is $\langle sp_{i'} \dashrightarrow sp_{j'}\rangle$. Formally, given a merged probability distribution $\hat{\boldsymbol{\mathcal{P}}}$, the detected loaded trajectory $\hat{tr}^{l}$ can be determined as follows:

$$\hat{tr}^{l} = \langle sp_{i'} \dashrightarrow sp_{j'}\rangle,\ (i', j') = \underset{(i',j')}{\arg\max}\, \hat{\boldsymbol{\mathcal{P}}}, \tag{13}$$

where (i', j') denotes the index of a probability $\hat{p}_{(i',j')} \in \hat{\boldsymbol{\mathcal{P}}}$. According to (i', j'), we can select the candidate trajectory $\langle sp_{i'} \dashrightarrow sp_{j'}\rangle$ as the detected loaded trajectory $\hat{tr}^{l}$.

C. Label Processing

To separately train the forward and backward detectors, we need to prepare two labels for each pair of forward and backward groups. Intuitively, we can acquire two discrete probability distributions (called real labels), based on the archived loaded trajectory. Each real label is like a one-hot vector where only one probability equals 1 while others are 0. Specifically, a real label of forward group is $\boldsymbol{\mathcal{P}}^{f} = [p^{f}_{(1,2)}, p^{f}_{(1,3)}, \ldots, p^{f}_{(n-1,n)}]$. There is one probability $p^{f}_{(i',j')}$=1 indicating that the candidate trajectory $\langle sp_{i'} \dashrightarrow sp_{j'}\rangle$ is the loaded trajectory, and other probabilities are 0. Similarly, a real label of backward group is $\boldsymbol{\mathcal{P}}^{b} = [p^{b}_{(1,2)}, p^{b}_{(2,3)}, \ldots, p^{b}_{(1,n)}]$, where $p^{f}_{(i',j')}$=1, and other probabilities are 0. However, the zero probabilities in real labels will cause undefined $log(0)$ in Equation (11) and (12), leading to invalid KLD losses. Therefore, we introduce a small constant ϵ, e.g., $\epsilon = 10^{-5}$, and process the real labels as follows:

$$\boldsymbol{\mathcal{P}}^{f} = [p^{f}_{(1,2)} = \epsilon, p^{f}_{(1,3)} = \epsilon, ..., p^{f}_{(i',j')} = 1 - k\epsilon, ..., p^{f}_{(n-1,n)} = \epsilon]$$

$$\boldsymbol{\mathcal{P}}^{b} = [p^{b}_{(1,2)} = \epsilon, p^{b}_{(2,3)} = \epsilon, ..., p^{b}_{(i',j')} = 1 - k\epsilon, ..., p^{b}_{(1,n)} = \epsilon]$$

where n is the number of stay points and k is the number of probabilities with ϵ in a real label. Thus, $\boldsymbol{\mathcal{P}}^{f}$ and $\boldsymbol{\mathcal{P}}^{b}$ as two discrete probability distributions can be effectively used to train the detectors.

VI. Experiment

A. Experimental Settings

Dataset. The dataset used in our experiments is collected from the city of Nantong, China. Nantong is heavily dependent on the chemicals industry, which contributes a proportion of 41.9% in the secondary industry GDP of Nantong in 2020. The dataset contains the historical raw trajectories and loaded trajectories. The raw trajectories are generated by HCT trucks within one day, and the loaded trajectories are extracted from the raw trajectories by government employees. It is easy to collect a large number of raw trajectories through GPS sensors, but it is costly to obtain the real loaded trajectory as ground truth, since employees are required to carefully determine the loaded trajectory from each raw trajectory one by one.

To the best of our ability, we collect 5,968 raw trajectories with loaded trajectories generated by 2,734 HCT trucks which are operated in Nantong, over a period of 2 months (from September 1^{th} to October 31^{th}, 2020). The GPS points of trajectories are based on WGS84 coordinate system [15], and the average sampling interval is around 2 minutes[2]. We split the dataset into training set, validation set and test set with a splitting ratio of 8:1:1. In addition, the HCT trucks of the validation set and test set do not overlap with the HCT trucks of the training set, thus making the evaluation more convincing.

Baselines. We compare our *LEAD* with several representative baselines. To the best of our knowledge, there is no existing solution that can detect the loaded trajectory for hazardous chemicals transportation. Therefore, we design the following stay point-based methods for comparison:

(**1**) SP-R: It detects the loaded trajectory via a rule-based classifier. Specifically, after the noise filtering and stay point extraction, a rule-based classifier can find all the potential loading/unloading (*l/u*) stay points, by matching each stay point with locations in the white list. The white list is generated from the training set, both ends of each loaded trajectory as two locations (i.e., loading and unloading locations) are stored in the white list. If a stay point nears a location in the white list, it is an *l/u* stay point. Otherwise, the stay point is an ordinary stay point (i.e., stay point neither loading nor unloading). For each stay point, we set the searching radius to $500m$. Next, we can determine a loading stay point and an unloading stay point from all *l/u* stay points. Based on our domain-specific knowledge, a loaded trajectory always starts with a loading stay point and ends with a unloading stay point. Thus, we consider a greedy strategy, which sets the first (earliest) *l/u* stay point as the loading stay point, and sets the last (latest) *l/u* stay point as the unloading stay point.

(**2**) SP-GRU: It detects the loaded trajectory via a GRU-based classifier. Specifically, after the noise filtering and stay point extraction, we introduce a GRU [16] with 128 hidden units as a binary classifier to classify each stay point into *l/u* stay point or ordinary stay point, where the input is the feature sequence of a stay point. Thereafter, the greedy strategy is used to determine a loading stay point and an unloading stay point from all *l/u* stay points.

(**3**) SP-LSTM: It detects the loaded trajectory via a LSTM-based classifier. The SP-LSTM is similar to SP-GRU except for the classifier. We use a LSTM with 128 hidden units as a binary classifier to classify each stay point into *l/u* stay point or ordinary stay point. Similarly, the greedy strategy is used to determine a loading stay point and an unloading stay point based on all *l/u* stay points.

[2]The detailed statistics of the dataset are not disclosed, due to our data confidential agreement.

We note that the three baselines will suffer from invalid detection results, when finding insufficient *l/u* stay points (e.g., 0 or 1 *l/u* stay point). In this case, we will set the first extracted stay point as the loading stay point and set the last one as the unloading stay point, as a default loaded trajectory.

Variants. To evaluate each component of our framework, we perform ablation studies with the following variants of *LEAD*:
(**1**) *LEAD*-NoPoi: We remove the POI feature in the feature extraction, and we use zero padding as the pseudo POI feature, keeping the dimension of the feature vector constant.
(**2**) *LEAD*-NoSel: We remove the self-attention mechanism in the hierarchical autoencoder. Instead, we directly use the last hidden state vector of each LSTM in the compressor.
(**3**) *LEAD*-NoHie: We remove the hierarchy and separation structures in the hierarchical autoencoder. There is only one compression operator and one decompression operator in the compressor and decompressor, respectively.
(**4**) *LEAD*-NoGro: We remove the group generation in the loaded trajectory detection. After deleting it, the forward and backward detectors are unavailable. We substitute the forward and backward detectors with 4 fully connected layers with Sigmoid activator to calculate the probability of each candidate trajectory. Specifically, the number of units from the first layer to the fourth layer is set to 64, 32, 32, and 1, respectively, and the Sigmoid activator is set in the fourth layer to output the probability.
(**5**) *LEAD*-NoFor: We remove the forward detector in the loaded trajectory detection, and only use the probability distribution of backward detector to detect the loaded trajectory.
(**6**) *LEAD*-NoBac: We remove the backward detector in the loaded trajectory detection, and only use the probability distribution of forward detector to detect the loaded trajectory.

Evaluation Metric. The purpose of our defined problem is to detect the loaded trajectory from a raw trajectory. Thus, we define the detection accuracy to show the performance of our framework and baselines as follows:

$$Acc = \frac{\sum_{i=1}^{N_{te}} hit_i}{N_{te}} \times 100\%,\ hit_i = \begin{cases} 1, & if\ \hat{tr}_i^l = tr_i^l, \\ 0, & otherwise, \end{cases} \tag{14}$$

where N_{te} is number of test samples, hit_i indicates that if i-th detected loaded trajectory hits the ground truth loaded trajectory, $\hat{tr}_i^l$ denotes i-th detected loaded trajectory, and tr_i^l denotes i-th ground truth loaded trajectory. The larger value of Acc indicates that the method detect the loaded trajectory more accurately.

Implementation Details. We set the hyperparameters in *LEAD* throughout the experiments as follows:
(**1**) Raw Trajectory Processing: Firstly, in noise filtering, the speed threshold V_{max} is set to $130km/h$ since the moving speed of an HCT truck rarely exceeds this threshold. Secondly, in stay point extraction, We test different parameter combinations and find that most staying behaviors (e.g., loading, unloading, resting, etc.) can be included in stay points when we set $D_{max} = 500m$ and $T_{min} = 15min$. Based on the dataset, the number of stay points extracted from a raw trajectory ranges from $3 \sim 14$, which is reasonable for the staying times of an HCT truck within one day. Finally, in candidate trajectory generation, the number of generated candidate trajectories ranges from $3 \sim 91$, deriving from the number of stay points.
(**2**) Candidate Trajectory Encoding: For the feature extraction, we collect 415,639 POIs in Nantong and categorize them into 29 typical categories, e.g., company, hospital, chemical factory, etc. The collected POIs are used to extract the POI feature of a GPS point, and the dimension of a feature vector is 32. For the hierarchical autoencoder, all compression operators in the compressor share the same architecture, and all decompression operators in the decompressor share the same architecture. To keep the hierarchical autoencoder within a tractable size, the number of hidden units in each LSTM and fully connected layer are the same, and we set the number of hidden units in the hierarchical autoencoder as 32. Thus, the compressed vector of any feature sequence has a fixed dimension of 64.
(**3**) Loaded Trajectory Detection: For the forward and backward detectors, all LSTMs have 64 hidden units, and the fully connected layers for calculating the probabilities have 1 unit. In addition, we tune the number of BiLSTM layers $\mathcal{L}$ from 1 to 10 and find the highest detection accuracy when $\mathcal{L} = 4$ on the validation set, thus we set $\mathcal{L} = 4$ as the default. For the label processing, we set the small constant ϵ to 10^{-5}.

Finally, for the offline phase of the hierarchical autoencoder, and the forward and backward detectors, we use the Adam optimizer [17] for updating the parameters with a scheduled learning rate of 0.0001. The training bath size is set to 1, because the dimension of the inputs (i.e., feature sequences and forward/backward groups) are not fixed. Nevertheless, we backpropagate the average loss of $\mathcal{B}$ consecutive training samples to simulate an iteration of batch training, and we set $\mathcal{B}$ as 64, thus improving the training efficiency to a certain extent. The above hyperparameters are tuned on the validation set by using the grid search. In addition, we use Early Stopping [18] to avoid the overfitting of the neural networks. Our experimental results are reported based on the above settings, unless expressly specified.

Environment. We implement all algorithms in Python 3.7.10, and run the experiments on an Ubuntu Server with an Intel(R) CPU i7-4770@3.4GHz, and NVIDIA Tesla V100 GPU.

B. End-to-End Evaluation of LEAD

In this subsection, we study the end-to-end performance of *LEAD* by comparing it against several baselines in terms of accuracy and efficiency.

Detection Accuracy. We adopt Acc defined by Equation (14) to evaluate the detection accuracy of *LEAD*. To make the evaluation more clearly, we separately report Acc under the different numbers of stay points on the test set, in Table III. As depicted, our *LEAD* outperforms all the other detection methods for all the test cases, and the accuracy of all the methods can be ranked as: *LEAD*>>SP-LSTM>SP-GRU>SP-R. We can see that as the number of stay points increases, the accuracy of all the methods decreases, due to the increased difficulty of detection. The details are as follows:

TABLE III
ACCURACY OF BASELINES AND OURS (*LEAD*) ON THE TEST SET

Acc(%)	#Stay Points (Percentage%)				
Method	**3∼5 (22%)**	**6∼8 (34%)**	**9∼11 (25%)**	**12∼14 (19%)**	**3∼14 (100%)**
SP-R	60.2	54.2	46.8	33.3	49.7
SP-GRU	66.4	63.5	54.7	49.2	59.2
SP-LSTM	67.2	63.9	56.2	51.6	60.4
LEAD	**95.6**	**92.4**	**87.5**	**83.8**	**90.2**

(**1**) 3∼5 Stay Points: There are 22% of raw trajectories that have 3∼5 stay points in the test set, thus the number of candidate trajectories ranges from 3∼10. In this case, *LEAD* outperforms SP-R by 59%, SP-GRU by 44%, SP-LSTM by 42% on Acc.
(**2**) 6∼8 Stay Points: There are 34% of raw trajectories that have 6∼8 stay points in the test set, thus the number of candidate trajectories ranges from 15∼28. In this case, *LEAD* outperforms SP-R by 70%, SP-GRU by 46%, SP-LSTM by 45% on Acc.
(**3**) 9∼11 Stay Points: There are 25% of raw trajectories that have 9∼11 stay points in the test set, and the number of candidate trajectories ranges from 36∼55. In this case, *LEAD* outperforms SP-R by 87%, SP-GRU by 60%, SP-LSTM by 56% on Acc.
(**4**) 12∼14 stay points: There are 19% of raw trajectories that have 12∼14 stay points in the test set, and the number of candidate trajectories ranges from 66∼91. In this case, *LEAD* outperforms SP-R by 1.5×, SP-GRU by 70%, SP-LSTM by 62% on Acc.
(**5**) 3∼14 stay points: In the test set, all raw trajectories have 3∼14 stay points, and the number of candidate trajectories ranges from 3∼91. In this case, *LEAD* outperforms SP-R by 81%, SP-GRU by 52%, SP-LSTM by 49% on Acc.

It is expected that SP-R performs the worst among all the methods since the defective rule-based classifier cannot cover all the locations in the white list, and returns the default loaded trajectory frequently. For the poor performance of SP-GRU and SP-LSTM, the main reasons are three-fold. Firstly, *LEAD* generates all candidate trajectories of a raw trajectory, and detects the loaded trajectory based on their probabilities. While SP-GRU and SP-LSTM, similar to SP-R, cannot specify the loaded trajectory when classifiers return 0 or 1 l/u stay point, obtaining the inaccurate default loaded trajectory. Secondly, *LEAD* captures not only the features of stay points but also the features of move points, while SP-GRU and SP-LSTM only consider the features of stay points, lacking the moving information between stay points. Thirdly, *LEAD* captures the inclusion, exclusion and analogy relationships between candidate trajectories, which is benefit to detect the accurate loaded trajectory. SP-GRU and SP-LSTM treat each stay point as an independent sample and detect the loaded trajectory based on their probabilities.

Efficiency. We record the mean inference time of all the methods to detect the loaded trajectory on the test set, and separately report them under the different numbers of stay points in Figure 8. As we can see, *LEAD* requires around 12 ∼ 25s in all the test cases, which is much faster than

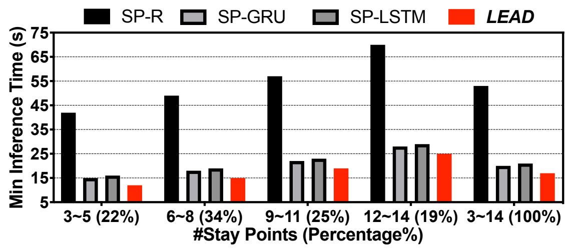


Fig. 8. Inference Time of Baselines and Ours (*LEAD*) on the Test Set

TABLE IV
ACCURACY OF *LEAD* AND *LEAD*-VARIANTS ON THE TEST SET

Acc(%)	#Stay Points (Percentage%)				
Method	**3∼5 (22%)**	**6∼8 (34%)**	**9∼11 (25%)**	**12∼14 (19%)**	**3∼14 (100%)**
LEAD-NoPoi	85.7	83.1	77.6	72.4	80.3
LEAD-NoSel	93.6	89.4	82.7	78.3	86.5
LEAD-NoHie	90.4	86.7	81.3	76.4	84.2
LEAD-NoGro	88.6	85.2	80.9	77.2	83.4
LEAD-NoFor	94.0	91.3	85.8	82.7	88.9
LEAD-NoBac	93.5	90.6	86.3	82.2	88.6
LEAD	**95.6**	**92.4**	**87.5**	**83.8**	**90.2**

other methods. Specifcally, *LEAD* outperforms SP-R by 64 ∼ 71%, SP-GRU by 11 ∼ 20%, and SP-LSTM by 14 ∼ 25% in all the test cases. SP-R performs the worst since it needs to traverse all the locations of white list when classifying a stay point. For SP-GRU and SP-LSTM, they need to classify all stay points before they return the loaded trajectory. In contrast, *LEAD* can return the loaded trajectory by making once forward computation of each component efficiently.

C. Evaluation of Candidate Trajectory Encoding

Effectiveness of Feature Extraction. In our feature extraction, we consider the spatiotemporal features and POI features of a GPS point. To evaluate the effectiveness of the additional POI features, we compare *LEAD* against *LEAD*-NoPoi, and report the Acc in Table IV. As shown, the accuracy of *LEAD* is noticeably better than *LEAD*-NoPoi. To be specific, *LEAD* outperforms *LEAD*-NoPoi by 11 ∼ 16% in all the test cases, since it considers the informative POI features, while *LEAD*-NoPoi lacks the POI features, leading to lower accuracy.

Effectiveness of Hierarchical Autoencoder. To study the effectiveness of the hierarchical autoencoder, we compare *LEAD* against two variants, including *LEAD*-NoSel and *LEAD*-NoHie, and we report their detection accuracy in Table IV. We can see that *LEAD*-NoSel has decent detection accuracy when the number of stay points is small, but it performs poorly when the number of stay points is large. Specifically, in the test cases of 3∼5 and 6∼8 stay points, the accuracy of *LEAD*-NoSel is around 90%, while in the test cases of 9∼11 and 12∼14 stay points, the accuracy of *LEAD*-NoSel decreases to 82.7% and 78.3%, respectively. This clearly shows that the self-attention mechanism is helpful for long-range features memorization. *LEAD*-NoHie performs noticeably worse than *LEAD*, specifically, *LEAD*-NoHie reduces the accuracy by 5∼ 9% compared to *LEAD*, in all the test cases. This is because *LEAD*-NoHie ignores not only the difference between stay points and move points, but also the difference between sequence hierarchies. We further record the curves of MSE loss (cf. Equation (8)) for

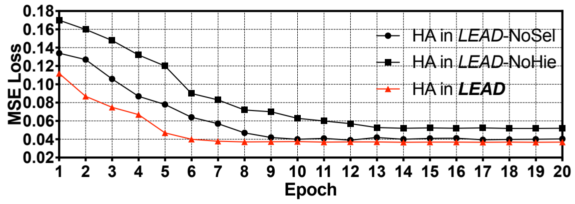


Fig. 9. Curves of MSE Loss for Several Hierarchical Autoencoders

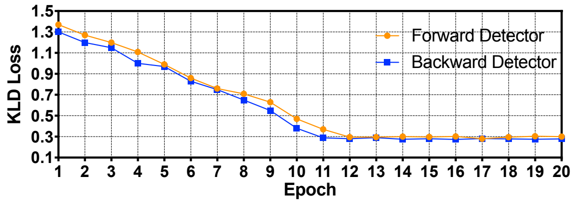


Fig. 10. Curves of KLD Loss for Forward and Backward Detectors

training the Hierarchical Autoencoder (HA) inside the *LEAD*, *LEAD*-NoSel and *LEAD*-NoHie on the training set. As shown in Figure 9, the MSE loss of the HA in *LEAD* is minimized at around 7 epoch with 0.038, the MSE loss of the HA in *LEAD*-NoSel is minimized at around 9 epochs with 0.042, and the MSE loss of the HA in *LEAD*-NoHie is minimized at around 13 epochs with 0.053. For the HA in *LEAD*-NoSel, the compression operators cannot effectively aggregate the historical features, thus increasing the training convergence epochs and the MSE loss. For the HA in *LEAD*-NoHie, the compressor cannot capture the representative information in the feature sequence, and the decompressor lacks the ability to restore the informative feature sequence, thus greatly increasing the training convergence time and the MSE loss.

D. Evaluation of Loaded Trajectory Detection

Effectiveness of Group Generation. The group generation is proposed to organize the disordered compressed vectors into forward and backward groups with inclusion, exclusion and analogy relationships. To evaluate the effectiveness of the group generation, we compare *LEAD* against *LEAD*-NoGro, and report the Acc in Table IV. As depicted, the accuracy of *LEAD* is noticeably better than *LEAD*-NoGro. Specifically, *LEAD* outperforms *LEAD*-NoGro by $8 \sim 9\times$ in all the test cases, this is because *LEAD*-NoGro treats each candidate trajectory as an independent sample and calculates the probability using the compressed vector, and ignores the relationships between candidate trajectories.

Effectiveness of Forward and Backward Detectors. The forward and backward detectors are designed to capture the relationships inside the forward and backward groups, respectively, while calculating the probability distributions. To evaluate the effectiveness of the group generation, we compare *LEAD* against two variants, including *LEAD*-NoFor and *LEAD*-NoBac, and report their detection accuracy in Table IV. We can see that both *LEAD*-NoFor and *LEAD*-NoBac have decent detection accuracy, but still worse than *LEAD*. Specifically, *LEAD* outperforms *LEAD*-NoFor by $1 \sim 2\%$, *LEAD*-NoBac by $1 \sim 2\%$ in all the test cases. The reason is that *LEAD* can fully capture the informative relationships in both forward and backward groups, and then consider the probability distributions from the forward and backward detectors, thus detecting the loading trajectory more accurately. *LEAD*-NoFor and *LEAD*-NoBac could easily obtain the sub-optimal solution due to the one-sided consideration. To further study the training effectiveness of the forward and backward detectors, we record the curves of KLD loss (cf. Equation (11)–(12)) for training the forward and backward detectors on the training set. As shown in Figure 10, the KLD loss of the forward detector is minimized at around 12 epochs with 0.296, and the KLD loss of the backward detector is minimized at around 11 epochs with 0.289. This proves that both forward and backward detectors can effectively approximate the label probability distributions and converge.

VII. Related Work

Hazardous Chemicals Transportation Problem. The problem of HCT attracts great attention in urban management since it is ubiquitous and dangerous. In academia, tremendous efforts have been devoted to dealing with HCT problem [4], [19]–[27]. Zhu et al. [4] propose an approach to find out unregistered and unqualified hazardous chemical facilities by mining HCT trajectories. Fabiano et al. [19] present a site-oriented framework for hazardous chemical facilities risk assessment. Planas et al. [21] provide a monitoring system to monitor HCT trucks based on regional responsibilities. Wang et al. [25] build a system for risky zones identification based on HCT trajectories. In this work, we propose and address the problem of loaded trajectory detection, which can help better manage and monitor the HCT process.

Urban Computing. Urban computing [28] aims to solve the issues caused by human's rapid progress in urbanization, such as bike lane planning recommendation [29], [30], crime rate inference [31], air quality prediction [32], fire risk prediction [33], crowd flow alert [34], [35], and resource rebalancing [36]. In this work, we focus on detecting the loaded trajectory from raw trajectory of the HCT truck, which is benefit for better supervising the HCT process in a city.

VIII. Conclusion

In this work, we propose a deep learning-based framework *LEAD*, to detect the loaded trajectory from the raw HCT trajectory accurately. *LEAD* firstly processes a raw trajectory into a set of candidate trajectories, and then encodes each candidate trajectory into a compressed vector. Finally, *LEAD* detects the loaded trajectory using the compressed vectors of candidate trajectories. Experiments show that the detection accuracy of *LEAD* exceeds 83% which outperforms competing baselines by at least 42%.

Acknowledgment

This work is partially supported by NSFC (No. 61972069, 61836007 and 61832017), National Key R&D Program of China (2019YFB2101801) and Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021).

References

[1] 30 officials punished over deadly tank blast in east china. [Online]. Available: http://www.xinhuanet.com/english/2021-01/02/c_139636668.htm

[2] J. Wang, C. Chen, J. Wu, and Z. Xiong, "No longer sleeping with a bomb: a duet system for protecting urban safety from dangerous goods," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1673–1681.

[3] Precautions for the transportation of hazardous chemicals. [Online]. Available: http://yjt.hubei.gov.cn/yjkp/yjwkt/202109/t20210901_3731770.shtml

[4] Z. Zhu, H. Ren, S. Ruan, B. Han, J. Bao, R. Li, Y. Li, and Y. Zheng, "Icfinder: A ubiquitous approach to detecting illegal hazardous chemical facilities with truck trajectories," in *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, 2021, pp. 37–40.

[5] Restrictions of hazardous chemicals transportation. [Online]. Available: http://zscom.zhoushan.gov.cn/art/2020/3/9/art_1228969521_42453341.html

[6] Y. Zheng, "Trajectory data mining: an overview," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, no. 3, pp. 1–41, 2015.

[7] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma, "Mining user similarity based on location history," in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, 2008, pp. 1–10.

[8] C. Cheadle, M. P. Vawter, W. J. Freed, and K. G. Becker, "Analysis of microarray data using z score transformation," *The Journal of molecular diagnostics*, vol. 5, no. 2, pp. 73–81, 2003.

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

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[11] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018.

[12] S. Zhang, D. Zheng, X. Hu, and M. Yang, "Bidirectional long short-term memory networks for relation classification," in *Proceedings of the 29th Pacific Asia conference on language, information and computation*, 2015, pp. 73–78.

[13] C. Bian, H. He, and S. Yang, "Stacked bidirectional long short-term memory networks for state-of-charge estimation of lithium-ion batteries," *Energy*, vol. 191, p. 116538, 2020.

[14] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[15] J. A. Slater and S. Malys, "Wgs 84—past, present and future," in *Advances in Positioning and Reference Frames*. Springer, 1998, pp. 1–7.

[16] 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.

[17] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv e-prints*, 2014.

[18] R. Caruana, S. Lawrence, and L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *NIPS*, 2000, p. 381–387.

[19] B. Fabiano, F. Currò, A. P. Reverberi, and R. Pastorino, "Dangerous good transportation by road: from risk analysis to emergency planning," *Journal of Loss Prevention in the process industries*, vol. 18, no. 4-6, pp. 403–413, 2005.

[20] B. Y. Kara and V. Verter, "Designing a road network for hazardous materials transportation," *Transportation Science*, vol. 38, no. 2, pp. 188–196, 2004.

[21] E. Planas, E. Pastor, F. Presutto, and J. Tixier, "Results of the mitra project: Monitoring and intervention for the transportation of dangerous goods," *Journal of hazardous materials*, vol. 152, no. 2, pp. 516–526, 2008.

[22] G. Purdy, "Risk analysis of the transportation of dangerous goods by road and rail," *Journal of Hazardous materials*, vol. 33, no. 2, pp. 229–259, 1993.

[23] M. Verma, "Railroad transportation of dangerous goods: A conditional exposure approach to minimize transport risk," *Transportation research part C: emerging technologies*, vol. 19, no. 5, pp. 790–802, 2011.

[24] M. Verma and V. Verter, "Railroad transportation of dangerous goods: Population exposure to airborne toxins," *Computers & operations research*, vol. 34, no. 5, pp. 1287–1303, 2007.

[25] J. Wang, C. Chen, J. Wu, and Z. Xiong, "No longer sleeping with a bomb: a duet system for protecting urban safety from dangerous goods," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1673–1681.

[26] D. Meng, Y. Jia, J. Du, and F. Yu, "Data-driven control for relative degree systems via iterative learning," *IEEE Transactions on Neural Networks*, vol. 22, no. 12, pp. 2213–2225, 2011.

[27] W. Li, Y. Jia, J. Du, and J. Zhang, "Distributed multiple-model estimation for simultaneous localization and tracking with nlos mitigation," *IEEE transactions on vehicular technology*, vol. 62, no. 6, pp. 2824–2830, 2013.

[28] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: concepts, methodologies, and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 5, no. 3, pp. 1–55, 2014.

[29] J. Bao, T. He, S. Ruan, Y. Li, and Y. Zheng, "Planning bike lanes based on sharing-bikes' trajectories," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 1377–1386.

[30] T. He, J. Bao, S. Ruan, R. Li, Y. Li, H. He, and Y. Zheng, "Interactive bike lane planning using sharing bikes' trajectories," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1529–1542, 2019.

[31] H. Wang, D. Kifer, C. Graif, and Z. Li, "Crime rate inference with big data," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 635–644.

[32] X. Yi, J. Zhang, Z. Wang, T. Li, and Y. Zheng, "Deep distributed fusion network for air quality prediction," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 965–973.

[33] M. Madaio, S.-T. Chen, O. L. Haimson, W. Zhang, X. Cheng, M. Hinds-Aldrich, D. H. Chau, and B. Dilkina, "Firebird: Predicting fire risk and prioritizing fire inspections in atlanta," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 185–194.

[34] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[35] C. Shi, X. Han, L. Song, X. Wang, S. Wang, J. Du, and S. Y. Philip, "Deep collaborative filtering with multi-aspect information in heterogeneous networks," *IEEE transactions on knowledge and data engineering*, vol. 33, no. 4, pp. 1413–1425, 2019.

[36] J. Liu, L. Sun, W. Chen, and H. Xiong, "Rebalancing bike sharing systems: A multi-source data smart optimization," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1005–1014.