School of Computer Science
The University of Auckland
New Zealand

Time-aware Trajectory Anomaly Detection

Wei Cheng

August 2022

Supervisors:

Kaiqi Zhao

Jiamou Liu

A thesis submitted in partial fulfillment of the requirements of Master of Science (Computer Science)

Abstract

Recent years have seen rapid development and proliferation of devices carrying GPS chips, such as mobile phones. These GPS chips continuously record the timestamps and GPS coordinates of moving objects, generating a large amount of trajectory data at an unprecedented scale. Mining and analyzing these trajectory data provides us valuable insights into many areas of public interest, including urban planning, transportation management, and tourism planning. One data mining task that attracts many research interests is trajectory anomaly detection. Particularly, for trajectory data collected from taxis, we are interested in the anomalous trajectories as they could be caused by taxi frauds or anomalous events. This task is not trivial since the normal routes in the real world can be very different from place to place. In addition, previous studies mainly focus on utilizing the spatial information of the trajectory to detect anomalies. In this research, we propose a novel time-aware framework that combines both spatial and temporal information to encode trajectories into a latent route space. We model the latent route space with a Gaussian mixture model to capture routes of different types (*e.g.*, commute during peak hours). We use the detect-by-generation strategy for detection, which means the anomaly score of a trajectory is computed by the likelihood that the trajectory follows a normal route in the latent route space. Experimental results show that our time-aware framework outperforms all baseline models, including the state-of-the-art GMVSAE model, in both offline and online anomaly detection tasks. We also use visualization techniques to show that our framework captures time-aware driving patterns successfully.

Acknowledgements

With the continued threat of the COVID-19, 2021 was not a peaceful year for most of us. Due to the lockdown starting in August, we had to cancel face-to-face meetings and stay home as much as possible. The physical isolation of people brought both physical and mental challenges to us, which we managed to overcome together.

I would like to thank my supervisors, Dr. Kaiqi Zhao and Dr. Jiamou Liu, for their supervision and professional support throughout my Master's degree.

I would also like to thank my family and friends for their accompany and mental support, which meant a lot to me.

Contents

1 Introduction **1**

1.1 Background . . . 1
1.2 Limitations of Existing Studies . . . 2
1.3 Research Aims . . . 4
1.4 Proposed Methodology . . . 5
1.5 Structure of Thesis . . . 6

2 Related Work **9**

2.1 Metric-based Approaches . . . 9
2.2 Learning-based Approaches . . . 14

3 Preliminary **19**

3.1 GRU . . . 19
3.2 Attention . . . 21
3.3 Variational Autoencoder . . . 23

4 Methodology **27**

4.1 Preprocessing . . . 28
4.2 Inference Module . . . 29
4.3 Route Space . . . 34
4.4 Generative Module . . . 36
4.5 Training Method . . . 38
4.6 Detect-by-generation Strategy . . . 39

5 Evaluation **41**

5.1 Experimental Setup . . . 42
5.2 Model Evaluation . . . 46

6 Conclusion **53**

6.1 Achievements . . . 53

6.2 Limitations 54
6.3 Future Work 54

List of Figures

3.1 Gated Recurrent Unit [25] 20
3.2 Attention Mechanism [39] 22
3.3 Encoder-Decoder Framework [39] 23
3.4 Variational Autoencoder [8] 24

4.1 Overview of the Time-aware Trajectory Anomaly Detection Framework . . 28
4.2 PGM of Inference Module 30
4.3 Linear Combination Module 30
4.4 Self-attention Module 31
4.5 Gaussian Combination Module 32
4.6 PGM of the Generative Process 37

5.1 Map of Porto 42
5.2 Example of the Generated Anomalous Trajectory 43
5.3 Example of PR Curve [13] 45
5.4 Efficiency Comparison for TAD Models 48
5.5 The Route Space of TAD-GC-20 50
5.6 Trajectories Sampled from Different Route Types 50
5.7 Heatmap of Trajectories for Different Time Periods 51

List of Tables

5.1 PR-AUC Scores for Checking Parameter Sensitivity 45
5.2 PR-AUC Scores for Offline Detection 47
5.3 PR-AUC Scores for Online Detection 49

1 Introduction

This chapter gives an overview of the existing studies in the field of trajectory anomaly detection, their limitations, and our novel time-aware framework. Section 1.1 introduces the research background and the importance of the trajectory anomaly detection task. Section 1.2 reviews the existing studies and shows their limitations and the challenges of overcoming the limitations. Section 1.3 lists the aims of our research and gives the formal definition of the research problem. Section 1.4 briefly introduces our proposed methodology and how it addresses the challenges. Section 1.5 shows the structure of this thesis.

1.1 Background

With the rapid development and proliferation of mobile phones, GPS trackers, and other devices that can continuously send their current location information over time, a tremendous amount of trajectory data (*i.e.*, sequences of GPS coordinates produced by a moving object over time) is being generated over the world at an unprecedented speed [41, 34, 4, 35, 33]. These trajectories might be generated by people who carry their phones while moving around or various vehicles such as cars, ships, and airplanes. Other things can also generate trajectories such as birds, seals, and other animals if we install GPS trackers on them. The availability of these data at a large scale provides a new opportunity for us to study and understand more about human mobility, community dy-

namics, and the migration of animals. Performing a variety of data mining tasks on these trajectory data can benefit many areas of public interests, including but not limited to public safety, urban planning, tourism planning, transportation management, ecological monitoring, and environmental protection [38, 3, 2].

Among the various data mining tasks we can perform on trajectory data, trajectory anomaly detection has been attracted more and more interest in recent years [7, 3]. There are a few reasons that people are so interested in detecting anomalous trajectories. One of these reasons is taxi fraud. For example, some taxi drivers might take unnecessary detours to overcharge people who are not familiar with the road networks in the city, especially the tourists. A research paper published by the National Bureau of Economic Research [21] reveals that taxi drivers in New York City take unnecessary long detours in about 7% rides starting from airports. If the passengers are not local people, the detours would generally be even longer. These fraudulent behaviors can cause financial losses and severe damages to the tourists' impression of the city. To stop these frauds and improve the quality of taxi services, we need to detect anomalous taxi trajectories by some automatic systems, preferably in real-time. Currently, this is handled mainly by manual inspections, which is very inefficient in terms of both time and cost.

Another reason is that anomalous trajectory detection can lead to anomalous event detection. Suppose there are many anomalous trajectories with similar driving patterns detected within a relatively short period of time. In that case, there might be some unusual events happening at that time, such as car accidents, traffic jams, or road closures due to temporary maintenance. Detecting these events would be very helpful to traffic management and urban planning. It is also worth noting that detecting and eliminating anomalies in data is usually an essential step in data preprocessing. Studying anomalous trajectory detection helps remove outliers in raw data, improve the quality of data, and benefit many other downstream tasks such as pattern mining.

1.2 Limitations of Existing Studies

The importance of trajectory anomaly detection has inspired many studies in this field. There are mainly two categories of the works, metric-based approaches that highly depend on hand-crafted features and predefined parameters, and learning-based approaches that use supervised or unsupervised machine learning methods. Most studies use metric-based approaches since it is very intuitive to compare trajectories, calculate the distance with some metrics, and classify trajectories that are far away from others as anomalies. For example, the earliest research presented by Knorr *et al.* [15] in this field calculates the Euclidean distance between trajectories in the dataset. Trajectories that are far away from a certain proportion of all trajectories are classified as anomalies. Lee *et al.* [18]

proposed a partition-and-detect framework, TRAOD, that performs two-level partitioning on trajectories and uses the Hausdorff distance as the distance metric. Wang *et al.* [32] proposed a novel DIS distance metric and a time-efficient algorithm named ATDC that can identify four types of anomalies. Zhang *et al.* [38] proposed an isolation-based framework, iBAT, that exploits the "few and different" property of anomalous trajectories and classifies trajectories that can be easily isolated from others as anomalies. This work is further adapted for online detection environment by Chen *et al.* [3]. The performance of metric-based approaches is usually sensitive to hand-crafted heuristics and predefined parameters such as the threshold of distance, and it is difficult to choose appropriate values for the parameters. To overcome the drawbacks of metric-based approaches, some researchers introduced machine learning techniques into this field. Song *et al.* [30] proposed a framework using stacked RNN layers and MLP layers to encode trajectories and directly generate anomaly scores. Gray *et al.* [10] introduced generative adversarial techniques into the field of trajectory anomaly detection and proposed IGMM-GAN, a model consisting of bidirectional GANs and an infinite Gaussian mixture model. Liu *et al.* [22] combined the Gaussian mixture model, variational autoencoder, and RNN to capture and encode sequential information of trajectories into a space of deep representations. The detection of anomalies is achieved by calculating the probability of generating the given trajectory from normal routes.

These previous researches generally achieve good performance on detecting anomalous trajectories, but all of them have a big missing part: they only focus on the spatial information of trajectories and simply ignore the temporal information. In many real-world situations, we have not only spatial trajectories but also timestamps associated with each geographical coordinate, which record the time when the object moved to a particular geographical location. This is very natural to have timestamps if we consider GPS devices that are pervasively used in the real world since every time the devices send a GPS coordinate, they send the current timestamp together. Ignoring temporal information does not come without a cost in the existing studies. The temporal information can help us detect anomalous trajectories in two ways. First, let us consider vehicles that are traveling in a big city, starting at location A and heading to location B. Due to the traffic jams during peak hours, the normal route between A and B at usual times (which is also the shortest and fastest route ignoring traffic conditions) might now become a bad choice as you might be stuck in an endless flow of vehicles and wait for hours. Taking an alternative route might be a better choice when traveling during peak times. In this example, we can see that the time during the day has a big influence on what route is considered a normal route and what route is an anomaly. Second, GPS coordinates are not necessarily sampled at a fixed rate, so the timestamps associated with coordinates implicitly provide information on the speed of the moving object. Considering this information in our model

might help us generate more accurate representations of trajectories and hence a better performance on anomaly detection.

Another important thing that is not considered by most studies is the long-term dependency within the trajectory. Most deep learning methods use the recurrent neural network (RNN) to encode trajectories, but RNN has problems discovering long-term dependencies within the input sequence. This could cause problems when discovering the normal driving patterns and predicting the next location of the trajectory because there could be multiple normal routes between two locations, and the long-term dependency is needed to understand from which route the trajectory comes. For example, suppose that there are two normal routes between the location S and D and both routes intersect at the location I. The two routes are $S \to A \to I \to B \to D$ and $S \to E \to I \to F \to D$. At the intersection I, we need to know the multi-step relations $A \to \ldots \to B$ and $E \to \ldots \to F$ to predict the next location of the vehicle. If the previous location is A, then the next location after I is more likely to be B rather than F. Without capturing the multi-step relations, an anomaly detector might incorrectly give B and F similar probabilities for being the next location.

To overcome the limitations of the previous studies, we need to handle the following challenges.

1. How do we encode and model the time?

2. Compared to spatial locations, time is a very different type of input. How do we combine spatial and temporal information?

3. How do we learn the driving patterns under different periods of time?

4. How do we solve the long-term dependency problem?

1.3 Research Aims

The objective of this work is to develop a time-aware trajectory anomaly detection framework to handle the challenges and address the limitations of existing studies. The aims of our framework are listed as follows:

1. Encode the temporal information of trajectories and generate time-aware route representations of trajectories.

2. Efficiently and effectively detect anomalous trajectories.

3. Learn time-aware driving patterns from trajectories.

Before briefly introducing our methodology, we need to formally define the research problem. We define a trajectory as a sequence of points $P = \{p_1, p_2, \ldots, p_n\}$, where n is the number of points in the sequence, and each i-th point p_i contains the current location and timestamp. The current location of a point is represented by a GPS coordinate $(p^{(i)}_{\text{latitude}}, p^{(i)}_{\text{longitude}})$, where the latitude value gives the north-south position and the longitude value gives the east-west position on the earth. The current timestamp of a point is the exact time when the GPS coordinate of the moving object is recorded. In Unix timestamp format, each timestamp is represented by a single integer.

The raw trajectory data needs to be preprocessed before feeding into our framework. More concretely, each trajectory needs to be split into two sequences so that we can deal with locations and timestamps separately. Following previous works [38, 3, 19], we draw a rectangular spatial area that contains all trajectories, split this area into many grid cells that share the same width and height, and map each trajectory to the grid. Each GPS coordinate is mapped to a grid cell, so the spatial information in a trajectory is converted from a sequence of GPS coordinates to a sequence of grid cells. The sequence of grid cells is denoted as $X = \{x_1, x_2, \ldots, x_n\}$ where each x_i is a grid cell. We process the temporal information similarly. We split the time during a day (24 hours) into 5760 timeslots, and each timeslot has a length of 15 seconds. Each timestamp is then mapped to one of the 5760 timeslots. The sequence of timestamps is $T = \{t_1, t_2, \ldots, t_n\}$ where each t_i is a timeslot.

With the *detect-by-generation* strategy, the problem of time-aware trajectory anomaly detection is defined as computing the likelihood of the trajectory consisting of the sequence of grid cells X and the sequence of timestamps T given the normal route r^*, *i.e.*, $p(X, T|r^*)$.

1.4 Proposed Methodology

Our research mainly focuses on integrating temporal information into the trajectory anomaly detection framework and discovering time-aware driving patterns. However, as Section 1.2 presents, there are four challenges that need to be addressed. To address the first and second challenges, we propose a framework containing an inference module and a generative module. The inference module accepts both the sequence of grid cells and the sequence of timestamps from the trajectory and infers the time-aware route. The generative module tries to reconstruct the input trajectory from the time-aware route. To address the third challenge, we propose to project trajectories into a time-aware route space, where each point in the space represents a time-aware route. In existing studies, the concept of routes considers spatial information only. For example, some routes are urban streets in the center of the city, and some are highways across the city. In con-

trast, the *time-aware routes* consider temporal information as well as spatial information. For example, some time-aware routes are urban street routes during peak hours, and some are the same routes during off-peak hours. These time-aware routes are encoded in different regions of the route space. To address the fourth challenge, we propose an attention-based inference module to capture the long-term dependency. We also propose two RNN-based inference modules for comparison. For detecting the trajectory anomalies, we use the detect-by-generation strategy to compute the anomaly score based on how likely the trajectory can be generated from the normal routes in the route space. We conduct experiments in both offline and online environments to show that the performance of our time-aware model is better than all baseline models. We use visualization techniques to show that our model successfully captures driving patterns under different periods of time.

The main contributions of our work are:

1. We propose a novel trajectory anomaly detection framework that takes the temporal information as an input and can find out time-aware trajectory patterns of normal routes.

2. We propose three approaches to infer time-aware routes and a generative module with the detection-by-generation strategy to detect trajectory anomalies.

3. We evaluate the proposed framework on a real-world dataset and show that it outperforms the state-of-the-art model by up to 8.1%. We also conduct case studies to show that our framework can capture time-aware routes of the given trajectories.

1.5 Structure of Thesis

This thesis is structured into the following chapters.

- Chapter 2 - Related Work: We present different approaches and algorithms in the field of trajectory anomaly detection. We also discuss their advantages and disadvantages.

- Chapter 3 - Preliminary: We introduce concepts and models that are used in our framework.

- Chapter 4 - Methodology: We propose a novel time-aware trajectory anomaly detection framework in detail.

- Chapter 5 - Evaluation: We analyze experimental results and visualize driving patterns captured by our framework.

- Chapter 6 - Conclusion: We conclude this thesis by discussing the limitations of our work and potential future works.

2

Related Work

This chapter reviews the background and current studies in anomalous trajectory detection. The importance of trajectory anomaly detection has inspired many research works in recent years. The approaches are various but can be divided into two main categories. Section 2.1 reviews approaches that are based on distance or other metrics. Section 2.2 reviews approaches that are based on supervised, semi-supervised or unsupervised learning.

2.1 Metric-based Approaches

2.1.1 Distance Metrics

Many studies use distance metrics to differentiate normal trajectories and anomalies in their proposed frameworks. In these distance-based methods, an appropriate metric is crucial and can significantly affect their performance on the anomaly detection task. This section reviews common distance metrics that are widely used in distance-based methods. The metrics we will introduce are Euclidean distance, Hausdorff distance, Longest Common Subsequence (LCSS) distance, and Dynamic Time Warping (DTW) distance.

2.1.1.1 Euclidean Distance

Euclidean distance is a simple and classical distance metric that is widely used in many areas of machine learning and data mining [37, 24]. The Euclidean distance between two trajectories is defined as:

$$D(T_i, T_j) = \frac{1}{n} \sum_{k=1}^{n} \sqrt{\sum_{m=1}^{p} (a_k^m - b_k^m)^2} \tag{2.1}$$

where T_i and T_j are two trajectories with the same length, n is the length of the trajectories, p is the dimensionality of the trajectories. In the scenarios of our concerns, p will always be 2, which means the trajectories are sitting in a two-dimensional space. The advantage of Euclidean distance is that it is simple and straightforward. However, the Euclidean distance requires the two trajectories having the same length, and each point in T_i should have a corresponding point in T_j (the timestamps of the two trajectories should match), which are not the case in the real world in the most of the times. Without any parameters to tune, the Euclidean distance is not robust to noises in datasets. Therefore, a lot of other distance metrics are proposed.

2.1.1.2 Hausdorff Distance

Hausdorff distance is a measure of dissimilarity between two sets of points. In the field of computational geometry and image processing, Hausdorff distance is usually used to recognise and match shapes [1]. The directed Hausdorff distance is

$$\vec{\delta}_H(A, B) = \max_{a \in A} \left\{ \min_{b \in B} \{d(a, b)\} \right\} \tag{2.2}$$

where A and B are two sets of points, $a \in A$, $b \in B$, $d(a, b)$ is some distance measure such as Euclidean distance. The directed Hausdorff distance is asymmetric and does not reflect a full comparison of the shapes of trajectories, so it is usually better to use the undirected Hausdorff distance $\delta_H(A, B)$ as the distance metric:

$$\delta_H(A, B) = \max \left\{ \vec{\delta}_H(A, B), \vec{\delta}_H(B, A) \right\} \tag{2.3}$$

Informally speaking, Hausdorff distance gives a low value when every point in A or B is close to some points in the other set. From the Equation 2.3, we can see that the time complexity of Hausdorff distance computation is $O(nm)$, where n is the length of A and m is the length of B. This means the computation of Hausdorff distance is very time-consuming when the data gets large. Hausdorff distance is also very sensitive to noisy points. An outlier caused by some random errors can result in a significant change

in Hausdorff distance.

2.1.1.3 LCSS Distance

Longest Common Sub-Sequence (LCSS) [11] is a distance metric that finds out the length of the longest common subsequence of two trajectories. Its formal definition is recursive:

$$\begin{aligned}&\mathrm{LCSS}\left(T_i, T_j\right)\\ &= \begin{cases} 0 & n = m = 0 \\ 1 + \mathrm{LCSS}_{\sigma,\epsilon}\left(\mathrm{Head}\left(T_i\right), \mathrm{Head}\left(T_j\right)\right) & \left|a_i^x - b_j^x\right| \leq \sigma, \text{ and } \left|a_i^y - b_j^y\right| \leq \epsilon \\ \max\left(\mathrm{LCSS}_{\sigma,\epsilon}\left(\mathrm{Head}\left(T_i\right), T_j\right), \mathrm{LCSS}_{\sigma,\epsilon}\left(T_i, \mathrm{Head}\left(T_j\right)\right)\right) & \text{others} \end{cases}\end{aligned} \tag{2.4}$$

where T_i and T_j are the two trajectories to be compared, and their lengths are denoted as n and m, respectively. σ and ϵ are the predefined thresholds along the x-axis and y-axis, respectively. As long as the x values and y values of a point in T_i and a point in T_j are close enough, *i.e.*, sitting in the range of σ and ϵ, the two points will be regarded as a pair of common points, and the length of the longest common sequence will increase by one. By introducing the predefined parameters σ and ϵ, LCSS distance allows for some deviations in the data points, but deciding the best values of the parameters is a difficult problem. The time complexity of LCSS distance computation is $O(nm)$, which is as high as Hausdorff distance.

2.1.1.4 DTW Distance

Dynamic Time Warping (DTW) [16] distance is a method that is good at matching similar trajectories by local scaling even if their lengths or sampling rates are different. The DTW distance is formally defined as follows:

$$D\left(T_i, T_j\right) = \begin{cases} 0 & m = n = 0 \\ \infty & m = 0 \| n = 0 \\ \mathrm{Dist}\left(a_i, b_j\right) + \min \begin{cases} D\left(\mathrm{Rest}\left(T_i\right), \mathrm{Rest}\left(T_j\right)\right) \\ D\left(\mathrm{Rest}\left(T_i\right), T_j\right) \\ D\left(T_i, \mathrm{Rest}\left(T_j\right)\right) \end{cases} & \text{others} \end{cases} \tag{2.5}$$

where $D(\cdot)$ means the DTW distance, T_i and T_j are two trajectories, and their lengths are m and n respectively. $\mathrm{Dist}(a_i, b_j)$ is the Euclidean distance between a point in T_i and a point in T_j, respectively a_i and b_j. $\mathrm{Rest}(\cdot)$ means the remaining part of the trajectory after removing points that are already matched. When both trajectories are empty, the DTW distance is defined as 0. When one trajectory contains points and the other is empty, the DTW distance is defined as ∞. When both trajectories are not empty, the

DTW distance is calculated by a recursion.

2.1.2 Distance-based Methods

In 2000, Knorr *et al.* [15] showed the importance of anomaly detection and proposed an efficient algorithm for detecting trajectory anomalies. This is the first work in the field of trajectory anomaly detection [24]. Trajectories are transformed into data points with features such as position, length, direction, and velocity. They first defined a distance-based outlier as $DB(p, D)$, which means at least proportion p of trajectories in the dataset are at least distance D away from the outlier. Two simple algorithms are proposed. One of them is an index-based algorithm that performs a range search for each data point, and the outliers are the data points left out after all range searches are completed. The other algorithm is a nested-loop algorithm that iteratively computes the distance between each pair of data points. This algorithm avoids building indexing structures but still has high time complexity. Based on these two simple algorithms, they proposed a time-efficient algorithm that builds a two-dimensional cell structure and uses the properties of the cell structure to avoid checking a large number of data points as they cannot be outliers. The time complexity is $O(m + N)$, where m is the number of cells and N is the number of data points. The time-efficient algorithm can be further generalized to detect anomalies in high-dimensional spaces. The default distance metric used in the algorithm is the Euclidean distance, but it can be generalized to the L_P-norm distance when the algorithm is applied on high-dimensional data points. For large datasets that can not be directly loaded into the memory, they also presented another version of the algorithm that guarantees the number of passes looping over the dataset is at most three. Since the definition of outliers is manually defined by hand-crafted parameters p and D, a challenge of applying the proposed algorithm is that it is difficult to determine suitable values for the parameters. Experimental results on three different datasets show that the proposed algorithm performs well, and outliers can be detected efficiently, even for large datasets.

In 2008, Lee *et al.* [18] proposed a partition-and-detect framework, TRAOD (TRAjectory Outlier Detection), which utilizes the line segment Hausdorff distance to measure the distance between trajectories and detect trajectory anomalies. TRAOD involves two phases. The first phase is partitioning, where each trajectory will be partitioned twice with different granularities to make sure both quality and efficiency are high. The second phase is detection, where outliers will be detected according to their relative distances. They first propose a simple algorithm that partitions each trajectory into the smallest meaningful units and then detect outliers by computing the distance between the partitions of trajectories. This algorithm needs three parameters, p, F, and D. The parameter p is the proportion of the set of trajectories that need to be at least distance D away from the outlier. The value of the parameter D is difficult to be determined as the per-

formance is highly sensitive to it. The parameter F is the threshold of noisy points that will not be classified as outliers. Then they introduced an improved version of the algorithm that is more time-efficient. Instead of partitioning into basic units, the improved version uses two-level partitioning. At the first level, each trajectory is partitioned at a coarse granularity without losing preciseness and conciseness. At the second level, only the trajectory partitions that are likely to be outliers are selected and partitioned again at a fine granularity. The time complexity depends on the granularities at the two levels of partitioning. Experiments conducted on hurricane track and animal movement datasets show that TRAOD can detect anomalous trajectories efficiently and accurately.

In 2011, Laxhammar and Falkman [17] proposed Similarity-based Nearest Neighbor Conformal Anomaly Detector (SNN-CAD), which detects anomalous trajectories based on the theory of conformal prediction and the undirected Hausdorff distance. Conformal prediction estimates levels of confidence of the predictions made by a machine learning model given previous data and predictions. It is intended for use in an online learning setting where labels of a sequence of data points are successively predicted, and the true label of a data point will be given before the next data point comes in [29]. A conformal anomaly detector (CAD) calculates a nonconformity score for each data point and then determines the p-value for the current data point. If the p-value is lower than a statistic significance level ϵ, then the current data point will be classified as an anomaly. SNN-CAD is designed by combining the k-nearest neighbors (kNN) technique into CAD to suit online settings better and using the undirected Hausdorff distance as the dissimilarity measure. The evaluation metric chosen by this work is accuracy. The datasets used are a synthetic dataset containing 260,000 two-dimensional trajectories [26] and a video trajectories dataset containing 239 video motion trajectories in which only two trajectories are labeled as anomalies [28]. Experimental results show that SNN-CAD can achieve high accuracy with no need to fine-tune parameters.

In 2020, Wang *et al.* [32] proposed a novel distance metric, the difference and intersection set (DIS) distance, which is essentially a ratio between the difference of the number of grids of two trajectories and the number of grids in the intersection set of two trajectories. The DIS distance metric is formally defined as:

$$\mathrm{DIS}(T_i, T_j) = \frac{|T_i| - |T_j|}{|T_i \cap T_j|} \tag{2.6}$$

where T_i and T_j are two trajectories. This work also proposed an anomalous trajectory detection and classification (ATDC) method based on the DIS distance metric to calculate anomaly scores of trajectories and use predefined thresholds to classify them into five groups, where each group represents a distinct moving pattern. Only one of the five groups represents normal trajectories, and the other four groups are different anomalous moving

patterns: global detour, local detour, global shortcut, and local shortcut. Experiments were conducted on the epfl/mobility dataset collected by Piorkowski *et al.* [27] which contains 463,860 trajectories produced by around 500 taxis in San Francisco, USA, in over 30 days. The evaluation metrics are F1 score and macro-F1 score. By adjusting a set of core parameters, ATDC is able to achieve high performance while being time-efficient.

2.1.3 Other Metric-based Methods

In 2011, Zhang *et al.* [38] proposed an isolation-based trajectory anomaly detection framework, iBAT. This method exploits the "few and different" property of trajectory anomalies and identifies trajectories that are easily isolated from other trajectories with the same source and destination as anomalies. The algorithm is based on the isolation forest (iForest) algorithm [20] proposed in 2008. The approach contains three steps, preprocessing, anomaly detection, and further analysis. In the preprocessing step, the geographical space is divided into grid cells of the same size, and each GPS coordinate in a trajectory falls into one grid cell. A trajectory mapped to the grid becomes a sequence of grid cells. They also interpolate grid cells between consecutive but not adjacent grid cells to solve the problem of low and inconsistent sampling rates. Then they apply an inverted index system on all trajectories to efficiently query trajectories that travel through a given pair of source and destination. The core method is based on iForest, which builds a random tree to isolate all instances, and the anomalous instances will have short paths on the tree while normal ones have much longer paths. In other words, anomalies are isolated in the early stage of building the random tree. They slightly change the iForest to make it work on trajectory data. They also apply the lazy learning method to ensure the capability for continuously growing training data. Experiments on taxi trajectory data collected from China show that iBAT performs well in real-world applications with high ROC-AUC scores. Further analyses show that iBAT can detect taxi frauds and road changes. In 2013, Chen *et al.* [3] adapted the iBAT method for the online anomaly detection task. By using the adaptive working window strategy, the new method named iBOAT can detect segments of trajectories that are anomalous in real-time.

2.2 Learning-based Approaches

The metric-based approaches are intuitive, and most of them are easy to understand and use with high explainability. However, their performance is highly sensitive to manually predefined parameters such as distance thresholds or proportions. It is critical for these approaches to choose appropriate values for predefined parameters, while it is very challenging to find the best values. Furthermore, trajectories and driving patterns are usually

very different from place to place, so even if we can find the best values of parameters for one city's trajectory data, we might need completely different values to achieve the same performance on another city's trajectory data. These issues bring challenges to applying metric-based approaches to real-life applications. On the other hand, machine learning techniques have been widely used in many fields, such as natural language processing and computer vision, in recent years. Therefore, some recent studies try to apply machine learning techniques, especially deep learning techniques, to the trajectory anomaly detection task. These learning-based approaches overcome the shortcomings of metric-based approaches and have better abilities to learn complex driving patterns of real-world trajectories.

In 2017, Wu *et al.* [36] proposed the Driving Behavior-based Trajectory Outlier Detection approach (DB-TOD), which learns a probabilistic model from historical driving behaviors with reinforcement learning techniques and can detect anomalous trajectories before they reach the destination (*i.e.*, when only a proportion of each trajectory is observed). To model the trajectory data and learn routing decision policies, they use the Maximum Entropy Inverse Reinforcement Learning (MEIRL) model [42]. There are three elements in reinforcement learning, namely state, action, and reward. To apply MEIRL on trajectory data, they define road segments as states, routing decisions such as taking turns or going straight as actions, and drivers as agents. They define the cost of taking action as a weighted sum of some selected features of historical driving behaviors. The model then is designed to give low probability scores to trajectories with high costs, and give high probability scores to low costs. The higher probability score means the trajectory is more likely to be an anomaly. The MEIRL model can already detect anomalous trajectories, but it has difficulty learning from implicit features of data. To overcome the drawback, they proposed an improved MEIRL model with Automatic Feature Correction (AFC-MEIRL). The improved model adds a learnable bias to the features of each action. If most vehicles choose to go straight at an intersection, and only a few vehicles choose to turn right, then the bias for taking the action of turning right will automatically increase, causing the cost of turning right to become higher. The stochastic gradient descent (SGD) algorithm is used for training the model. The evaluation metric is the normalized discounted cumulative gain (NDCG). Experimental results show that the proposed model achieves high NDCG scores with high efficiency in time.

In 2018, Gray *et al.* [10] introduced generative adversarial techniques into the field of trajectory anomaly detection and proposed IGMM-GAN, a model consisting of bidirectional GANs and an infinite Gaussian mixture model (IGMM). The generative adversarial network (GAN) was proposed by Goodfellow *et al.* [9] in 2014. The GAN model contains a generator network G and a discriminator network D. The generator network generates data from a latent space and tries to fool the discriminator. The discriminator network

tries to correctly identify which data is generated and which data is real. By playing the minimax game, the GAN gradually learns how to generate new data that looks very realistic. The bidirectional GAN proposed by Donahue *et al.* [6] adds an encoder network E based on the GAN. The encoder encodes data into the latent space, and the discriminator needs to classify the pairs $(G(z), z)$ and $(x, E(x))$ as generated data or real data, where z denotes a sample from the latent space. For detecting the anomalies, they map the trajectory to the latent space and calculate the Mahalanobis distance from the latent representation of the trajectory to every cluster in the space. The minimum Mahalanobis distance is the anomaly score of the trajectory. The latent space is modeled as an IGMM to capture multimodal mobility patterns. The IGMM models each cluster as a Gaussian distribution and models the prior probability of the clusters as a Dirichlet distribution. Experimental results show that IGMM-GAN can generate synthetic trajectories and detect anomalous trajectories effectively.

In 2018, Song *et al.* [30] proposed the Anomalous Trajectory Detection using Recurrent Neural Network (ATD-RNN) to automatically capture sequential information in trajectories and learn low-dimensional embeddings of them with RNN. The trajectories are preprocessed and fed into a stacked RNN module. The RNN cell could be one of the two kinds: LSTM cell and GRU cell. The dropout technique is used in the RNN to prevent the model from overfitting the training data. The output of the stacked RNN is E_i, which is the embedding of the i-th trajectory:

$$E_i = concat(h_t^1, h_t^2, ..., h_t^l) \tag{2.7}$$

where h_t^j is the hidden state at the last time step of the the j-th RNN layer, and l is the number of layers in the stacked RNN module. After obtaining E_i, we feed it into an MLP followed by a softmax layer to generate the anomalous probability of the trajectory. The experiments were conducted on a taxi trajectory dataset collected in Porto, Portugal. The results show that ATD-RNN has a much better performance than all of the baseline models in terms of accuracy and F1 score. Also, ATD-RNN with GRU cells achieves higher scores than ATD-RNN with LSTM cells in most SD-pair subsets of the dataset.

In 2020, Liu *et al.* [22] proposed an online trajectory anomaly detection model, Gaussian Mixture Variational Sequence AutoEncoder (GMVSAE). This model uses RNN and variational autoencoder to capture and encode sequential information of trajectories into a Gaussian mixture model consisting of various route types. The architecture of GMVSAE contains a token embedding layer, an inference network, a probability distribution of routes, and a generative network. The trajectories are first fed into an RNN (*i.e.*, the inference network) to generate a representation vector in a latent space. Instead of modeling the latent space as a Gaussian distribution, GMVSAE models it as a Gaussian mixture

distribution. During training, the representation vector is fed into another RNN (*i.e.*, the generative network) that tries to restore the original trajectory with the encoded information. During detection, the mean vectors of all Gaussian components in the Gaussian mixture model are fed into the generative network. If the probability of generating the given trajectory from the mean vectors is low, the trajectory will have a high anomaly score, and otherwise it is likely to be traveling on a normal route. Experimental results show that GMVSAE outperforms TRAOD, iBAT, and DB-TOD, and it is suitable for online environments due to its high efficiency in time.

3
Preliminary

This chapter introduces some neural network architectures used in our model. Section 3.1 introduces the GRU, which is an architecture that captures long-term dependencies better than RNN. Section 3.2 introduces the attention mechanism that overcomes the long-term dependency problem in RNN. Section 3.3 introduces the variational autoencoder, which is a widely used generative model.

3.1 GRU

The Recurrent Neural Network (RNN) is naturally suitable for processing sequentially structured data such as text, voice signals, and spatio-temporal trajectories. However, the RNN has problems with long-term memories, which means the information at the beginning of a sequence will gradually vanish as the RNN reads in the subsequent points in the sequence. To solve this problem, Hochreiter and Schmidhuber [12] introduced the Long Short-Term Memory networks (LSTM) in 1997. In 2014, Cho *et al.* [5] proposed the Gated Recurrent Unit (GRU) as a simplified version of LSTM. In contrast to LSTM that has three gates, namely forget gate, input gate, and output gate, GRU only contains two gates, reset gate and update gate. Figure 3.1 illustrates the GRU, where the red dotted line rectangle is the reset gate, and the blue dotted line rectangle is the update gate. The yellow nodes are activation functions, where σ denotes the sigmoid function. The red nodes with $\times$ and $+$ are pointwise multiplications and additions. The arrows show flows

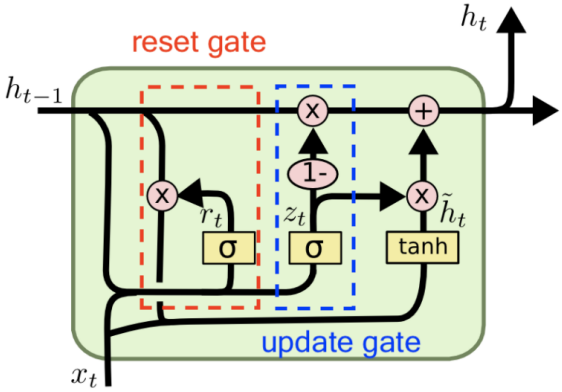

Figure 3.1: Gated Recurrent Unit [25]

and concatenations of vectors.

The reset gate is similar to the forget gate in LSTM, which controls what information to be forgot. The input at the current timestep is denoted as x_t. The hidden state from the previous timestep is denoted as h_{t-1}. The current input is concatenated with the previous hidden state before feeding into a sigmoid function. The output r_t is the output of the reset gate, which can be formally defined as:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{3.1}$$

The update gate is similar to a combination of forget gate and input gate in LSTM. This gate controls what information can be erased and what information need to be added into the hidden state. The concatenated vector of the current input and previous hidden state is fed into a sigmoid function to produce z_t:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{3.2}$$

The new candidate hidden state $\tilde{h}_t$ is then generated with x_t and the processed previous hidden state with some useless information erased:

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{3.3}$$

Finally, the hidden state for the current timestep is calculated by a weighted sum of the previous hidden state and the current candidate hidden state, where z_t controls which side has a larger impact:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{3.4}$$

Overall, the structure and performance of GRU are similar to LSTM, but the time cost of training GRU is not as much as LSTM since there are fewer gates and operations. In our work, we use GRU as it is faster in training without decreasing performance.

3.2 Attention

The attention mechanism mimics how human beings capture and process information. When we scan a piece of information such as an image, we will automatically pay more attention to the part that attracts us the most (*e.g.*, the cute cat in the center of the photo) and less attention to other areas of the image. Similarly, the attention mechanism in machine learning uses a weighted sum to give more attention to the item in the sequence that is most relevant to the current item and less attention to irrelevant items.

There are three important elements in the attention mechanism: key, value, and query. We use an example of reading a book to explain the concepts. If we have a book that introduces different Asian cultures and we are specifically interested in the Japanese culture, then we might pay more attention to chapters introducing Japanese culture, and less or no attention to other chapters. In this example, the query is "Japanese culture", the key might be the title of each chapter, and the value is the text of each chapter. As Figure 3.2 illustrates, we first compute the similarity between the query and each key in the sequence.

$$s_i = \mathcal{F}(Q, K_i) \tag{3.5}$$

for $i = 1, 2, ..., n$, where n is the length of the sequence, and $\mathcal{F}$ is a function that computes the similarity. The s_i scores will then go through a softmax layer and become "weights" of the values, denoted as a_i. Finally, we calculate a weighted sum of the values to produce the attention scores:

$$\text{Attention}(Q, K, V) = \sum_{i=1}^{n} (a_i V_i) \tag{3.6}$$

There are a few choices for the similarity function $\mathcal{F}$, such as the following functions:

- Dot product:

$$\mathcal{F}(Q, K_i) = Q \cdot K_i \tag{3.7}$$

- Cosine similarity:

$$\mathcal{F}(Q, K_i) = \frac{Q \cdot K_i}{||Q|| \cdot ||K_i||} \tag{3.8}$$

- Multilayer perceptron (MLP):

$$\mathcal{F}(Q, K_i) = \text{MLP}(Q, K_i) \tag{3.9}$$

Apart from the above functions of calculating the similarity between the query and the key, there is a specific attention mechanism proposed by Vaswani *et al.* [31] that becomes a popular choice. In the famous work that proposed the transformer model, they also introduced the scaled dot-product attention. The attention method is formally defined

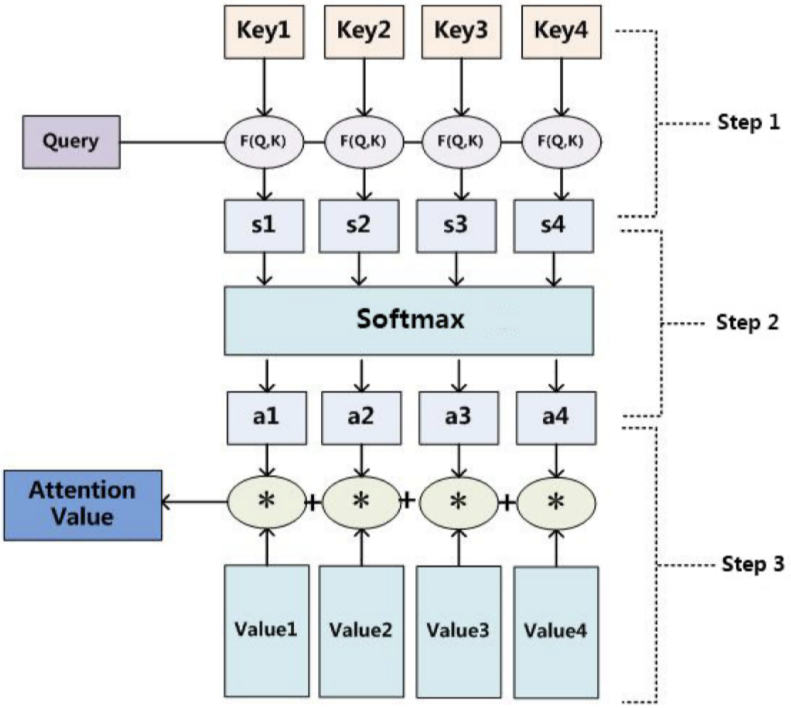

Figure 3.2: Attention Mechanism [39]

as:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{3.10}$$

where d_k is the dimensionality of the keys. The reason for adding a scaling factor is that the dot product of two random variables q and k will have a variance d_k if both variables have a variance of 1. The scaling factor cancels out the effects on the distribution brought by dot products. In one of the inference modules, we use the scaled dot-product attention to capture the global dependencies within the trajectories.

When we use the same sequence for Q, K, and V, we force the sequence to pay attention to itself, hence the name self-attention or intra-attention. Instead of comparing the similarity between the query and the key and generating weights for values, self-attention aims to find the relations between any two items within the same input sequence. As we mentioned in Section 3.1, RNN suffers from the long-term dependency problem. Although LSTM and GRU alleviate the problem by delicately designed recurrent units, the problem of long-term dependency still exists. Self-attention is also friendly to parallel computing, which the RNN cannot easily achieve. Due to these reasons, more and more researchers use the self-attention mechanism as a replacement for RNN. A typical example is the study by Vaswani *et al.* [31], where they proposed the transformer model using self-attention layers instead of RNN layers, and claimed the attention mechanism is all we need for processing the sequentially structured data.

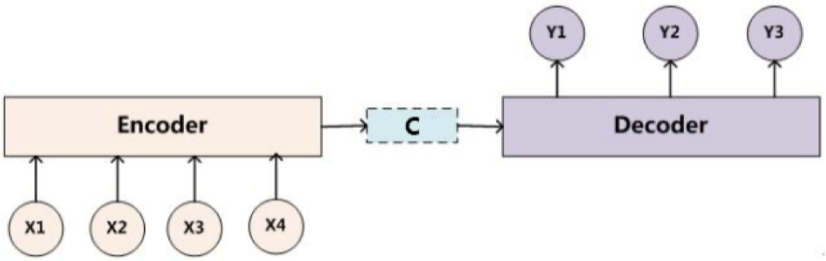

Figure 3.3: Encoder-Decoder Framework [39]

3.3 Variational Autoencoder

3.3.1 Encoder-Decoder Framework

The variational autoencoder is based on the encoder-decoder framework, so we introduce it first. The architecture of the encoder-decoder framework is shown in Figure 3.3. As the name suggests, the encoder encodes the input sequence into an intermediate representation C, and the decoder decodes C into another sequence as the output. We denote the input sequence as $X = (x_1, x_2, ..., x_n)$, and the output sequence as $Y = (y_1, y_2, ..., y_m)$, then the intermediate representation is:

$$C = \mathcal{F}(X) = \mathcal{F}(x_1, x_2, ..., x_n) \tag{3.11}$$

where $\mathcal{F}$ is a non-linear function. We then have:

$$y_i = \mathcal{G}(C, y_1, y_2, ..., y_{i-1}) \tag{3.12}$$

for $i = 1, 2, ..., m$, where $\mathcal{G}$ is a non-linear function, and y_i is the current item to be generated. This framework takes a sequence and outputs another sequence. In the field of natural language processing, the encoder-decoder framework accepts a sentence (*i.e.*, a sequence of words) in one language and generates another sequence of words. The output sequence would be the translation of the input sequence for translation tasks, and the answer to the question for question-answering tasks.

3.3.2 Autoencoder

The target sequence of the encoder-decoder framework can be different from the input sequence. When the target is set to be the same as the input, it becomes the architecture of an autoencoder. The main idea of the autoencoder is to convert a high-dimensional input vector into a low-dimensional vector in a latent space. To ensure that the compressed vector contains all essential information of the input vector, the decoder is trained to restore the input as much as possible. Both encoder and decoder can have multiple

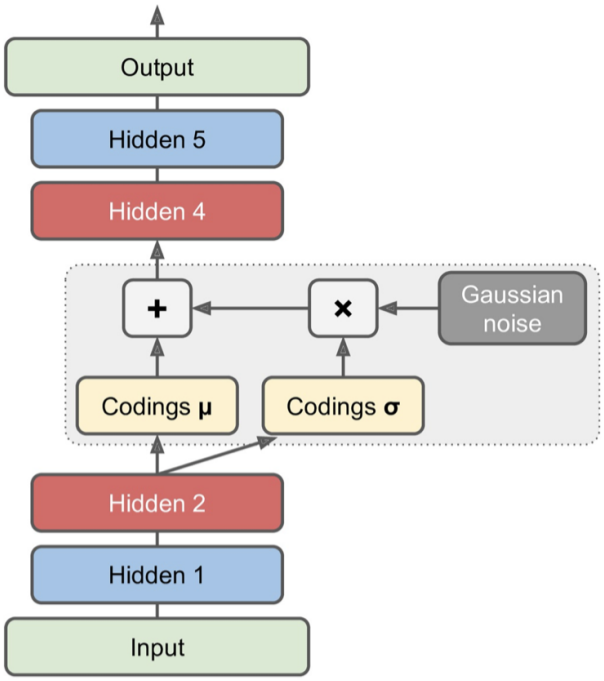

Figure 3.4: Variational Autoencoder [8]

neural network layers, but they need to be symmetrical with regard to the central layer, which is also called the bottleneck of the entire model. The autoencoder is widely used in many areas to learn low-dimensional representations of the input data. However, the autoencoder lacks the ability to generate new valid data and is prone to overfitting because its latent space of representation vectors is not explicitly structured.

3.3.3 Variational Autoencoder

To adapt the autoencoder for generative purposes, we need to explicitly regularize the latent space with a statistical model. We denote the input as x, and the latent representation of the input as z, then the probability distribution of the latent representation is $p(z|x)$. The encoder aims to convert the input vector to a proper distribution in the latent space. When reconstructuring the input, we draw a sample z from the distribution $z \sim q(z|x)$. The decoder takes z and aims to exactly reconstruct x.

From the statistic perspective, we assume that the observed data x is generated from the latent variable z, and we want to infer z from x. By Bayesian theorem, we have:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{3.13}$$

To infer z from x means we need to compute $p(z|x)$. However, $p(x)$ is usually intractable in practice, meaning that it is very difficult to directly compute $p(z|x)$. To solve this problem, we use the variational inference method. The word "variational" essentially

means to convert the problem of estimating a value to an optimization problem. Since we cannot directly compute $p(z|x)$, we approximate it by another probability distribution $q(z|x)$ that is easier to compute. Now the problem of computing $p(z|x)$ is converted to minimizing the difference between $p(z|x)$ and $q(z|x)$, *i.e.*, $D_{KL}(q(z|x)||p(z|x))$. The notation of $D_{KL}(\cdot||\cdot)$ is the Kullback-Leibler divergence (KL divergence), which is a common measure for difference between two probability distributions. This KL divergence can be further transformed into a sum of two terms, which is also the loss function for training the model:

$$D_{KL}(q(z|x)||p(z)) + (-E_q[\log p(x|z)]) \tag{3.14}$$

The first term measures the distance between the true distribution $p(z)$ and our approximation $q(z|x)$, and the second term is the reconstruction loss. In practice, $p(z|x)$ is typically assumed to be a Gaussian distribution. Figure 3.4 illustrate a variational autoencoder. The encoder (*i.e.*, $q(z|x)$) processes the input data and generates two vectors, a mean μ and a standard deviation σ. Using the reparameterization trick, the step of sampling from the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ is converted to computing $\mu+\sigma\epsilon$, where ϵ is a small Gaussian noise. The sample vector is then fed into the decoder (*i.e.*, $p(x|z)$), and the decoder reconstructs the input x as $\hat{x}$.

Both autoencoder and variational autoencoder have a continuous latent space. Each training instance can be encoded and mapped to a vector in the lower-dimensional latent space. The difference is that the autoencoder does not explicitly regularize the latent space, which might lead to overfitting. This also means that a randomly sampled vector from the latent space might not generate meaningful output. In contrast, the latent space in the variational autoencoder is explicitly modeled as a Gaussian distribution, and randomly sampled vectors generate semantically meaningful outputs, which means the variational autoencoder can be used for generative purposes.

4
Methodology

This chapter introduces our novel time-aware trajectory anomaly detection framework. Figure 4.1 is an illustration of the proposed framework. The four main parts of the framework are: preprocessing, inference module, route space, and generative module. In the preprocessing step, the geographical space is divided into grid cells of the same size to mitigate the data sparsity problem. In this way, the sequence of GPS coordinates of each trajectory is mapped to the grid and becomes a sequence of grid cells. The sequence of grid cells and the sequence of timestamps of a trajectory are fed into a spatial embedding layer and a temporal embedding layer, respectively, to project the discrete inputs into a continuous embedding space. The inference module accepts embeddings of both grid cell and timestamp sequences of a trajectory and generates a time-aware route representation of the input trajectory. The time-aware route representation is denoted by a vector in the latent *route space*. We argue that trajectories that follow similar driving routes should be close in the latent route space, forming clusters of routes. Motivated by this, we use a Gaussian mixture model to capture different types of routes, and the mean of each Gaussian to denote the representation of the *normal route* of each route type. During training, the generative module accepts the time-aware route representations and tries to restore the original trajectories. During detection, we first find the most relevant route type. Then, the generative module accepts the mean of the Gaussian distribution corresponding to the route type, and produces the probability of the trajectory traveling on a normal route. Section 4.1, 4.2, 4.3, and 4.4 introduce each part of the framework

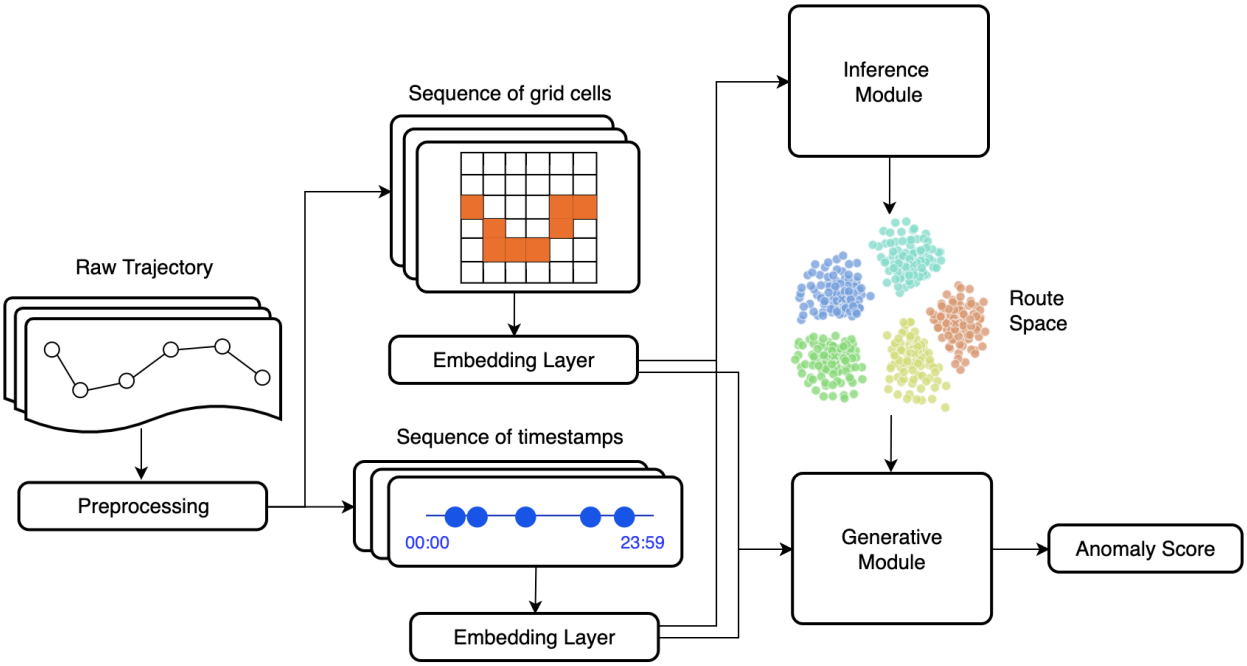

Figure 4.1: Overview of the Time-aware Trajectory Anomaly Detection Framework

respectively. Section 4.5 explains the method of training the model. Section 4.6 introduces the detect-by-generation strategy and how we generate anomaly scores.

4.1 Preprocessing

Every point in a raw trajectory $P = \{p_1, p_2, \ldots, p_n\}$ contains both spatial information and temporal information. To deal with each type of information separately, we need to split each trajectory into two sequences: a sequence of GPS coordinates and a sequence of timestamps. Intuitively, different source-destination pairs (SD pairs) might have different normal driving patterns. For example, most people might choose the shortest route when the candidate routes between the source and destination have similar road conditions. However, if the shortest route has a bad road condition, such as rocky or muddy roads, most people might prefer farther routes to avoid getting troubles on the road. Thus, we group trajectories by their SD pairs to discover normal driving patterns. However, even if two trajectories share the same source and destination, their exact GPS coordinates recorded for the two locations might be slightly different due to the precision of GPS devices. This will cause the two trajectories to be identified as belonging to two distinct SD pairs, leading to severe data sparsity issues. Following previous works [38, 3, 19], we draw a rectangular spatial area that contains all trajectories in datasets, split this area into many grid cells that share the same width and height, and map the sequence of GPS coordinates of each trajectory to the grid. Each GPS coordinate is mapped to a cell in the grid. The resulting sequence of grid cells is $X = \{x_1, x_2, \ldots, x_n\}$ where each x_i is

a token of a grid cell. By doing so, we get approximate positions of the moving object represented by grid cells instead of exact coordinates, and hence solve the data sparsity problem.

In the sequence of timestamps, each point is a timestamp that includes date and time information. In Unix timestamp format, each timestamp is represented by a single integer that means how many seconds have passed since 00:00:00 UTC, 1 January 1970, until now. We want to keep only the information of the time on the day, meaning that we treat 21:00 yesterday and 21:00 today as the same time point. We also want to treat the nearby time points as the same time, so we need to discretize the time of a day. We split the 24 hours of a day into 5760 timeslots so that each timeslot has a length of 15 seconds. Each timestamp is then mapped to one of the 5760 timeslots. The resulting sequence of timestamps is $T = \{t_1, t_2, \ldots, t_n\}$ where each t_i is a token of a timeslot.

4.2 Inference Module

After preprocessing the trajectories, we obtain a sequence of grid cells X and a sequence of timestamps T for each trajectory in datasets. We feed the two types of sequences into embedding layers to convert grid cell tokens and timeslot tokens into vectors in embedding spaces. The embedded X and T are denoted as X_e and T_e, respectively. Then we feed X_e and T_e into an inference module to infer the time-aware route representation. The inference module is denoted as a conditional probability distribution $q_\phi(r|X, T)$, where ϕ is the set of all parameters of the inference module, and r is the time-aware route representation, which is a high-dimensional vector in the route space. We assume r is a sample drawn from a probability distribution because this allows for some uncertainties and makes the model more robust to noises in input data. For the simplicity of the model, we assume $q_\phi(r|X, T)$ as a Gaussian distribution with a mean μ_{XT} and a standard deviation σ_{XT}. From a perspective of statistics, the inference module can be represented as a probabilistic graphical model (PGM) shown in Figure 4.2. The grey circles are observed variables, *i.e.*, what we observe from datasets. The white circles are hidden variables that we cannot directly observe and can only infer by some means. The arrows indicate dependencies between variables.

The biggest challenge here is that many existing works try to infer routes using X alone, but how to utilize time information T as well in the inference step is still a problem. We propose three approaches that combine spatial and temporal information in different ways to generate time-aware route representations of trajectories. The three approaches are named after the way they combine X and T: linear combination module, self-attention module, and Gaussian combination module. These approaches share the same PGM, and the only difference is how to obtain μ_{XT} and σ_{XT}.

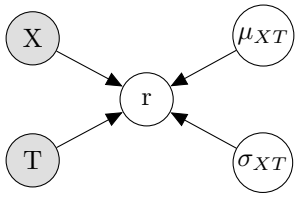

Figure 4.2: PGM of Inference Module

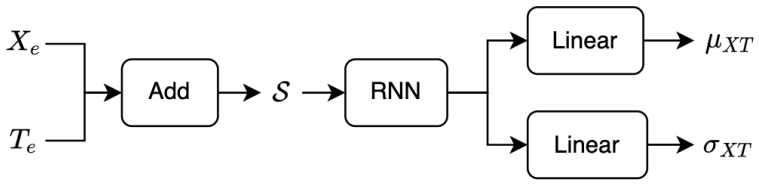

Figure 4.3: Linear Combination Module

4.2.1 Linear Combination Module

Figure 4.3 illustrates our first approach. The first approach is to add the sequences of embedded grid cells and timestamps to obtain a new sequence that combines spatial and temporal information: $\mathcal{S} = X_e + T_e$. Every i-th point in $\mathcal{S}$ is $s^{(i)} = x_e^{(i)} + t_e^{(i)}$, where $x_e^{(i)}$ and $t_e^{(i)}$ are the i-th point in X_e and T_e. Since the recurrent neural network (RNN) is naturally better at capturing hidden information in sequences, we use an RNN to encode the spatio-temporal sequence $\mathcal{S}$. RNN iterates through each item in $\mathcal{S}$ while repeatedly updating an internal state that keeps the sequential information accumulated so far. At each timestep i, we have:

$$h_i = f_1(s_i, h_{i-1}) \tag{4.1}$$

where h_i is the hidden state of the RNN at timestep i, s_i is the i-th point in $\mathcal{S}$, and f_1 is a non-linear function that can be either LSTM or GRU and will be learned during model training. The length of $\mathcal{S}$ is n, so the final state of the RNN is h_n, and we use it as a new representation of $\mathcal{S}$. We denote two linear transformation functions as g_1 and g_2 and apply them on h_n:

$$\begin{aligned} \mu_{XT} &= g_1(h_n) \\ \sigma_{XT} &= g_2(h_n) \end{aligned} \tag{4.2}$$

These functions will be learned during training as well. Now we have inferred parameters of the Gaussian distribution of the underlying route, we obtain r by drawing a sample from the distribution:

$$r \sim q_\phi(r|X,T) = \mathcal{N}(\mu_{XT}, \sigma_{XT}^2 I) \tag{4.3}$$

where $\mathcal{N}(\cdot,\cdot)$ denotes a Gaussian distribution and I is an identity matrix.

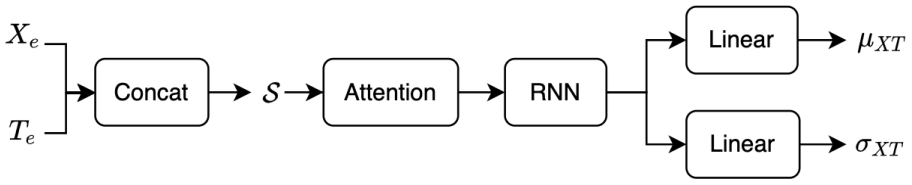

Figure 4.4: Self-attention Module

4.2.2 Self-attention Module

Our first approach combines X_e and T_e linearly, which does not allow much complexity, so the trained model might not fit the data very well. Furthermore, using an RNN alone to capture sequential information hidden in the trajectory might not be good enough. For example, the starting point of a trajectory might have an influence on the following points, even including points that are far away from the starting point. RNN is not good at remembering information that it read many timesteps ago, and even though LSTM and GRU have a better long-term memory, the problem still exists. Figure 4.4 illustrates our second approach. We introduce the attention mechanism in this approach to overcome the long-term memory problem and capture potential relationships globally between any two points in a sequence.

The first step is to concatenate the two sequences X_e and T_e:

$$\mathcal{S} = concat(X_e, T_e) \tag{4.4}$$

Following the previous work [31], we then use the Scaled Dot-Product Attention instead of other attention mechanism variations. We let Q, K, and V all equal to $\mathcal{S}$, which means query, key, and value will all be the concatenated sequence. This is the self-attention mechanism since the sequence is "paying attention" to itself.

$$\begin{aligned} \text{Attention}(Q, K, V) &= \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V \\ &= \text{Softmax}\left(\frac{\mathcal{S}\mathcal{S}^\top}{\sqrt{d_\mathcal{S}}}\right) \mathcal{S} \\ &= \{a_1, \ldots, a_n\} \end{aligned} \tag{4.5}$$

The Scaled Dot-Product Attention captures the global relationships between two location points, two timestamps, or even a location point and a timestamp. By doing this, we are not assuming or limiting what kind of influence a location point or a timestamp can have on any other points in the same trajectory. The output of the attention is $A = \{a_1, \ldots, a_n\}$. The next step is to feed the attention output into an RNN. At each timestep i, we have:

$$h_i = f_1(a_i, h_{i-1}) \tag{4.6}$$

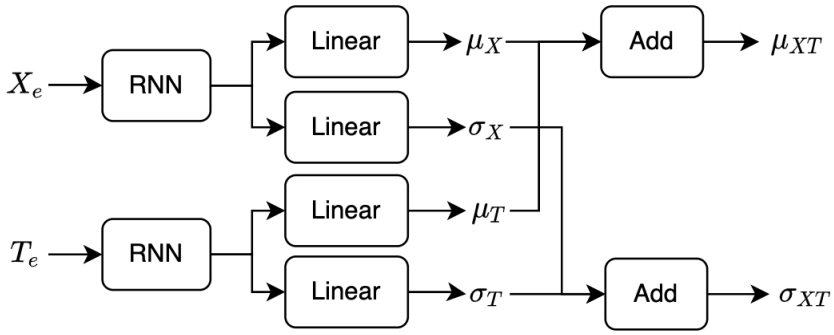

Figure 4.5: Gaussian Combination Module

where h_i is the hidden state of the RNN, and f_1 is a non-linear function that can be either LSTM or GRU. Then we perform linear transformations and sample the underlying route r of the trajectory from a Gaussian distribution:

$$\begin{aligned} \mu_{XT} &= g_1(h_n) \\ \sigma_{XT} &= g_2(h_n) \\ r \sim q_\phi(r|X,T) &= \mathcal{N}(\mu_{XT}, \sigma_{XT}^2 I) \end{aligned} \tag{4.7}$$

where the symbols are defined the same as in the previous section.

4.2.3 Gaussian Combination Module

The previous two approaches use some methods to combine X_e and T_e before feeding into an RNN to capture spatio-temporal driving patterns. In contrast, our third approach shown in Figure 4.5 captures the spatial and temporal driving patterns separately and then combines the encoded information to obtain the time-aware representations of routes. We assume that the latent representations of X and T follow the Gaussian distribution. In this way, their linear combination is also a Gaussian distribution, which represents the distribution of the underlying routes of the given trajectory. This approach deals with spatial and temporal sequences separately until the last step so that the two types of driving patterns can be fully captured before mixing the different information. We first use an RNN to encode the embedded sequence of grid cells X_e:

$$h_i = f_1(x_e^{(i)}, h_{i-1}) \tag{4.8}$$

where $x_e^{(i)}$ is the i-th point in X_e, and other symbols carry the same meaning as the previous section. Similarly, we use another RNN to encode the embedded sequence of timestamps T_e:

$$h'_i = f_2(t_e^{(i)}, h'_{i-1}) \tag{4.9}$$

where $t_e^{(i)}$ is the i-th point in T_e, h'_i is the hidden state of the second RNN at timestep i, and f_2 is the non-linear function of the cell of the second RNN. We perform linear transformations g_1 and g_2 on h_n, and perform linear transformations g'_1 and g'_2 on h'_n to obtain parameters of Gaussian distributions for the two types of sequences:

$$\begin{aligned} \mu_X &= g_1(h_n) \\ \sigma_X &= g_2(h_n) \\ \mu_T &= g'_1(h'_n) \\ \sigma_T &= g'_2(h'_n) \end{aligned} \tag{4.10}$$

The final step is to linearly combine the two Gaussian distributions:

$$\begin{aligned} r \sim q_\phi(r|X,T) &= \mathcal{N}(\mu_{XT}, \sigma_{XT}^2 I) \\ &= \mathcal{N}(\mu_X, \sigma_X^2 I) + \mathcal{N}(\mu_T, \sigma_T^2 I) \\ &= \mathcal{N}(\mu_X + \mu_T, (\sigma_X^2 + \sigma_T^2) I) \end{aligned} \tag{4.11}$$

where the symbols are defined the same as in the previous section.

4.2.4 Reparameterization Trick

All of the proposed approaches for the inference module require sampling from some probability distributions. The random node r will prevent backpropagation from going through, making it difficult to train our model. A common trick used to solve this problem is the reparameterization trick proposed by Kingma and Welling [14]. To understand why we need the reparameterization trick and how it works, we start by thinking about the gradient of the expectation of a function $f_\theta(r)$:

$$\begin{aligned} \nabla_\theta \mathbb{E}_{p(r)} \left[f_\theta(r) \right] &= \nabla_\theta \left[\int_r p(r) f_\theta(r) dr \right] \\ &= \int_r p(r) \left[\nabla_\theta f_\theta(r) \right] dr \\ &= \mathbb{E}_{p(r)} \left[\nabla_\theta f_\theta(r) \right] \end{aligned} \tag{4.12}$$

where $p(r)$ is the probability density function of r. We successfully converted a gradient of an expectation to an expectation of a gradient. However, we cannot do the same thing

when $p(r)$ is parameterized by θ:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{p_\theta(r)}\left[f_\theta(r)\right] &= \nabla_\theta \left[\int_r p_\theta(r) f_\theta(r) dr\right] \\
&= \int_r \nabla_\theta \left[p_\theta(r) f_\theta(r)\right] dr \\
&= \int_r p_\theta(r) \nabla_\theta f_\theta(r) dr + \int_r f_\theta(r) \nabla_\theta p_\theta(r) dr \\
&= \mathbb{E}_{p_\theta(r)}\left[\nabla_\theta f_\theta(r)\right] + \int_r f_\theta(r) \nabla_\theta p_\theta(r) dr
\end{aligned} \tag{4.13}$$

We are familiar with the first term as it is essentially the same as the result of Equation 4.12, but the second term is problematic since it is not an expectation of anything. Without a theoretical guarantee, we cannot get a good estimate of the gradient $\nabla_\theta \mathbb{E}_{p_\theta(r)}\left[f_\theta(r)\right]$ in practice even after sampling a large number of r using the Monte Carlo method. The estimate is just always bad with a high variance. Recall that r is sampled from $q_\phi(r|X,T)$. The solution is to introduce a deterministic function $g_\theta(\epsilon, X, T)$ such that $r = g_\theta(\epsilon, X, T)$ and $\epsilon \sim p(\epsilon)$. Now we transform the previous gradient again with the new functions:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{p_\theta(r)}\left[f(r)\right] &= \nabla_\theta \mathbb{E}_{p(\epsilon)}\left[f(g_\theta(\epsilon, X, T))\right] \\
&= \mathbb{E}_{p(\epsilon)}\left[\nabla_\theta f(g_\theta(\epsilon, X, T))\right] \\
&\approx \frac{1}{L}\sum_{l=1}^{L} \nabla_\theta f(g_\theta(\epsilon^{(l)}, X, T))
\end{aligned} \tag{4.14}$$

By doing so, we once again converted the gradient of the expectation to the expectation of the gradient. This guarantees that we can use the Monte Carlo method to give estimates of the gradient $\nabla_\theta \mathbb{E}_{p_\theta(r)}\left[f(r)\right]$ as long as $g_\theta(\epsilon, X, T)$ is differentiable. In other words, we can say that the backpropagation can go through the node r as it is now a deterministic node with an input from a random node ϵ. Since $r \sim q_\phi(r|X,T) = \mathcal{N}(\mu_{XT}, \sigma_{XT}^2 I)$, a valid form of reparameterization is $r = \mu_{XT} + \epsilon \cdot \sigma_{XT}$, where ϵ is a random noise sampled from a standard Gaussian distribution $\mathcal{N}(0,1)$. We apply this form of reparameterization in the inference module to better train our model.

4.3 Route Space

Now we have time-aware route representations inferred from trajectories, but we still do not know how these routes are distributed in the latent route space, which routes are normal, and which routes are anomalous in the context of the given dataset. We want to model the route space as a probability distribution so that we can not only know which routes are anomalous, but also measure how anomalous they are. This is not a trivial

task because there are various types of routes in a big city. For example, some routes are highways, while some are urban streets with strict speed limits. In terms of city areas, some routes are in the central area of the city, some are in the rural area, and some are crossing from rural areas to downtown. Furthermore, considering time information together creates even more types of routes. For example, some routes are urban street routes at peak time, some are urban street routes at midnight, and some are rural area routes at working hours.

To solve the problem, we propose to model the route space as a Gaussian mixture model. This statistic model consists of a number of components, where each component is a Gaussian distribution, and there is a probability distribution of selecting one of the components. We denote the number of components as C. In our context, using the Gaussian mixture model means that we assume there are C different types of routes in our dataset. Each type of route has a different meaning in the real world, such as some specific road type or area of the city at some specific period of time.

To elaborate on the Gaussian mixture model, we need to introduce two kinds of probability distributions. One of them is a categorical distribution $p_\psi(c) = \text{Categorical}(\alpha)$, where c is a specific type of route (*i.e.*, the specific Gaussian distribution component of the Gaussian mixture model) and α represents all parameters of the distribution. The other one is a Gaussian distribution $p_\psi(r|c) = \mathcal{N}(\mu_c, \sigma_c^2 I)$ for each type of route c, where μ_c and σ_c are mean and standard deviation of the group of routes. The set of parameters of the Gaussian mixture model is denoted as $\psi = \{\alpha, \mu_c, \sigma_c\}$. Combining the two distributions, we then have $p_\psi(r) = p_\psi(r|c)p_\psi(c)$ as the probability of the route r in the route space.

When introducing inference modules, we were considering how to infer $q_\phi(r|X, T)$. Now since we have another latent variable c that denotes the type of routes in the model of the route space, we need to make a minor modification to the inference module. Instead of inferring r alone, we now need to infer c as well: $q_\phi(r, c|X, T)$. Using mean-field approximation, we have:

$$q_\phi(r, c|X, T) = q_\phi(r|X, T)q_\phi(c|X, T) \tag{4.15}$$

The conditional distribution we want to infer is broken into two terms. We deal with each term one by one. The first term $q_\phi(r|X, T)$ can already be inferred by the proposed inference modules (refer to Equation 4.3, 4.7, and 4.11). The second term $q_\phi(c|X, T)$ is more tricky. The probability that a trajectory belongs to a specific type c can be considered as an equivalent of the probability of type c given the underlying route. This

means that we can define $q_\phi(c|X,T)$ to be equal to $p_\psi(c|r)$. By Bayesian theorem:

$$\begin{aligned} q_\phi(c|X,T) &:= p_\psi(c|r) \\ &= \frac{p_\psi(r|c)p_\psi(c)}{\sum_{i=1}^{C}\left(p_\psi(r|c_i)p_\psi(c_i)\right)} \end{aligned} \tag{4.16}$$

The denominator of the fraction is a constant, so we can omit it. $p_\psi(c)$ is the prior probability which represents how likely each component in the Gaussian mixture model will be selected. If we assume $p_\psi(c)$ as a uniform distribution, then we have:

$$q_\phi(c|X,T) := p_\psi(c|r) \propto p_\psi(r|c) \tag{4.17}$$

The probability of a trajectory having an underlying route of type c is proportional to the probability of a route r given that the route type is selected.

4.4 Generative Module

As a critical part of the detect-by-generation strategy, we need to design a generative module that can generate a trajectory from a given route. The generative module is also a probability distribution $p_\theta(X,T|r)$ with θ as all of the associated parameters. The higher the probability is, the more likely the generated X and T follow the normal route r. When training the generative module, we aim to maximize the probability of the sequence of grid cells X and the sequence of timestamps T being generated when we know their underlying route in the route space, *i.e.*, $p_\theta(X,T|r)$. It is worth noting that X and T here are the same sequences from which r is inferred. Therefore, this variational inference architecture can be seen as a variational autoencoder where the encoder is the inference module and the decoder is the generative module. Figure 4.6 shows PGM of the process of trajectory generation. First, we select a type of route from the latent space following the Gaussian mixture model. With the parameter set α, a type c is selected (C types of routes in total). We use the mean and standard deviation of the type of routes to generate r. Finally, we use r to generate X and T separately, and the trajectory generation is completed.

This generation task is different from other tasks such as image generation. The generation of a trajectory includes generating each point, and we need to do it sequentially (which is very different from image generation) because at each timestep, we should be able to see all previous points that are already generated. This means that the information we need to consider at each timestep is dynamic and keeps changing. RNN is suitable for such tasks due to its sequential nature. More specifically, at each timestep i, the generation of x_i depends on two things: the given route r and the sequence of all previous

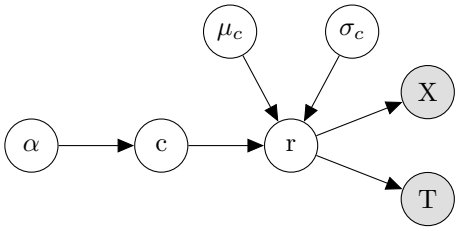

Figure 4.6: PGM of the Generative Process

points $X_{<i} = \{x_1, x_2, \ldots, x_{i-1}\}$. Therefore, the probability of x_i is $p_\theta(x_i|r, X_{<i})$. From the perspective of RNN, we recurrently update a hidden state based on r and the current sequence $X_{\leq i} = \{x_1, x_2, \ldots, x_i\}$. The way of updating the hidden state is:

$$h_i = f_3(x_i, h_{i-1}) \qquad h_0 = r \tag{4.18}$$

where f_3 is a non-linear function to be learned and h_i is the hidden state of the RNN. The initial hidden state is set to the selected route. At each timestep i, x_i is then generated from a categorical distribution:

$$\begin{aligned} x_i \sim p_\theta(x_i|r, X_{<i}) &= p_\theta(x_i|h_{i-1}) \\ &= \text{Categorical}(\text{Softmax}(g_3(h_{i-1}))) \end{aligned} \tag{4.19}$$

where g_3 is a linear function that converts the previous hidden state h_{i-1} to a vector of logits with the length of the number of grid cells. We then use a softmax function to convert the logits vector to a probability distribution indicating how likely a grid cell will be chosen at the current timestep. Finally, we sample x_i from the categorical function. By doing this recurrently, we obtain a sequence of grid cells X. Similarly, we can obtain a sequence of timestamps T using another RNN (with a GRU or LSTM cell):

$$\begin{aligned} h'_i &= f'_3(t_i, h'_{i-1}) \qquad h'_0 = r \\ t_i \sim p_\theta(t_i|r, T_{<i}) &= p_\theta(t_i|h'_{i-1}) \\ &= \text{Categorical}(\text{Softmax}(g'_3(h'_{i-1}))) \end{aligned} \tag{4.20}$$

where h'_i is the hidden state of the second RNN at timestep i, f'_3 is the non-linear function of the cell of the second RNN, and g'_3 is a linear function that converts h'_{i-1} to a vector of logits.

4.5 Training Method

Our framework has a number of parameters that need to be learned during training, including $\phi = \{f_1, g_1, g_2\}$ for the linear combination module and the self-attention module, $\phi = \{f_1, f_2, g_1, g_2, g_1', g_2'\}$ for the Gaussian combination module, $\psi = \{\alpha, \mu_c, \sigma_c\}$ for the route space, $\theta = \{f_3, f_3', g_3, g_3'\}$ for the generative module, embeddings for grid cells and timeslots. The training objective is to optimize the inference module, the route space, and the generative module jointly so that the generative module can restore the original trajectory with the time-aware representation of the underlying route inferred by the inference module. Formally speaking, the training objective is to maximize the marginal log-likelihood of our generative module reproducing the training dataset:

$$\begin{aligned} \log p_\theta(\text{traj}_1, \text{traj}_2, \ldots, \text{traj}_N) &= \log p_\theta(X_1, X_2, \ldots, X_N, T_1, T_2, \ldots, T_N) \\ &= \sum_{i=1}^{N} \log p_\theta(X_i, T_i) \end{aligned} \tag{4.21}$$

where N is the number of trajectories in the training dataset, X_i and T_i are the sequences of grid cells and timestamps separated from the i-th trajectory in the dataset. For each trajectory, we are interested in its posterior probability:

$$\begin{aligned} p_{\theta,\psi}(r, c|X, T) &= \frac{p_{\theta,\psi}(X, T|r, c)p_{\theta,\psi}(r, c)}{p_{\theta,\psi}(X, T)} \\ &= \frac{\prod_c p_\psi(c) \prod_r p_\psi(r|c) p_\theta(X, T|r)}{\sum_c \int_r \prod_c p_\psi(c) \prod_r p_\psi(r|c) p_\theta(X, T|r)} \end{aligned} \tag{4.22}$$

On the right-hand side of the equation, the denominator is intractable in practice, so we cannot directly calculate the posterior probability. To solve this problem, we introduce $q_\phi(r, c|X, T)$ into the equation as the approximation of the posterior distribution $p_{\theta,\psi}(r, c|X, T)$ and use the variational inference method to derive the evidence lower bound (ELBO) of the marginal log-likelihood of the trajectory:

$$\begin{aligned} \log p_\theta(X, T) &= \log \mathcal{L}(\phi, \psi, \theta|X, T) \\ &= \log \int_r \sum_c p_{\theta,\psi}(X, T, r, c) \\ &= \log \int_r \sum_c q_\phi(r, c|X, T) \frac{p_{\theta,\psi}(X, T, r, c)}{q_\phi(r, c|X, T)} \\ &\geq \int_r \sum_c q_\phi(r, c|X, T) \log \frac{p_{\theta,\psi}(X, T, r, c)}{q_\phi(r, c|X, T)} \quad \text{(Jensen's inequality)} \end{aligned} \tag{4.23}$$

We convert the right-hand side of Equation 4.23 to the expectation form:

$$\begin{aligned}\log p_\theta(X,T) &\geq \mathbb{E}_{q_\phi(r,c|X,T)}[\log p_{\theta,\psi}(X,T,r,c)] - \mathbb{E}_{q_\phi(r,c|X,T)}[\log q_\phi(r,c|X,T)] \\ &= \mathbb{E}_{q_\phi}[\log p_\theta(X|r)] + \mathbb{E}_{q_\phi}[\log p_\theta(T|r)] - D_{KL}(q_\phi(c|X,T)||p_\psi(c)) \\ &\quad - D_{KL}(q_\phi(r|X,T)||p_\psi(r|c))\end{aligned} \tag{4.24}$$

where $D_{KL}(a||b)$ denotes the KL divergence from distribution b to distribution a. Since we want to maximize the marginal log-likelihood $\log p_\theta(X,T)$ and the last line of Equation 4.24 is the evidence lower bound of $\log p_\theta(X,T)$ (*i.e.*, a theoretical guarantee that the actual likelihood is higher than or equal to it), we can set the training objective to maximizing the lower bound instead of $\log p_\theta(X,T)$. Furthermore, we can break down the first two terms of the last line into the sum of the likelihood of generating each item in the sequence:

$$\begin{aligned}\mathbb{E}_{q_\phi}[\log p_\theta(X|r)] &= \mathbb{E}_{q_\phi}\left[\sum_{i=1}^{n} \log p_\theta(x_i|X_{<i},r)\right] \\ \mathbb{E}_{q_\phi}[\log p_\theta(T|r)] &= \mathbb{E}_{q_\phi}\left[\sum_{i=1}^{n} \log p_\theta(t_i|T_{<i},r)\right]\end{aligned} \tag{4.25}$$

From the perspective of machine learning, the first two terms in the ELBO are reconstruction losses, the third term is categorical loss, and the fourth term is latent loss. We negate the ELBO and use it as the loss function when training our model.

4.6 Detect-by-generation Strategy

The previous sections have introduced how we infer the underlying routes and then regenerate trajectories. In this section, we introduce the detect-by-generation strategy that uses the normal route r^* and the generative module $p_\theta(X,T|r)$ to compute the anomaly score. More specifically, for each input trajectory, we first find the most probable route type c in the latent route space, and then feed the input trajectory and the mean of the route type (*i.e.*, the normal route) into the generative module to obtain the probability of the trajectory follows the normal route of type c. The lower the probability, the higher the anomaly score, and the more likely the trajectory is an anomaly.

The normal route is defined as the route that most people would take to travel between two places. Following the previous work [22], we use the center of each Gaussian distributed cluster μ_c to approximate the normal route of the cluster r_c^*. The probability of a trajectory traveling on the normal route μ_c is the probability of generating the

trajectory from μ_c. The anomaly score of a trajectory is calculated as:

$$\begin{aligned} score(X,T) &= 1 - \arg\max_c \exp\left[\frac{\log p_\theta(X,T|\mu_c)}{n}\right] \\ \text{where} \quad \log p_\theta(X,T|\mu_c) &= \log p_\theta(X|\mu_c)p_\theta(T|\mu_c) \\ &= \sum_{i=1}^{n} \log p_\theta(x_i|X_{<i},\mu_c)p_\theta(t_i|T_{<i},\mu_c) \end{aligned} \tag{4.26}$$

It is worth noting that we divide the log-likelihood $\log p_\theta(X,T|\mu_c)$ by n, the length of the trajectory, as a normalization. The advantage of using the detect-by-generation strategy is that online detection tasks can be very efficient. The way of updating anomaly scores in the online detection task is:

$$score(X_{\leq i+1}, T_{\leq i+1}) = 1 - \arg\max_c \exp\left[\frac{\log p_\theta(X_{\leq i}|\mu_c)p_\theta(x_{i+1}|X_{\leq i},\mu_c)p_\theta(T_{\leq i}|\mu_c)p_\theta(t_{i+1}|T_{\leq i},\mu_c)}{i+1}\right] \tag{4.27}$$

where $X_{\leq i} = \{x_1, x_2, \ldots, x_i\}$. The range of the anomaly score is $[0,1]$, where 1 means the given trajectory cannot be generated from normal routes in the Gaussian mixture model (*i.e.*, the trajectory is an anomaly), and 0 means the trajectory travels on a normal route. We can set a threshold when applying the proposed framework in real-world applications, and an anomalous trajectory is detected once the anomaly score is higher than the threshold.

5
Evaluation

This chapter introduces the details of how we conduct experiments, presents the experimental results, and shows further assessments of how well our model performs on detecting anomalous trajectories and capturing time-aware driving patterns. To fully evaluate our proposed model, we list the research questions that need to be answered by experiments and assessments:

- How is the overall performance of our model in the task of offline anomalous trajectory detection? Does it outperform the current state-of-the-art model?
- As we proposed three inference modules in the previous chapter, how is the performance of each inference module?
- How do parameters affect the performance of models?
- In the online anomalous trajectory detection task, how does our model perform?
- Can our model discover the underlying time-aware driving patterns?

This chapter is organized as follows. Section 5.1 introduces the dataset, preprocessing steps, the method of generating anomalies systematically, models to be evaluated, and the evaluation metric. Section 5.2 evaluates the parameter sensitivity, the performance on offline and online anomaly detection, the efficiency, and also compares the three inference modules. At the end of Section 5.2, we use visualization methods to show that our model is able to capture the underlying driving patterns in terms of both space and time.

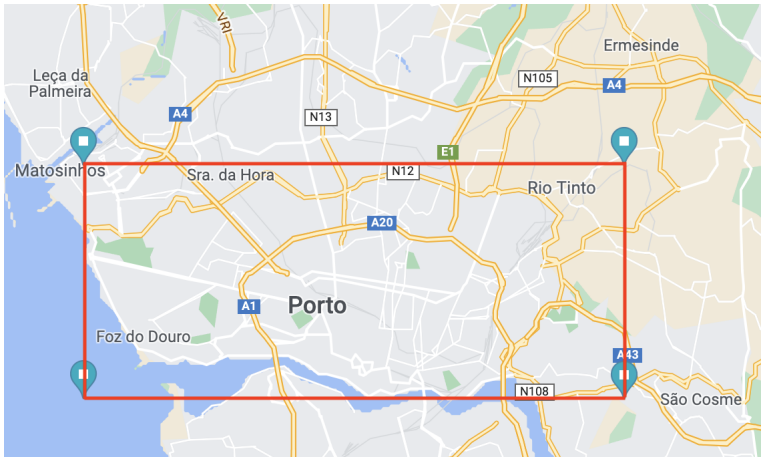

Figure 5.1: Map of Porto

5.1 Experimental Setup

5.1.1 Datasets and Preprocessing

In this work, we perform experiments on a real-world dataset that contains the trajectories of 442 taxis running in Porto, a city in Portugal, over a period of 12 months from 7th January 2013 to 30th June 2014. The sampling rate is 15 seconds, which means each taxi sends its current GPS coordinates every 15 seconds. The dataset contains some noisy GPS coordinates that are out of Porto or even out of Portugal. Following the previous studies [22, 38, 3], we draw a rectangular spatial area on the map of Porto and remove all noises that are out of the drawn area. As Figure 5.1 shows, we only focus on the taxi trajectories in the area of the red rectangle that contains the central area of the city. We then split the area into 100m×100m grid cells so that trajectories with close source and destination locations can be grouped together later. The resulting grid map has 51×118 cells, and we assign each cell a unique cell token. We group trajectories by their source-destination pairs (SD pairs) and filter SD pairs that contain no more than 25 trajectories. The resulting dataset has 301,063 trajectories. The way to split this dataset into a training set and a test set is different from other machine learning tasks. Since the driving patterns between different SD pairs are irrelevant [38], if we randomly split all trajectories into two sets, many source cells and destination cells might be missing in the training set, and we will have no information on the driving patterns of SD pairs containing those cells. Following the previous work [22], we pick all trajectories except the last five from each SD pair as the training set, and the remaining trajectories are used as the test set. This method of splitting the dataset makes sure that every SD pair appears in the training set at least 20 times, as every SD pair in the filtered dataset contains at least 25 trajectories.

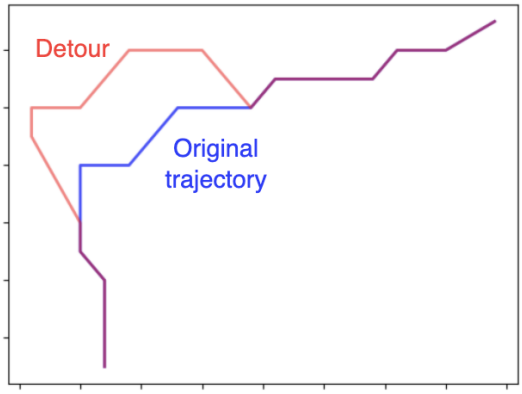

Figure 5.2: Example of the Generated Anomalous Trajectory

5.1.2 Anomaly Generation

Due to the lack of labeled datasets that are available for trajectory anomaly detection, some existing works [38, 3] attempt to label anomalies manually. However, manually labeling datasets is very expensive and error-prone, especially when we have a large amount of data to be labeled. Additionally, unlike other tasks such as labeling images, manually labeling anomalous trajectories is particularly difficult as it is difficult to determine which routes are normal and which are not. Inspired by some previous studies [40, 22], we systematically generate anomalous trajectories for the evaluation of the model. The characteristic of anomalous trajectories, especially for taxis trajectories, is that they tend to take unnecessary detours so that the taxi drivers can charge the passengers more. Although not all anomalous trajectories must be detours, it is definitely a critical issue that we want to deal with, especially for taxi trajectory anomaly detection. Therefore, we generate anomalous trajectories by shifting a part of the trajectory away in a direction that is perpendicular to the moving direction of the taxi. As Figure 5.2 shows, the blue part is the original trajectory which seems to be normal, while the red part is the shifted blue part and should be identified as an anomaly.

The parameters controlling the shifted part are denoted as d and p, where d is the distance between the original part and the shifted part, *i.e.*, how far we shift the chosen part of the trajectory, and p is the proportion of the trajectory to be shifted. For example, $d = 5$ and $p = 0.1$ means that the randomly chosen part of the trajectory will contain 10% of the consecutive points in the trajectory, and this part will be shifted perpendicularly to the moving direction with a distance of 5 grid cells. For the experiments, we change the values of d and p and evaluate the performance of different models. Following previous studies [38, 40, 36, 22], we set the number of generated anomalous trajectories to be 5% of the size of the entire dataset. This is a simulation of the real-world situation as the anomalies are usually the minority of all trajectory data.

5.1.3 Evaluated Models

The trajectory anomaly detection methods we use in the experiments include the following:

- **iBAT** [38]: This is an isolation-based detection method that identifies whether the trajectory is an anomaly by how much it can be isolated from the reference trajectories within the same SD pair.
- **Sequence Autoencoder (SAE)** [23]: This is the classic autoencoder model adapted for sequence inputs and sequence outputs with RNN layers. The objective of training this model is to minimize the reconstruction error, which is also used for calculating the anomaly score.
- **Variational Sequence Autoencoder (VSAE)** [22]: This is the combination of SAE and variational autoencoder. The latent space of spatio-temporal routes is assumed to be a Gaussian distribution, which is a smoother latent space than the latent space of SAE.
- **Gaussian Mixture Variational Sequence Autoencoder (GMVSAE)** [22]: This is the state-of-the-art model in the field of trajectory anomaly detection. This model can be seen as a combination of the VSAE and the Gaussian mixture model. Instead of a single Gaussian distribution, this model assumes the latent space as a number of Gaussian distributions, and each one represents a specific type of route. There is a hyperparameter in GMVSAE, C, that represents the number of Gaussian distribution components and can significantly affect its performance. We set C to three different values, 10, 20, and 50. The value of C is denoted following a dash. For example, GMVSAE -10 represents the GMVASE model with $C = 10$.

The above methods are used as baselines that give benchmarks we want to go beyond. The following are our proposed models with different designs of inference modules: Time-aware Anomalous Trajectory Detection with Linear Combination Inference Module (TAD-LC), Time-aware Anomalous Trajectory Detection with Self-attention Inference Module (TAD-SA), and Time-aware Anomalous Trajectory Detection with Gaussian Combination Inference Module (TAD-GC). Similar to GMVSAE, our TAD models have the hyperparameter C as well. For consistency, the value of C is also set to 10, 20, and 50.

5.1.4 Evaluation Metrics

As we introduced in Chapter 4, our model does not only predict which trajectory is an anomaly, but also gives an anomaly score that measures how anomalous the trajectory is.

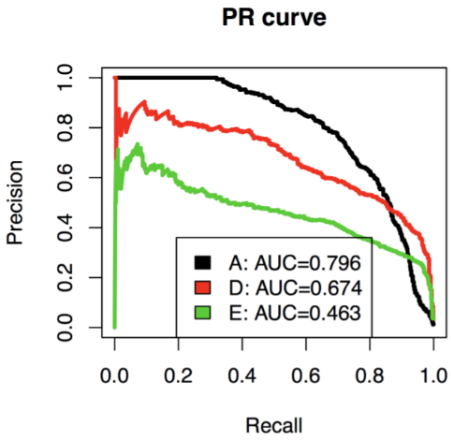

Figure 5.3: Example of PR Curve [13]

Table 5.1: PR-AUC Scores for Checking Parameter Sensitivity

d	1	3	5
p	0.1	0.3	0.5
$s_1 = 32, s_2 = 128$	0.3279	0.8890	0.9523
$s_1 = 32, s_2 = 256$	**0.3892**	0.8919	0.9532
$s_1 = 64, s_2 = 256$	0.3886	**0.8981**	**0.9548**

The range of the anomaly score values is between 0 and 1. The higher the value is, the more anomalous the trajectory is. The metric we use to evaluate the experimental results is the Area Under Curve (AUC) score of the Precision-Recall curve (PR-AUC). Figure 5.3 illustrates how the PR curve generally looks. Precision is calculated by $TP/(TP + FP)$, and recall is calculated by $TP/(TP+FN)$, where TP is true positive, FP is false positive, and FN is false negative. For highly imbalanced datasets where the positive class has only a few data points, a model might predict all data points as positive and still have a high recall score as TP is the number of all positive data points and FN is zero. However, the precision score will be very low since FP is high as the most positive predictions we made are incorrect. The PR curve tries to give a complete view of the performance of models compared to using either precision or recall alone. The goal is to have a PR curve that is close to the upper right corner of the graph, which makes a bigger area under the curve, hence a higher AUC score. The previous studies [40, 22] found that PR-AUC is suitable for imbalanced datasets and hence an appropriate metric for the anomalous trajectory detection task.

5.2 Model Evaluation

5.2.1 Parameter Sensitivity

Before training and evaluating all models, it is worth noting that there are two parameters in our proposed framework that might affect the performance, the size of grid cell embeddings s_1 and the size of hidden states s_2. To test the sensitivity to the two parameters, we use the GC inference module and set the number of Gaussian components to 10, *i.e.*, $C = 10$. We vary d and p and briefly check the performance of the models under three settings of the parameters. The results of the experiments are shown in Table 5.1. We can see that when d and p are larger, the difference between models with different s_1 and s_2 is not significant. However, when $d = 1$ and $p = 0.1$, the first model is significantly worse than the other two models. The third model outperforms the first one by 18.5%. The second model outperforms the first one by 18.7%. Considering the time cost of training and testing brought by larger s_1 and s_2, the second set of s_1 and s_2 balances the performance and the efficiency and thus is the best choice of parameters. The following experiments are all conducted based on $s_1 = 32$ and $s_2 = 256$.

5.2.2 Performance in the Offline Detection Task

We first evaluate models with the offline anomalous trajectory detection task where models can observe the complete trajectories before making a prediction. Table 5.2 shows the evaluation results for baseline models and our proposed models with different inference modules under different settings of the hyperparameter C (denoted as -10, -20, and -50). The two parameters controlling how we generate the anomalous trajectories are d and p, and their values can be varied to compare the performance of all methods on different synthetic trajectory anomalies. We set $d = \{1, 3, 5\}$, $p = \{0.1, 0.3, 0.5\}$ and conduct the experiments.

From Table 5.2, we can see that our proposed models with different inference modules (TAD-LC, TAD-SA, and TAD-GC) are generally better than the baseline models, and most results are significantly better than the best baseline model. The best results under each setting of parameters are bolded, and all of them are produced by our proposed models. The iBAT model has the worst performance under all settings of parameters because it is a metric-based method while other models are based on deep learning methods. Among deep learning models, SAE is usually worse than VSAE because VSAE has a better generalization ability with its Gaussian distributed latent space. The performance of GMVSAE is impacted by the number of Gaussian components. GMVSAE is generally better than SAE but sometimes better than VSAE. From left to right, the PR-AUC scores of all models are generally increasing, which means that it is much easier to detect

Table 5.2: PR-AUC Scores for Offline Detection

d	1	3	5	1	3	5	1	3	5
p	0.1	0.1	0.1	0.3	0.3	0.3	0.5	0.5	0.5
iBAT	0.3116	0.3203	0.3230	0.4282	0.4575	0.4630	0.5653	0.6268	0.6337
SAE	0.3443	0.6241	0.7290	0.6772	0.8575	0.9049	0.8015	0.9088	0.9407
VSAE	0.3568	0.6438	0.7318	0.6610	0.8631	0.9147	0.8159	0.9225	0.9430
GMVSAE -10	0.3552	0.6472	0.7473	0.6771	0.8648	0.9094	0.7928	0.9116	0.9425
GMVSAE -20	0.3602	0.6392	0.7423	0.6820	0.8622	0.9052	0.7977	0.9112	0.9402
GMVSAE -50	0.3601	0.6388	0.7347	0.6817	0.8654	0.9077	0.7961	0.9141	0.9373
TAD-LC -10	0.3365	0.6622	0.7726	0.6673	0.8833	0.9282	0.8032	0.9320	**0.9589**
TAD-LC -20	0.3688	0.6770	0.7835	0.7002	0.8941	0.9310	0.8164	0.9350	0.9563
TAD-LC -50	0.3678	0.6784	**0.7876**	0.7014	0.8963	**0.9311**	0.8196	0.9343	0.9556
TAD-SA -10	0.3667	0.6779	0.7801	0.6962	0.8930	0.9283	0.8173	0.9361	0.9538
TAD-SA -20	0.3631	0.6634	0.7724	0.6960	0.8868	0.9216	0.8201	0.9314	0.9508
TAD-SA -50	0.3722	0.6776	0.7805	0.7040	**0.8983**	0.9287	0.8252	**0.9380**	0.9549
TAD-GC -10	**0.3892**	0.6754	0.7815	**0.7118**	0.8919	0.9257	**0.8267**	0.9313	0.9532
TAD-GC -20	0.3555	0.6683	0.7836	0.6833	0.8890	0.9252	0.8079	0.9306	0.9547
TAD-GC -50	0.3787	**0.6841**	0.7811	0.7093	0.8901	0.9288	0.8259	0.9330	0.9558

the anomaly when there are more points in the trajectory that are different from normal trajectories, even if the shifted distance is not very long. In other words, the trajectory anomaly detection task becomes easier when p is higher. When p is fixed, a higher d that indicates a more prominent shifted part makes the detection easier and hence higher performance for all models. Comparing our TAD models with baselines, we can see that although the best PR-AUC scores of TAD are always higher than baselines, the improvements are very different under different settings of parameters. For example, when $d = 5$ and $p = 0.5$, *i.e.*, when we move half of a trajectory five grid cells away, the best result of TAD (0.9589) is only 1.7% higher than the best baseline (0.9430). However, when $d = 1$ and $p = 0.1$, the best result of TAD (0.3892) is 8.1% higher than the best baseline (0.3602). This shows that when it is easy to detect the anomalies, the improvement of TAD is not too much because other models can already achieve high scores. When the anomalies are not so obvious, the advantage of TAD is highlighted.

5.2.3 Performance Comparison of Inference Modules

Different inference modules and different values of C (*i.e.*, number of Gaussian components in the route space) have significant effects on the performance of TAD. From Table 5.2, we can see that when $d = 1$, TAD-GC consistently achieves the best PR-AUC score. When $d = 5$, TAD-LC consistently achieves the best PR-AUC score. This shows that different inference modules have different sensitivities to the shifted distance of the anomalous trajectories. When the shifted part is prominent, the LC module performs better, and when the shifted part is close to the normal route, the GC module can better differentiate

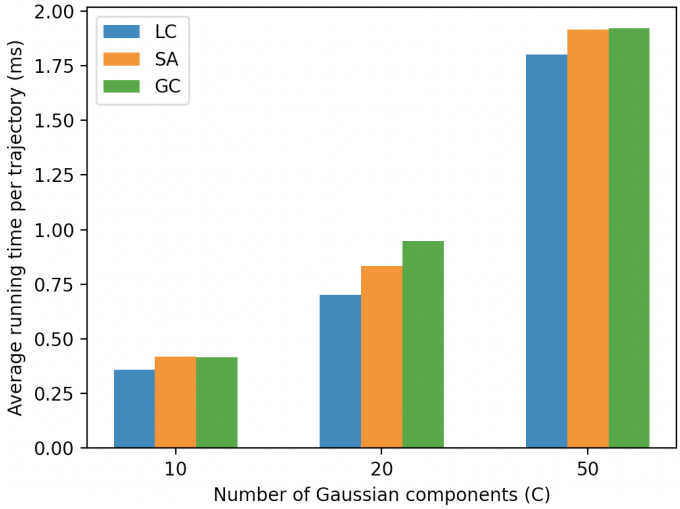

Figure 5.4: Efficiency Comparison for TAD Models

the normal routes and anomalies. For models with the LC module, TAD-LC-50 has a higher PR-AUC score than other models most of the time (6 out of 9 parameter settings of d and p). For the SA module, TAD-SA-50 is the best under 8 out of 9 parameter settings. For the GC module, TAD-GC-10 and TAD-GC-50 are equally good as they both achieve the best scores under 4 out of 9 settings. Therefore, for all three inference modules, $C = 50$ is the best setting for the number of Gaussian components. However, this is not the case for GMVSAE models. GMVSAE-10 is usually better than other GMVSAE models because the latent route space of GMVSAE considers spatial information only. In contrast, for TAD models, each Gaussian distribution in the route space is a type of time-aware representations of routes, and adding the time into consideration makes the number of routes become larger. To fit all types of routes, we need to have a large C value for TAD models. However, regardless of the value of C, 94% of all the 81 PR-AUC scores produced by TAD models beats the best baseline model under the same parameter settings, which proves that our TAD models are superior to baseline models even if the inference module and the value of C are not deliberately selected.

5.2.4 Efficiency Evaluation

Figure 5.4 compares the efficiency of TAD models with different inference modules and C values. We can see that although the LC module is a little more efficient than the other inference modules, their difference in running time is not significant under the same setting of C. $C = 50$ is the overall best setting of C, but the cost is a longer running time for each trajectory in model evaluation. However, even the worst performance is under 2 milliseconds per trajectory, which means 500 trajectories can be processed in only one second. This is fast enough for online anomaly detection tasks, given that we

Table 5.3: PR-AUC Scores for Online Detection

d	3	5	3	5
p	0.1	0.1	0.3	0.3
iBAT	0.3236	0.3211	0.4334	0.4353
SAE	0.3971	0.4456	0.6330	0.6455
VSAE	0.3970	0.4442	0.6428	0.6497
GMVSAE -10	0.4355	0.4817	0.6643	0.6769
GMVSAE -20	0.4369	0.4894	0.6717	0.6800
GMVSAE -50	0.4311	0.4812	0.6662	0.6759
TAD-LC -10	0.4121	0.4742	0.6464	0.6638
TAD-LC -20	0.4223	0.4781	0.6575	0.6675
TAD-LC -50	0.4269	0.4834	0.6642	0.6701
TAD-SA -10	0.4220	0.4791	0.6568	0.6660
TAD-SA -20	0.4213	0.4839	0.6612	0.6671
TAD-SA -50	0.4530	0.4907	0.6748	0.6769
TAD-GC -10	**0.4531**	**0.5018**	**0.6835**	**0.6851**
TAD-GC -20	0.4294	0.4876	0.6615	0.6697
TAD-GC -50	0.4439	0.4911	0.6707	0.6764

need to process the time information as well, and other models only process geographical locations.

5.2.5 Performance in the Online Detection Task

The experiments we conducted and showed in Table 5.2 assume that we can observe the complete trajectory before predicting whether it is an anomaly. However, it is usually better to detect anomalous trajectories earlier in real-world applications. For example, if we can detect the unnecessary detour made by the taxi driver before the driver finishes the trip, we can send a warning to the driver and the passengers or take other measures to ensure the driver can stop taking detours and the passengers do not pay too much for the trip. This is much better than taking action after the taxi fraud succeeded. Therefore, we also conduct online detection experiments where the models need to make a prediction after observing only half of the trajectory. As Table 5.3 shows, the iBAT model still has the worst performance. SAE and VSAE have similar scores. GMVSAE models with different values of C are well above both SAE and VSAE. Overall, the PR-AUC scores are lower than Table 5.2 as the detection of anomalies is harder when only the first half trajectory is observable. Our TAD models still achieve higher scores than baselines. When $d = 3$ and $p = 0.1$, TAD outperforms the best baseline model by 3.7%, which is a good improvement given the difficulty of the experimental settings. In terms of the difference in inference modules, all the highest scores are produced by TAD-GC-10, but it is worth

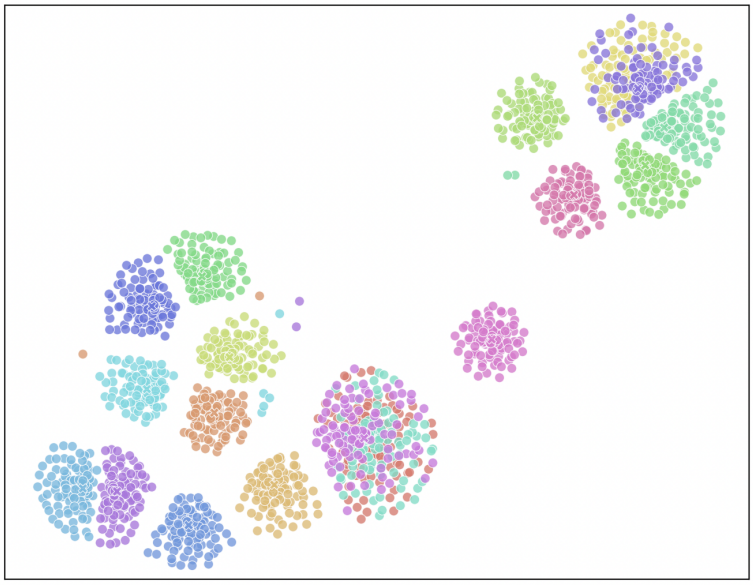

Figure 5.5: The Route Space of TAD-GC-20

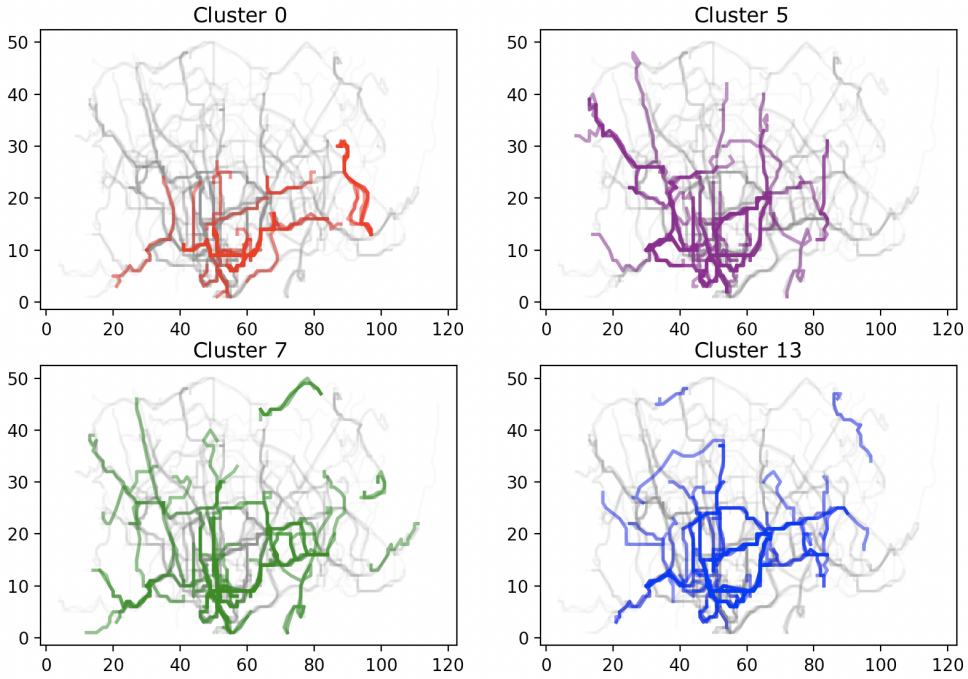

Figure 5.6: Trajectories Sampled from Different Route Types

noting that the scores of TAD-SA-50 and TAD-GC-50 are also very high compared to other models.

5.2.6 Case Study and Visualization

The PR-AUC scores show the performance of models in the anomaly detection task but do not give an intuitive demonstration of the models. In this subsection, we run case

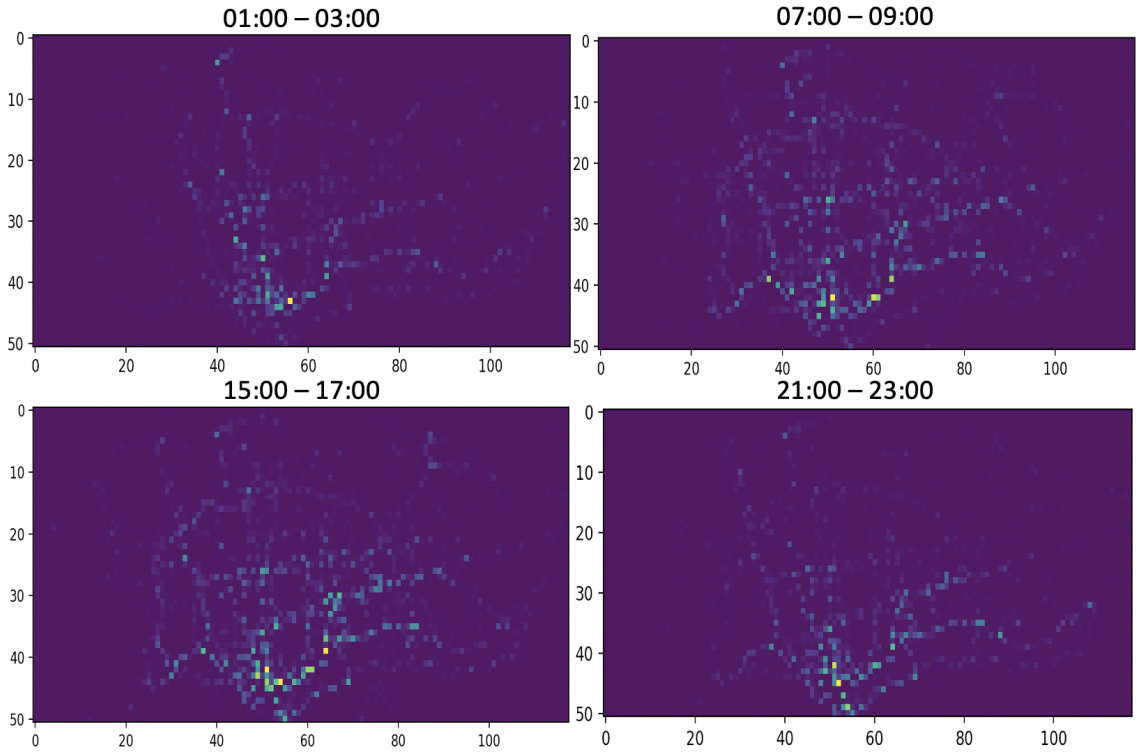

Figure 5.7: Heatmap of Trajectories for Different Time Periods

studies on the learned clusters in the latent route space. To investigate the effectiveness of our model for finding the time-aware routes, we also visualize the routes of a given time period.

Figure 5.5 shows a two-dimensional demonstration of routes in the high-dimensional route space of TAD-GC-20. The method of visualization is t-distributed stochastic neighbor embedding (t-SNE), which is commonly used to show clusters of vectors in a high-dimensional space. Each color in the graph represents a type of routes. As we can see, some clusters are mixed together, but most clusters are well separated. This shows that our model can capture and encode the routes of trajectories and thus form a meaningful route space.

Each Gaussian distributed cluster in the route space represents a route type. We randomly choose four clusters from TAD-GC-20 and visualize the trajectories belonging to the corresponding route types in Figure 5.6. The number on top of each plot is the cluster ID. The trajectories from cluster #0 are mostly on the east side of the city, while the trajectories from cluster #5 cover the whole west side. The trajectories from cluster #7 are mostly in the south, including some trajectories that travel across the city from the north to the south. The trajectories from cluster #13 are mostly trajectories that travel between the east and the west of the city. This visualization proves that each cluster in the route space carries a semantic meaning the route type in terms of spatial locations.

The time during the day has a significant impact on the geographical distribution of

trajectories. Figure 5.7 shows heatmaps of trajectories generated by TAD-GC-20 given different timestamps as inputs. It is worth noting that the moving patterns we can discover are more relevant to tourists since the dataset we use is a taxi trajectory dataset, and we assume most people do not use taxis to commute every day. At the beginning of the day, 01:00 - 03:00, most people are resting in their beds, so the city is not very active, and most trajectories during this time cover only a tiny area in the center of the city. In rush hours, 7:00 - 09:00, the entire city becomes much more active, and we can vaguely see the road network of Porto from the heatmap. The most crowded area is in the south because there are many scenic spots and historical sites such as Bolsa Palace and World of Discoveries, a museum introducing the early history of Portugal. Later in the day, 15:00 - 17:00, we can see more light spots on the heatmap. The central area of the city is even more active. Finally, during 21:00 - 23:00, the road network disappears while the center of the city remains relatively active. The visualization in Figure 5.7 shows that our model can successfully capture time-aware driving patterns.

6
Conclusion

This chapter summarizes of our research, presents some limitations of our work, and shows potential directions for future work. Section 6.1 shows achievements of our work. Section 6.2 presents limitations of our work. Section 6.3 suggests directions of future works.

6.1 Achievements

- We proposed a novel trajectory anomaly detection framework that integrates the temporal information. Experimental results conducted in both online and offline environments show that our framework outperforms all baseline models, including the state-of-the-art GMVSAE model.

- With our visualization, we showed that our framework successfully captures different driving patterns under different periods of time. This is an unprecedented achievement since previous works implicitly assume the driving pattern between two locations does not change over time.

- Our framework generates time-aware representations of trajectories, which can be potentially used in other downstream tasks such as trajectory similarity computation and trajectory classification.

6.2 Limitations

We list the limitations of our work as follows:

- The temporal information is difficult to be modeled as it is periodic. If we only consider the time during a day, the end of the day 23:59 is followed by the beginning of the next day 00:00. We modeled the time during a day as a straight stripe divided into many timeslots, but it is actually more like a circle where the end of the stripe touches the beginning of the stripe. If we model the time during a week, the end of one week is the beginning of the next one, and we have the periodicity again. This limitation in time modeling might affect the ability of our proposed framework to capture time-aware driving patterns.

- Due to the limit of time and computing resources, we use RNN layers with GRU units for good performance without losing the time efficiency. However, there are some replacements of RNN that can potentially boost performance.

- One of the inference modules is based on self-attention. The limitation of the self-attention for processing sequential data is that it does not consider the order of items within the sequence. Therefore, in the future, we might need to replace self-attention with a model that explicitly considers the order of the sequence.

- The training stage of our framework is essentially learning time-aware representations of trajectories. We only focus on trajectory anomaly detection in this work, but the trajectory representations can be potentially used in other related areas.

6.3 Future Work

We list the potential future works as follows:

- The temporal information needs to be handled with a better model that explicitly shows the periodicity of time. Instead of modeling only the time of one day, we can model the time of one week and hence be able to discuss the difference in driving patterns between workday and weekends. Instead of slicing the time into 15-second timeslots, we can also try different lengths of timeslots in the future.

- In recent years, the transformer model has been widely used in many research areas. Unlike the simple self-attention layer, the transformer model contains a layer of positional encoding that explicitly takes the order of sequence into account. In the future, we will use the transformer model to replace RNN and self-attention in the inference modules and study the effects on performance.

- The time-aware trajectory representations learned by our framework can be used for other related tasks. For example, with some appropriate distance metrics, we can potentially compute similarities of trajectories. We can also use the inference module and the latent route space to classify the type of trajectories. In the future, we can extend the usage of our framework to many other research areas of trajectory data.

Bibliography

[1] Helmut Alt. The computational geometry of comparing shapes. In *Efficient Algorithms*, pages 235–248. Springer, 2009.

[2] Francesco Calabrese, Massimo Colonna, Piero Lovisolo, Dario Parata, and Carlo Ratti. Real-time urban monitoring using cell phones: A case study in rome. *IEEE transactions on intelligent transportation systems*, 12(1):141–151, 2010.

[3] Chao Chen, Daqing Zhang, Pablo Samuel Castro, Nan Li, Lin Sun, Shijian Li, and Zonghui Wang. iboat: Isolation-based online anomalous trajectory detection. *IEEE Transactions on Intelligent Transportation Systems*, 14(2):806–818, 2013.

[4] Lu Chen, Yunjun Gao, Ziquan Fang, Xiaoye Miao, Christian S Jensen, and Chenjuan Guo. Real-time distributed co-movement pattern detection on streaming trajectories. *Proceedings of the VLDB Endowment*, 12(10):1208–1220, 2019.

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[7] Yong Ge, Hui Xiong, Zhi-hua Zhou, Hasan Ozdemir, Jannite Yu, and Kuo Chu Lee. Top-eye: Top-k evolving trajectory outlier detection. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1733–1736, 2010.

[8] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2019.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[10] Kathryn Gray, Daniel Smolyak, Sarkhan Badirli, and George Mohler. Coupled igmm-gans for deep multimodal anomaly detection in human mobility data. *arXiv preprint arXiv:1809.02728*, 2018.

[11] Daniel S Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)*, 24(4):664–675, 1977.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] Jens Keilwagen, Ivo Grosse, and Jan Grau. Area under precision-recall curves for weighted and unweighted data. *PloS one*, 9(3):e92209, 2014.

[14] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[15] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3):237–253, 2000.

[16] Joseph B Kruskal. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review*, 25(2):201–237, 1983.

[17] Rikard Laxhammar and Göran Falkman. Sequential conformal anomaly detection in trajectories based on hausdorff distance. In *14th International Conference on Information Fusion*, pages 1–8. IEEE, 2011.

[18] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. Trajectory outlier detection: A partition-and-detect framework. In *2008 IEEE 24th International Conference on Data Engineering*, pages 140–149. IEEE, 2008.

[19] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S Jensen, and Wei Wei. Deep representation learning for trajectory similarity computation. In *2018 IEEE 34th international conference on data engineering (ICDE)*, pages 617–628. IEEE, 2018.

[20] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[21] Meng Liu, Erik Brynjolfsson, and Jason Dowlatabadi. Do digital platforms reduce moral hazard? the case of uber and taxis. Working Paper 25015, National Bureau of Economic Research, September 2018.

[22] Yiding Liu, Kaiqi Zhao, Gao Cong, and Zhifeng Bao. Online anomalous trajectory detection with deep generative sequence modeling. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 949–960. IEEE, 2020.

[23] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[24] Fanrong Meng, Guan Yuan, Shaoqian Lv, Zhixiao Wang, and Shixiong Xia. An overview on trajectory outlier detection. *Artificial Intelligence Review*, 52(4):2437–2456, 2019.

[25] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, August 2015.

[26] Claudio Piciarelli, Christian Micheloni, and Gian Luca Foresti. Trajectory-based anomalous event detection. *IEEE Transactions on Circuits and Systems for video Technology*, 18(11):1544–1554, 2008.

[27] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. CRAWDAD dataset epfl/mobility (v. 2009-02-24). Downloaded from `https://crawdad.org/epfl/mobility/20090224`, February 2009.

[28] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. Incremental local outlier detection for data streams. In *2007 IEEE symposium on computational intelligence and data mining*, pages 504–515. IEEE, 2007.

[29] Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008.

[30] Li Song, Ruijia Wang, Ding Xiao, Xiaotian Han, Yanan Cai, and Chuan Shi. Anomalous trajectory detection using recurrent neural network. In *International Conference on Advanced Data Mining and Applications*, pages 263–277. Springer, 2018.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[32] Jingwei Wang, Yun Yuan, Tianle Ni, Yunlong Ma, Min Liu, Gaowei Xu, and Weiming Shen. Anomalous trajectory detection and classification based on difference and intersection set distance. *IEEE Transactions on Vehicular Technology*, 69(3):2487–2500, 2020.

[33] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, and Gao Cong. Reverse k nearest neighbor search over trajectories. *IEEE Transactions on Knowledge and Data Engineering*, 30(4):757–771, 2017.

[34] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, and Xiaolin Qin. Fast large-scale trajectory clustering. *Proceedings of the VLDB Endowment*, 13(1):29–42, 2019.

[35] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. Torch: A search engine for trajectory data. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 535–544, 2018.

[36] Hao Wu, Weiwei Sun, and Baihua Zheng. A fast trajectory outlier detection approach via driving behavior modeling. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 837–846, 2017.

[37] Guan Yuan, Penghui Sun, Jie Zhao, Daxing Li, and Canwei Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, 2017.

[38] Daqing Zhang, Nan Li, Zhi-Hua Zhou, Chao Chen, Lin Sun, and Shijian Li. ibat: detecting anomalous taxi trajectories from gps traces. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 99–108, 2011.

[39] Junlin Zhang. The attention model in deep learning. `https://zhuanlan.zhihu.com/p/37601161`, June 2018.

[40] Guanjie Zheng, Susan L Brantley, Thomas Lauvaux, and Zhenhui Li. Contextual spatial outlier detection with metric learning. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2161–2170, 2017.

[41] Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):1–41, 2015.

[42] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.