PrivTrace: Differentially Private Trajectory Synthesis by Adaptive Markov Models

Haiming Wang, *Zhejiang University;* Zhikun Zhang, *CISPA Helmholtz Center for Information Security;* Tianhao Wang, *University of Virginia;* Shibo He, *Zhejiang University;* Michael Backes, *CISPA Helmholtz Center for Information Security;* Jiming Chen, *Zhejiang University;* Yang Zhang, *CISPA Helmholtz Center for Information Security*

https://www.usenix.org/conference/usenixsecurity23/presentation/wang-haiming

This paper is included in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

PrivTrace: Differentially Private Trajectory Synthesis by Adaptive Markov Models

Haiming Wang[1] Zhikun Zhang[2*] Tianhao Wang[3] Shibo He[1]
Michael Backes[2] Jiming Chen[1] Yang Zhang[2]

[1]*Zhejiang University*
[2]*CISPA Helmholtz Center for Information Security*
[3]*University of Virginia*

Abstract

Publishing trajectory data (individual's movement information) is very useful, but it also raises privacy concerns. To handle the privacy concern, in this paper, we apply differential privacy, the standard technique for data privacy, together with Markov chain model, to generate synthetic trajectories. We notice that existing studies all use Markov chain model and thus propose a framework to analyze the usage of the Markov chain model in this problem. Based on the analysis, we come up with an effective algorithm PrivTrace that uses the first-order and second-order Markov model adaptively. We evaluate PrivTrace and existing methods on synthetic and real-world datasets to demonstrate the superiority of our method.

1 Introduction

Trajectory data analysis plays an important role in tasks related to the social benefit such as urban planning and intelligent transportation [4, 47]. The trajectory data used usually comes from users who carry mobile devices to record their locations. However, the sensitive nature of the trajectory data also gives rise to privacy risks when being shared. Recent research demonstrated effective privacy attacks despite aggregation or anonymization, such as trajectory reconstruction [25], de-anonymization [11], and membership inference [12, 39]. Without a sufficient privacy protection technique, trajectory data analysis is not possible since users will refuse to share their data.

A promising technique to overcome the privacy concern is *differential privacy* (DP) [17, 18, 48, 56], which has become the golden standard in the privacy community. Intuitively, on a given dataset, DP defines a random transformation process (algorithm) so that the output of the transformation contains some "noise" that can "cover" the existence of any possible record in the dataset. DP can be used by both companies and government agencies. With DP protection, hopefully, more users will be willing to share their data. From the government side, agencies can also share more data for the public good.

Most of the previous studies on differentially private trajectory data analysis focus on designing tailored algorithms for specific tasks, such as community discovering [53], participatory sensing [33], and recommendation [50]. Our paper focuses on a more general approach, where we publish a *synthetic trajectory dataset* that shares similar properties with the original one while satisfying DP. While publishing synthetic datasets would not work better than directly publishing aggregate statistics [42], this paradigm can easily enable any (even unseen) down-stream data analysis tasks without modifying existing algorithms and has been adopted in, e.g., the 2020 US Census data publication [10] and a dataset for population flow analysis [29]. There are several recent studies focusing on generating data satisfying DP, and they are typically composed of two steps: *model learning* and *generation* [36, 54, 57]. In the scenario of trajectory data, because the information is typically continuous, existing works [25, 28] also need to first *discretize* the data upfront.

Existing Work. The first work to generate synthetic trajectories with DP is DPT by He et al. [28]. It uses a set of uniform grids with different granularities to discretize the space and establishes multiple differentially private prefix trees [13] (each with a different grid) to model transitions; then generates data by random walk on the prefix trees. More recently, Gursoy et al. [25] proposed AdaTrace. It discretizes the geographical space into two kinds of grids using different granularities. Then a first-order Markov chain model and other three important features are learned based on the discretization results. Then the data is generated by random walk on the first-order Markov chain model with help of extracted features. The drawback of AdaTrace is that information contained in the first-order Markov chain model is not enough to generate data of high quality.

Our Contributions. Existing approaches either get only the first-order Markov chain model, which fails to obtain enough transition information, or get a high-order Markov chain model, which introduces excessive noise due to DP. Our

*Zhikun Zhang is the corresponding author.

idea is to reach a middle ground between DPT and AdaTrace. To this end, we employ only the first and second-order Markov chain model, for three reasons. First, previous studies have shown that the second-order Markov chain model can achieve promising accuracy for next step prediction [22, 32, 41]. Second, Markov chain model of higher order is too space and time-consuming. A k-order Markov chain model with m states needs m^{k+1} transition probability values. When m is 300, and k is 3, this requires 30GB of storage. Third, to satisfy DP, building more models will introduce more noise. Although more information is extracted, the amount of noise also grows. As a result, after some specific order of the Markov chain model, the overall information quality will decrease. In practice, we observe the threshold is between the second-order and third-order.

To generate high-quality trajectories, we use both the first-order model and the second-order model in our random-walk-based generation process. Every time to predict a state as the next step, we select between the first-order model and the second-order model. The selection principle contains two aspects of considerations: the effect of noise on different models and the confidence ability of models.

The standard Markov chain model does not have the information about where the trajectory starts and ends. We add virtual start and end in every trajectory to record this information. However, due to some actions to bound the sensitivity, the recorded information is biased. To reduce the bias, we propose an optimization-based method to estimate the trip distribution. The optimization problem is built based on the observation that a trajectory only contributes one to the count related to the virtual start and end. The experimental results show that our estimation method is effective, especially when the privacy budget is small.

We conduct empirical experiments on both synthetic and real-world trajectory datasets to compare PrivTrace with the state-of-the-art. PrivTrace consistently outperforms the existing methods for a variety of metrics. We further conduct comprehensive ablation studies to illustrate the effectiveness of the three main components of PrivTrace. One limitation of our approach is that the accuracy of the generated long trajectories is worse than that of the short trajectories. This is due to the fact that long trajectories are more complicated and more challenging to generate correctly.

To summarize, the main contributions of this paper are two-fold:

- We propose a new trajectory data synthesis method PrivTrace. Its key insight is to exploit both the first-order and second-order Markov chain models. PrivTrace is built with a new method to choose between the first-order and second-order transition information for next-step prediction, and a new method to estimate the trip distribution from the first-order Markov chain model without consuming extra privacy budget.
- We conduct extensive experiments on both synthetic and real-world trajectory datasets to validate the effectiveness of PrivTrace. Our code is open-sourced at `https://github.com/DpTrace/PrivTrace`.

Roadmap. In Section 2, we give the definition of the problem and present background knowledge of DP. Then we show the framework of trajectory data generation Section 3. Following this framework, we provide the design of our method in Section 4. The experimental results are presented in Section 5 with discussions in Section 6. Finally, we discuss related work in Section 7 and provide concluding remarks in Section 8.

2 Preliminaries

2.1 Problem Definition

In this paper, we consider a dataset consisting of a set of *trajectories*. Each trajectory is composed of a *sequence of points*. We are interested in the following question: Given a sensitive trajectory dataset $\mathcal{D}_o$, how to generate a synthetic trajectory dataset $\mathcal{D}_s$ that shares similar properties with $\mathcal{D}_o$ while satisfying DP. Generating the synthetic trajectory dataset facilitates down-stream data analysis tasks without modifications.

Following prior work [25], we use four statistical metrics to measure the similarity between $\mathcal{D}_s$ and $\mathcal{D}_o$: length distribution, diameter distribution, trajectory density, and transition pattern. The trajectory length is the total distance of a trajectory, which can be used to study the commute distance of people. The trajectory diameter measures the largest Euclidean distance between any two points in a trajectory, which gives information about the range of the individual's activities. The *length distribution* and *diameter distribution* capture the frequency of different trajectory lengths and trajectory diameters in a trajectory dataset, respectively. We use Jensen-Shannon divergence (JSD) [34] to measure the similarity of distributions between $\mathcal{D}_o$ and $\mathcal{D}_s$. A smaller JSD means the generated dataset $\mathcal{D}_s$ is more similar to the original $\mathcal{D}_o$.

The *trajectory density* calculates the number of trajectories that pass through a given area on the map, which can be a good indicator for urban planning, such as estimating the flow of traffic in a specific area. The *transition pattern* captures the frequency of transiting from one place to another, which can help solve problems like next location prediction. We use the average relative error (ARE) of different randomly generated queries of trajectory density and transition pattern to measure their similarity between $\mathcal{D}_o$ and $\mathcal{D}_s$. A smaller ARE implies $\mathcal{D}_s$ is more similar to $\mathcal{D}_o$.

2.2 Markov Chain Model

To generate a synthetic dataset, a commonly used method is the Markov chain model. A Markov chain model is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous events. It is frequently used to analyze sequential data such as location trajectories [5, 15, 22, 24, 35] and natural language [19, 31, 38, 44]. Formally, the Markov chain model is defined as follows:

Definition 1 (Markov Chain Model)**.** *Given a finite set of discrete states* $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_k\}$*, a sequence* $T = (\lambda_1, \lambda_2, \lambda_3, \ldots \lambda_s)$ *is said to follow a kth-order Markov process if* $k \leq i \leq s-1, \forall \lambda \in \Sigma$

$$\Pr[\lambda_{i+1} = \sigma_j | \lambda_i \ldots \lambda_1] = \Pr[\lambda_{i+1} = \sigma_j | \lambda_i \ldots \lambda_{i-k}]$$

where $\Pr[\lambda_{i+1}|\lambda_i \ldots \lambda_1]$ *is called the transition probability from* $\lambda_i \ldots \lambda_1$ *to* λ_{i+1}*.*

Given a dataset $\mathcal{D}$, if r is a subsequence of any $T \in \mathcal{D}$, the empirical transition probability $\Pr[\sigma|r]$ is defined as

$$\Pr[\sigma|r] = \frac{\sum\limits_{\forall T \in \mathcal{D}} N_T(r\sigma)}{\sum\limits_{\forall T \in \mathcal{D}} \sum\limits_{\forall x \in \Sigma} N_T(rx)} = \frac{N_{\mathcal{D}}(r\sigma)}{\sum\limits_{\forall x \in \Sigma} N_{\mathcal{D}}(rx)} \tag{1}$$

Here rx is a sequence where r is followed by x. $N_T(rx)$ is the total number of occurrences of rx in T, and we denote $\sum\limits_{\forall T \in \mathcal{D}} N_T(rx)$ as $N_{\mathcal{D}}(rx)$.

The kth-order Markov chain model is the Markov chain model that can provide $\Pr[\sigma|r]$ for any state σ and any subsequence r with length k. By calculating all $N_{\mathcal{D}}(rx)$ for all possible length-k subsequence r and all $x \in \Sigma$, we can learn a kth-order Markov chain model from the dataset.

2.3 Differential Privacy

Definition 2 (ε-Differential Privacy)**.** *An algorithm* $\mathcal{A}$ *satisfies* ε*-differential privacy (*ε*-DP), where* $\varepsilon > 0$*, if and only if for any two neighboring datasets* $\mathcal{D}$ *and* $\mathcal{D}'$*, we have*

$$\forall O \subseteq Range(\mathcal{A}) : \Pr[\mathcal{A}(\mathcal{D}) \in O] \leq e^{\varepsilon} \Pr\left[\mathcal{A}(\mathcal{D}') \in O\right],$$

where $Range(\mathcal{A})$ *denotes the set of all possible outputs of the algorithm* $\mathcal{A}$*.*

In this paper, we consider two datasets $\mathcal{D}$ and $\mathcal{D}'$ to be neighbors, denoted as $\mathcal{D} \simeq \mathcal{D}'$ if and only if $\mathcal{D} = \mathcal{D}' + S$ or $\mathcal{D}' = \mathcal{D} + S$, where $\mathcal{D} + S$ denotes the datasets resulted from adding one trajectory S to the dataset $\mathcal{D}$.

Laplacian Mechanism. The Laplace mechanism computes a function f on input dataset $\mathcal{D}$ while satisfying ε-DP, by adding to $f(\mathcal{D})$ a random noise. The magnitude of the noise depends on GS_f, the *global* L_1 *sensitivity* of f, defined as,

$$\mathsf{GS}_f = \max_{\mathcal{D} \simeq \mathcal{D}'} ||f(\mathcal{D}) - f(\mathcal{D}')||_1$$

When f outputs a single element, such a mechanism $\mathcal{A}$ is given below:

$$\mathcal{A}_f(\mathcal{D}) \quad = f(\mathcal{D}) + \mathcal{L}\left(\frac{\mathsf{GS}_f}{\varepsilon}\right)$$

where $\mathcal{L}(\beta)$ denotes a random variable sampled from the Laplace distribution with scale parameter β such that $\Pr[\mathcal{L}(\beta) = x] = \frac{1}{2\beta} e^{-|x|/\beta}$.

When f outputs a vector, $\mathcal{A}$ adds independent samples of $\mathcal{L}(\mathsf{GS}_f/\varepsilon)$ to each element of the vector. The variance of each such sample is $2\mathsf{GS}_f^2/\varepsilon^2$.

2.4 Composition Properties of DP

The following composition properties are commonly used for building complex differentially private algorithms from simpler subroutines.

Sequential Composition. Combining multiple subroutines that satisfy DP for $\varepsilon_1, \cdots, \varepsilon_k$ results in a mechanism that satisfies ε-DP for $\varepsilon = \sum_i \varepsilon_i$.

Post-processing. Given an ε-DP algorithm $\mathcal{A}$, releasing $g(\mathcal{A}(\mathcal{D}))$ for any g still satisfies ε-DP. That is, post-processing an output of a differentially private algorithm does not incur any additional privacy concerns.

3 Existing Solutions

Before diving into the descriptions of existing methods that use Markov chain model to generate synthetic trajectories, we first present a general recipe that we observe in all these solutions.

3.1 A Framework for Markov-based Trajectory Synthesis

To generate synthetic trajectories using the Markov chain model, there are three major components: Geographical space discretization, model learning, and trajectory generation. We integrate all the components in a general framework:

- **Discretization.** The purpose of discretization is to create discrete states for the Markov chain model. We discretize the continuous geographical space into one or more *grids* where each grid partitions the geographical space. After discretization, each area in the grid is regarded as a state in the Markov chain model.
- **Model Learning.** Given a set of geographical states and a trajectory dataset, we need to learn some models that can capture the transition pattern of the trajectory dataset. Here, the model can be a Markov chain model and other information (e.g., trip distribution) extracted from the dataset. To train the Markov chain model, we calculate all $N_{\mathcal{D}}$ needed in Equation 1 and then add noise to achieve DP.
- **Generation.** After the model is learned, we can generate the synthetic trajectory (e.g., by random walk on the model). The trajectory generation component is a post-processing step in the context of DP, thus does not have an additional privacy concern.

Table 1: Summary of existing methods on different steps.

Method \ Step	Discretization	Model Learning	Generation
AdaTrace [25]	Adaptive partition	First-order Markov chain model	Markov sampling + auxiliary info
DPT [28]	Reference system	Multiple Prefix Trees	Markov sampling
PrivTrace	Adaptive partition	First-/second-order Markov chain model	Random walk

3.2 Existing Methods

Table 1 summarizes these three phases of existing work. In what follows, we review these steps for previous studies.

AdaTrace. This method extracts the first-order Markov chain model, trip distribution, and length features from the sensitive dataset with DP, and then generates private trajectories according to these features. In discretization, AdaTrace proposes to discretize the geographical space into grids twice with two different granularities. It first uniformly discretizes the geographical space into a coarse-grained grid. For the cells in the first grid with a large number of trajectories passing through, AdaTrace further discretizes them into finer-grained grids.

In model learning, AdaTrace learns a first-order Markov chain model using the first grid. To better capture the inherent patterns of the dataset, AdaTrace also extracts the length distribution and trip distribution (count of pairs of starting and ending point in the trajectory). Noise is added to achieve DP.

In generation, AdaTrace first samples a starting state and an ending state from the trip distribution, and samples a trajectory length $|T|$ from the length distribution. Then, AdaTrace generates the trajectory from the starting state, and repeatedly generates the next state by random walking on Markov chain model. After $|T|-1$ steps, the trajectory directly jumps to the ending state. Finally, for each generated state, AdaTrace uniformly samples a location point from the corresponding cell. If the cell is further partitioned, the location point is sampled according to the density of the second-layer grid.

DPT. In order to capture more precise information, DPT uses multiple uniform grids with different granularities. When discretizing trajectories, DPT will choose the most coarse-grained grid for which there exists a transition for two consecutive points of the trajectory (i.e., using a more coarse-grained grid, the two locations in the trajectory will be in the same cell). A trajectory will be transformed into multiple segments of transition, where transitions in the same segment are on the same grid.

Now for each grid, DPT establishes a differentially private prefix tree [13] to model the transitions using the transition segments. A trajectory will be generated following the path on the prefix tree. Note that there are also transitions to allow a state in one prefix tree to move to another prefix tree with different granularity.

4 Our Proposal

Both AdaTrace and DPT adopt Markov chain model to capture the transition pattern of the trajectory dataset. But they work in two extremes: AdaTrace mostly uses the first-order Markov chain model to reduce the amount of noise. The drawback is that the first-order Markov chain model only captures a limited amount of information, and thus the result is not accurate. On the other hand, DPT uses high-order Markov chain model, but this leads to excessive noise being added, and also makes the result inaccurate.

Intuitively, there is a trade-off between the amount of information we can extract and the amount of noise we have to add due to the constraint of DP. Our insight is to work in the middle ground between the two extremes of AdaTrace and DPT, where we use the Markov chain model in a way that is neither too coarse-grained nor too fine-grained. In particular, we employ the first-order and second-order Markov chain models and select useful information in the two models for generation. To this end, we propose PrivTrace, which achieves a good trade-off between accuracy and noise.

4.1 Method Overview

PrivTrace follows the general framework for generating synthetic trajectory data described in Section 3, which consists of three major parts: geographical space discretization, model learning, and trajectory generation.

Step 1: Discretization. To better capture the transition information of the trajectories, we rely on the observation that the places with more trajectories passing through should be discretized in a finer-grained manner. To this end, we use a density-sensitive two-layer discretization scheme. The core idea is to first discretize the map into a coarser-grained uniform grid (or first-layer grid), and the cells in the first-layer grid with many trajectories passing through are further discretized into finer-grained (second-layer) grid. The details of this step are in Section 4.2.

Step 2: Model Learning. Our model learning step contains two components: the Markov chain models learning and trip distribution estimation.

Markov Chain Models Learning. We first estimate the first-order and second-order models in a differentially private manner. The details of this component are discussed in Section 4.3.

Trip Distribution Estimation. We estimate the trip distribu-

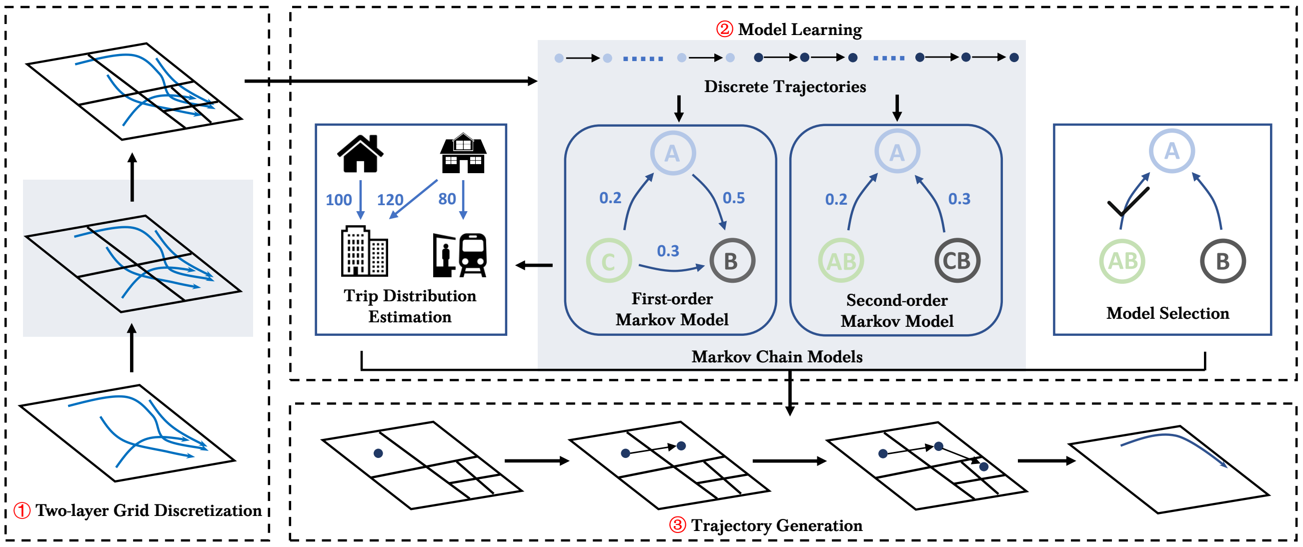

Figure 1: Method overview. PrivTrace is composed of three parts: discretization, model learning, and trajectory generation. The discretization step first partitions the space into a coarse-grained uniform grid, and then the density of the cells with DP to determine which cells need to be expanded. For model learning, we learn both the first-order and second-order Markov chain models with differential privacy, and obtain the trip distribution from the two models. The trajectory generation process is a random-walk-based algorithm to generate a synthetic trajectory. We propose a method to select from the two models during the generation of synthetic trajectories.

tion to obtain the frequency of the starting and ending points of the trajectories. The Markov chain model already contains this, but it is biased due to a normalization step when training the Markov chain model. To this end, we propose a method to obtain an accurate estimation of the distribution estimation. The details of this component are referred to Section 4.4.

Step 3: Generation. We use a random-walk-based method to generate trajectories, which starts at a random state and predict the next state by using either first-order or second-order Markov chain model. Concretely, we first sample a pair of starting and ending states from the estimated trip distribution and random walk with the two Markov chain models. The details of this step are referred to Section 4.5.

Note that we use two Markov chain models to predict the next step in the synthetic trajectory. In the prediction process, the first-order and second-order Markov chain models have different prediction abilities. To take advantage of both two models, we propose two criteria to choose between them, which will be described in Section 4.6.

4.2 Geographical Space Discretization

We borrow the idea of Qardaji et al. [40] to discretize the geographical space. In particular, we first divide the geographical space into $K \times K$ cells (we call them the first-layer cells) of equal size. For each cell, we further calculate the number of trajectories that passes through this cell, and if the number is large, we further partition it.

To estimate the occurrences of trajectories for all first-layer cells, we use the Laplacian mechanism with parameter ε_1. Here one challenge is that the sensitivity is unbounded, since a trajectory can have an arbitrary number of occurrences in cells. This introduces infinite DP noise. To address this issue, we normalize the trajectory (the normalization method introduced in [25]) to bound the sensitivity (proved in Appendix B.1). For example, in Figure 1, we first discretize the space into four cells C_1, C_2, C_3, and C_4. Consider a trajectory that occurs in C_1 and C_2, its total number of occurrences in all cells is 2. After we normalize the occurrence of the trajectory in C_1 and C_2 by the total occurrence, the trajectory contributes to the trajectory occurrence in C_1 and C_2 by $1/2$ and $1/2$ instead of 1 and 1.

If a cell has many trajectories passing over it, we expand it using a more fine-grained cell so that we can understand that area with more details. In Figure 1, step 1, the right bottom cell is expanded into four cells.

Difference from AdaTrace. AdaTrace [25] adopts a similar two-layer discretization scheme as PrivTrace; however, the use of two-layer grids is different. In AdaTrace, the cells in the first layer are used as states for Markov chain model, while the cells in the second layer are used for sampling in the generation phase. On the other hand, in PrivTrace, the cells from both the first-layer grid (if not divided again in the second layer) and the second-layer grid are used as states. The advantage of using cells from all layers as states is that it can capture finer-grained transition information.

4.3 Markov Chain Models Learning

After the two-layered discretization, the whole space is split into disjoint areas, denoted by $\{C_i\}$. To learn the Markov chain models, we calculate the counts of each possible transition P. As our goal is to train the first-order and second-order Markov models, we only consider P of length 2 or 3. Note that for the purpose of sampling, there should be a virtual starting and an ending states in the Markov models. To incorporate this, we augment each trajectory with a start and an end state.

Similar to the case of occurrence estimation in discretization, here the sensitivity of the transition counts is also unbounded (since a sequence can be long and lead to a large change in transition counts), and we also use a length normalization method to bound the sensitivity (by dividing the counts by the length of the trajectories) to 1. Concretely, denoting the occurrence of transition P in the trajectory T as $N_T(P)$, the transition count of P in the Markov chain model is defined as the length-normalized value $\sum_{\forall T \in \mathcal{D}} \frac{N_T(P)}{|T|}$. Using such a definition, every trajectory contributes to the transition count of P in the Markov chain model by $\frac{N_T(P)}{|T|}$, which is at most 1. Therefore, the change of one trajectory will change the counts in the Markov chain model by at most 1 (see details in Appendix B.2). We then add Laplace noise to the transition counts (we spend ε_2 and ε_3 for the length-2 and length-3 transitions, respectively) and use the noisy transition counts to build the Markov chain models.

Note that adding noise might make the transition count negative, which makes the sampling of trajectory infeasible. To deal with this issue, we adopt the postprocessing method NormCut [49, 57] to handle the negative values. We refer the readers to Appendix B.6 of [46] for the details of NormCut.

4.4 Trip Distribution Estimation

The normalization operations in Section 4.3 introduce bias to the transition counts (intuitively, we over-count short trajectories and under-count long trajectories because they have equal weights after normalization) used in the Markov models. In this subsection, we propose a novel method to reduce bias. The key idea is to leverage the spacial structure and the assumption that people tend to follow the shortest path when traveling. Therefore, for trajectories starting from location i and ending at j, the normalization factor is the length of the shortest path from i to j. Building on this assumption, we estimate the distribution of trips between any pair of i and j.

Specifically, denote t_{ij} as the number of trajectories from i to j, such that $\sum_{i,j=1, i\neq j}^{m} t_{i,j}$ equals the number of trajectories in the original dataset. From the Markov model, we know the normalized transition counts from the virtual starting point to any first location i, denoted by b_i, and from any last location j to the virtual ending point, denoted by q_j. Moreover, we know the shortest paths from i to j, denoted by l_{ij}. Ideally, we have

$$b_i \simeq \sum_{j=1}^{m} \frac{t_{ij}}{l_{ij}} \quad \text{and} \quad q_j \simeq \sum_{i=1}^{m} \frac{t_{ij}}{l_{ij}}$$

which captures the intuition that the number of normalized trajectories starting from i should equal the summation (over the ending location j) of all normalized trajectories starting from i, and similarly for the normalized trajectories ending at j. Now we have $2m$ approximate equations with m^2 unknown variables (the t_{ij}'s). We cannot directly solve the unknowns. Instead, we build an optimization problem to estimate approximate values for them. In particular, we use existing solvers to find t_{ij}'s that can minimize the following quantity:

$$\begin{aligned} \min_{t_{i,j}} \quad & \sum_{i=1}^{m}\left(\sum_{j=1}^{m}\frac{t_{ij}}{l_{ij}} - b_i\right)^2 + \sum_{j=1}^{m}\left(\sum_{i=1}^{m}\frac{t_{ij}}{l_{ij}} - q_j\right)^2 \\ s.t. \quad & \sum_{i,j=1}^{m} t_{i,j} = |\mathcal{D}| \\ & t_{i,j} \geq 0, \qquad i,j = 1,2,3,\ldots,m \end{aligned}$$

More details of the optimization problem are given in Appendix B.3. Given the trip distribution, next, we will then describe how to use it in the final generation process. We empirically show in Section 5.3 that using the trip distribution estimated by our method can achieve better accuracy than that of directly extracting from the original dataset, especially when the privacy budget is small.

4.5 Trajectory Generation

Our trajectory generation algorithm relies on random walking on the first- or second-order Markov models. The workflow of the trajectory generation algorithm is illustrated in Algorithm 1. It takes as input the first-order and second-order Markov chain models M_1 and M_2, and the trip distribution $\{t_{ij}\}$, and works in three steps as follows:

Step 1: Initialization. We first set T_{state} as an empty sequence of states. Then a start-end state pair (λ_{start}, λ_{end}) is sampled from the trip distribution $\{t_{ij}\}$. The predicted real end state λ_{end} will not be used in the generation process. We estimate it mainly to make the optimization problem in Section 4.4 more accurate. We then set λ_{last} as the λ_{start} and λ_{now} as $M_1(\lambda_{start})$ (Line 4-Line 6).

Step 2: Random Walk. We first add λ_{now} to the end of T_{state}. Then the next state prediction is conducted. Before predicting the next step, we select a proper model to use. The model selection method will be explained in Section 4.6. Concretely, if M_2 is the chosen model, then the next state λ_{next} is $M_2(\lambda_{now}, \lambda_{last})$ (Line 18). Otherwise, M_1 is the chosen model, λ_{next} is $M_1(\lambda_{now})$ (Line 20). $M_1(\lambda_{now})$ represents predicting λ_{next} only relying on the *current state* λ_{now}, and $M_2(\lambda_{now}, \lambda_{last}))$ represents predicting λ_{next} relying on both

Algorithm 1: Synthetic Trajectory Generation

```
Input: Noisy first-order and second-order transition
       models $M_1$ and $M_2$, trip distribution $\{t_{ij}\}$;
Output: Synthesis trajectory Dataset $\mathcal{D}_s$;
1  Set $\mathcal{D}_s$ as an empty set;
2  for $|\mathcal{D}_s| \leq n_{syn}$ do
3      Step 1: Initialization
4      Set $T_{state}$ as an empty sequence of state;
5      Sample state $\lambda_{start}$ from the trip distribution $\{t_{ij}\}$;
6      $\lambda_{last} \leftarrow \lambda_{start}$; $\lambda_{now} \leftarrow M_1(\lambda_{start})$ ;
7      Step 2: Random Walk
8      Add $\lambda_{start}$ to $T_{state}$;
9      while $\lambda_{now}$ is not virtual end do
10         $T_{state} \leftarrow T_{state}\lambda_{now}$;
11         Step 2-1: Model Selection.
12         if $N_{now,sum} < \theta_1$ or $N_{now,1}/N_{now,2} \geq \theta_2$ then
13             $M_1$ is selected
14         else
15             $M_2$ is selected
16         Step 2-2: Model Prediction.
17         if $M_2$ is selected then
18             $\lambda_{next} \leftarrow M_2(\lambda_{now}, \lambda_{last})$;
19         else
20             $\lambda_{next} \leftarrow M_1(\lambda_{now})$;
21         $\lambda_{last} \leftarrow \lambda_{now}$; $\lambda_{now} \leftarrow \lambda_{next}$;
22     Step 3: Location Point Sampling
23     $T_{syn} \leftarrow$ sampling a location for every state in $T_{state}$;

24     $\mathcal{D}_s \leftarrow \mathcal{D}_s \cup \{T_{syn}\}$;
```

the current state λ_{now} and the *previous state* λ_{last}. After acquiring λ_{next}, we set λ_{last} as λ_{now}. Then value of λ_{next} is given to λ_{now}. The above process is repeated until λ_{now} is the virtual end.

Step 3: Location Point Sampling. After the random walk process, we obtain the discrete version trajectory T_{state}. It is a state sequence. We first set the synthetic trajectory sequence T_{syn} as an empty sequence. Then we sample locations for all states in T_{state} and add all these locations into T_{syn} (Line 23). Specifically, for every state, we sample a location from the geographical area corresponding to it uniformly. T_{syn} is the output synthetic trajectory.

4.6 Markov Chain Models Selection

Given M_1 and M_2, one core question in random walk is how to choose between them in the trajectory generation process. Without loss of generality, supposing we have already generated i states $(\lambda_1, \ldots, \lambda_i)$, we need to determine whether to predict the *next state* λ_{i+1} relying on the first-order model $M_1(\lambda_i)$ or relying on the second-order model $M_2(\lambda_{i-1}, \lambda_i)$.

Selection Rationales. To determine which model to select, we need to consider two important factors: One is the noise error introduced by the Laplacian noise, and the other is the prediction confidence of the two models. Intuitively, if the noise has a significant impact on both models, we select the model with less noise. Otherwise, we select the model with higher prediction confidence.

Selection Principles. The model selection is mainly based on the count of transitions in $\gamma^1(\mathcal{D})$. Given the current state λ_i, the count of transitions from λ_i to any possible state is denoted as $N_i = \{N_{\mathcal{D}}(\lambda_i\lambda_j), \forall \lambda_j \in \Sigma\}$. For the sake of brevity, we denote the largest and the second largest count value in N_i as $N_{i,1}$ and $N_{i,2}$, and the sum of all counts as $N_{i,sum}$. We analyze the advantages and disadvantages of M_1 and M_2 from two aspects:

1. **Count vs. Noise.** If $N_{i,sum}$ is smaller than a threshold θ_1, we use M_1 as the model to predict the next step; otherwise we will consider the next principle.

   The reason for this rule is that noise has a larger impact on the second-order model than the first-order model since the true counts in the second-order model are always smaller than or equal to those in the first-order model. Thus, the first-order model has a larger signal-to-noise ratio, and will more likely to bring better performance when the counts are small.

2. **Existence of Dominant State.** If the counts are not too small, we will compare $\frac{N_{i,1}}{N_{i,2}}$ to a threshold θ_2. If $\frac{N_{i,1}}{N_{i,2}} \geq \theta_2$, we use M_1 as the model to predict the next step, otherwise we will use M_2.

   Here, we use the term $\frac{N_{i,1}}{N_{i,2}}$ as an indicator of whether there is a dominant state. If $\frac{N_{i,1}}{N_{i,2}} \geq \theta_2$, the state corresponds to $N_{i,1}$ is much more likely to be chosen as the next state than any other states in M_1.

In summary, when the state count is large enough ($N_{i,sum}\theta_1$) and the largest count ($N_{i,1}$) is not dominant, we prefer to choose M_2; otherwise, we prefer to choose M_1.

The values of θ_1 and θ_2 are hyperparameters, which we will discuss and verify in the evaluation (see Appendix F of [46]).

4.7 Algorithm Analysis

DP Guarantee. Here we show that PrivTrace theoretically satisfies ε-DP.

Theorem 1. PrivTrace *satisfies* ε*-differential privacy, where* $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$.

We refer the readers to Appendix C for the detailed proof of Theorem 1.

Computational Complexity Analysis. Due to space limitation, we refer the readers to Appendix A for the detailed time and space complexity analysis of PrivTrace, AdaTrace, and DPT. We also empirically evaluate the time and space consumption of different methods. We also discuss the practical time consumption in Section 6.

Table 2: Dataset Statistics.

Dataset	Type	Area	Scale
Brinkhoff	Synthetic	California	30,000
Taxi	Real	Porto	200,000
Geolife	Real	Beijing	17,621

5 Evaluation

In this section, we first conduct an end-to-end experiment to illustrate the superiority of PrivTrace over the state-of-the-arts. We then conduct comprehensive ablation studies to illustrate the effectiveness of different components of PrivTrace. Finally, we compare the utility of the synthetic trajectories with different lengths.

5.1 Experimental Setup

Datasets. We run experiments on one synthetic and two real-world trajectory datasets. The statistics of the datasets are summarized in Table 2.

- **Brinkhoff [9].** Brinkhoff is a popular network-based trajectory generator in the field of traffic research. Given the road network of a certain area, it can generate the trajectories residing in the roads. We use the road network data of California bay area to generate trajectories.
- **Taxi [37].** It contains more than 1.7 million trajectories from 442 taxis in Porto, Portugal. We randomly select 200,000 trajectories to conduct our experiment.
- **Geolife [58].** This dataset is collected from volunteers recruited by Microsoft Research Asia carrying GPS devices. It contains 17,621 trajectories, with a total distance of more than 1.2 million kilometers and a total time of more than 48,000 hours. Most trajectories are in Beijing, China.

Metrics. We adopt four metrics to measure the similarity between the original dataset $\mathcal{D}_o$ and synthetic dataset $\mathcal{D}_s$: Length distribution, diameter distribution, trajectory density, and transition pattern. Due to space limitation, we refer the readers to Appendix B.4 for the detailed description of these metrics.

Note that AdaTrace also uses these metrics in their experiments; however, our parameter settings are different, which might lead to inconsistent values for the AdaTrace method with those reported in their paper. Our parameter settings aim for more fine-grained information. For the length and diameter distribution metrics, AdaTrace uses 20 bins to obtain the distribution while we use 50 bins. For the trajectory density metric, AdaTrace considers the query areas with a constant radius, while we consider random radii. For the transition pattern metric, AdaTrace uses a 6×6 uniform grid to obtain the transition patterns, while we use a 20×20 uniform grid.

Competitors. We compare PrivTrace with DPT and AdaTrace discussed in Section 3.2. We use their open-sourced implementations to conduct our experiments, i.e., AdaTrace [26] and DPT [51]. We also use the recommended parameters from their papers. It is worth noting that, after carefully checking the code snippet of the length extraction process of AdaTrace, we find that some steps do not satisfy differential privacy. For a fair comparison, we modify the corresponding code snippets to make them differentially private and run the experiment using the modified code. Due to space limitation, we refer the readers to Appendix B.5 for the modification of the code.

Parameter Settings. PrivTrace has two groups of parameters: discretization parameters K, κ in Section 4.2 and model selection parameters θ_1, θ_2 in Section 4.6.

The choice of K is related to whether the trajectories are uniformly or concentratedly distributed on the map and the number of trajectories. A dataset with more concentratedly distributed trajectories should have fewer first-layer cells to reduce states which are unrelated to trajectories. Therefore, K is set as $(|\mathcal{D}|/c)^{\frac{1}{2}}$, where c is a parameter related to the distribution of trajectories. We set c as 5000, 1200, and 500 for Brinkhoff, Taxi, and Geolife, respectively. The corresponding K are 3, 13, 6. One the other hand, κ is calculated by $(d_i \times K \times pop/2 \times 10^7)^{\frac{1}{2}}$, where pop is the population of the area where the trajectories are residing in. We obtain the information of pop from the Internet. We verify the effectiveness of the parameter setting of K and κ in Appendix E of [46].

For θ_1, the standard deviation of the noise added to the transition counts of each state is $(\frac{\sqrt{2}}{\varepsilon_2}) \cdot m$, where $\frac{\sqrt{2}}{\varepsilon_2}$ is the standard deviation of Laplace noise, m is the total number of states. Therefore, we set θ_1 to $(\frac{\sqrt{2}}{\varepsilon_2}) \cdot m$ to choose states with transition counts comparable to noise. On the other hand, θ_2 is used for choosing states with dominant transition states. We empirically find that setting θ_2 as 5 works well since we believe when the largest transition count is 5 times the second-largest transition count, the transition state with the largest count dominates other transition states. We verify the effectiveness of these parameter settings Appendix F of [46].

For AdaTrace and DPT, we use their default parameter settings in experiments.

Implementation. We use different ε in our experiments, ranging from 0.2 to 2.0. We set $\varepsilon_1 = 0.2\varepsilon, \varepsilon_2 = 0.4\varepsilon$, and $\varepsilon_3 = 0.4\varepsilon$ (see Appendix G of [46] for the empirical results of the effectiveness of this privacy budget allocation). Each experiment is repeated 10 times with mean and standard deviation reported.

We implement PrivTrace with Python 3.6 and NumPy 1.19.1. All experiments are run on an Intel E5-2680 server with 128 GB memory and Ubuntu 20.04 LTS system.

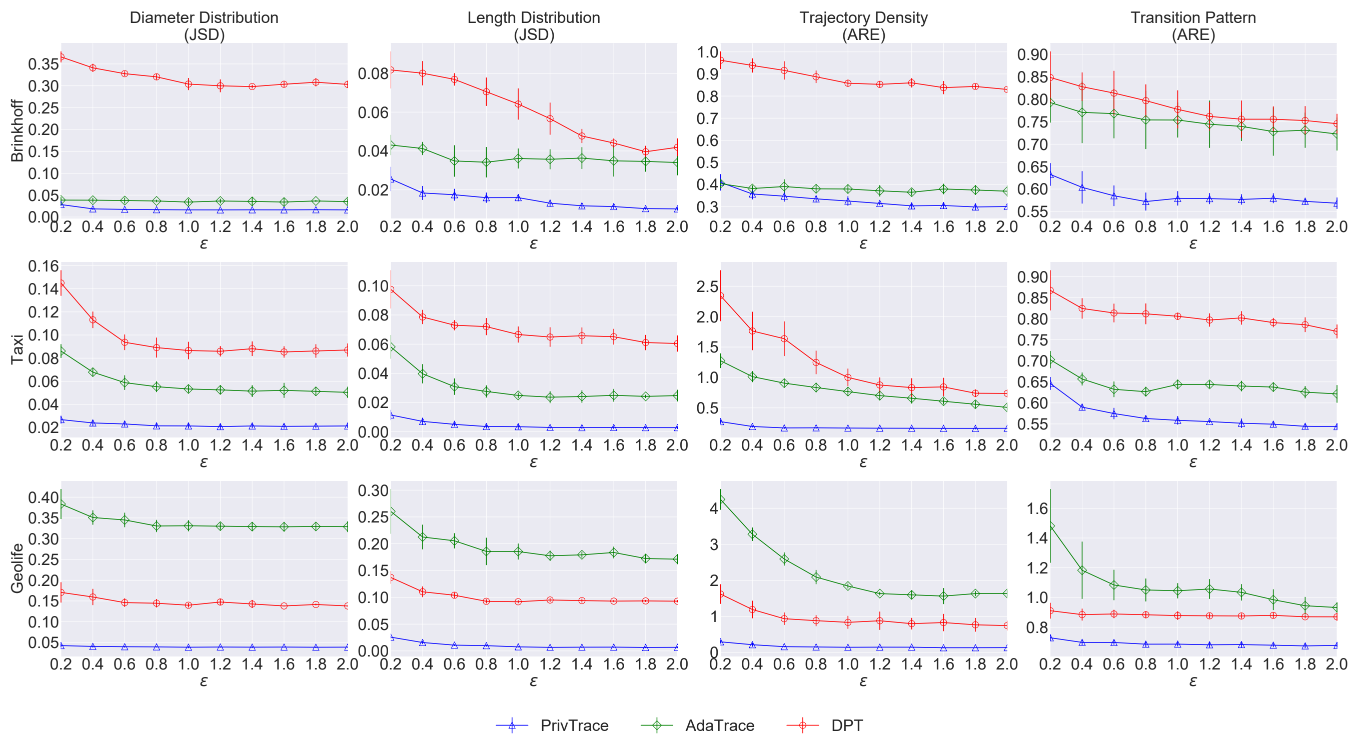

Figure 2: End-to-end comparison of different trajectory synthesis methods. Rows and columns stand for different datasets and different metrics respectively. In each subfig, the y-axis stands for the error of the generated trajectories, and the x-axis stands for the privacy budget ε.

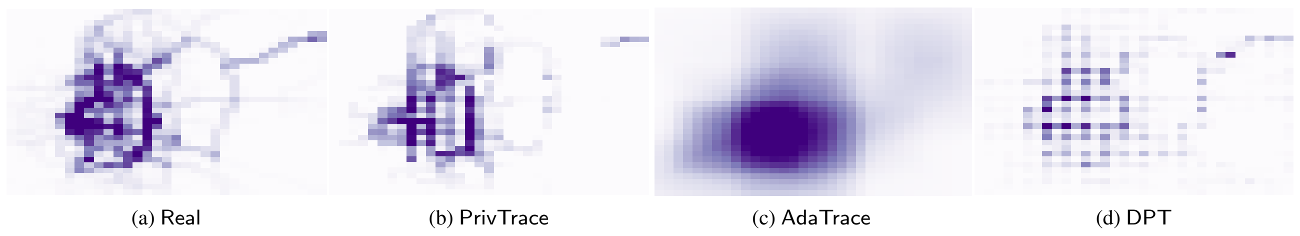

Figure 3: Density visualization for different trajectory synthesis methods on the Taxi dataset.

5.2 End-to-end Evaluation

In this section, we provide an end-to-end evaluation to illustrate the effectiveness of PrivTrace from two perspectives: quantitative evaluation and visualization.

Quantitative Results. We first quantitatively compare PrivTrace with the state-of-the-art methods using the metrics discussed in Section 5.1. Figure 2 illustrates the experimental results on three datasets. In general, we observe that PrivTrace consistently outperforms AdaTrace and DPT for all settings.

For the length and diameter distribution, we observe that PrivTrace reduces the JSD by more than 50% in most of the cases. When the privacy budget is low, PrivTrace can even achieve 1 order of magnitude performance improvement over the state-of-the-art.

With respect to the trajectory density and transition pattern, PrivTrace can also reduce the error by at least 50% in most settings. We attribute this result to the benefit of the two-layer grids in the discretization process in Section 4.2. Note that AdaTrace also uses two-layer discretization. As discussed in Section 4.2, the main difference is that PrivTrace treats the second-layer cells as states while AdaTrace only uses the second-layer grid for sampling. The experimental results further validate that using the second-layer grid as states can capture more transition information, leading to better-quality synthetic trajectories.

Interestingly, comparing AdaTrace and DPT on all datasets, we observe that AdaTrace performs better on the Brinkhoff and Taxi datasets, while DPT performs better on the Geolife dataset. Furthermore, we observe that the abso-

Table 3: Details of ablation studies. In the first study (Figure 4), we evaluate component (a) and considers ablation 1-3; In the second study (Figure 5), we evaluate component (b) and considers ablation 3 and 4; In the last study (Figure 6), we evaluate component (c) and considers ablation 4 and 5.

Ablation study	Component (a)	Component (b)	Component (c)
Ablation1	First-order Markov model	-	-
Ablation2	Second-order Markov model	-	-
Ablation3	Adaptive model	-	-
Ablation4	Adaptive model	Using second-layer cells as state	-
Ablation5	Adaptive model	Using second-layer cells as state	Trip distribution estimation

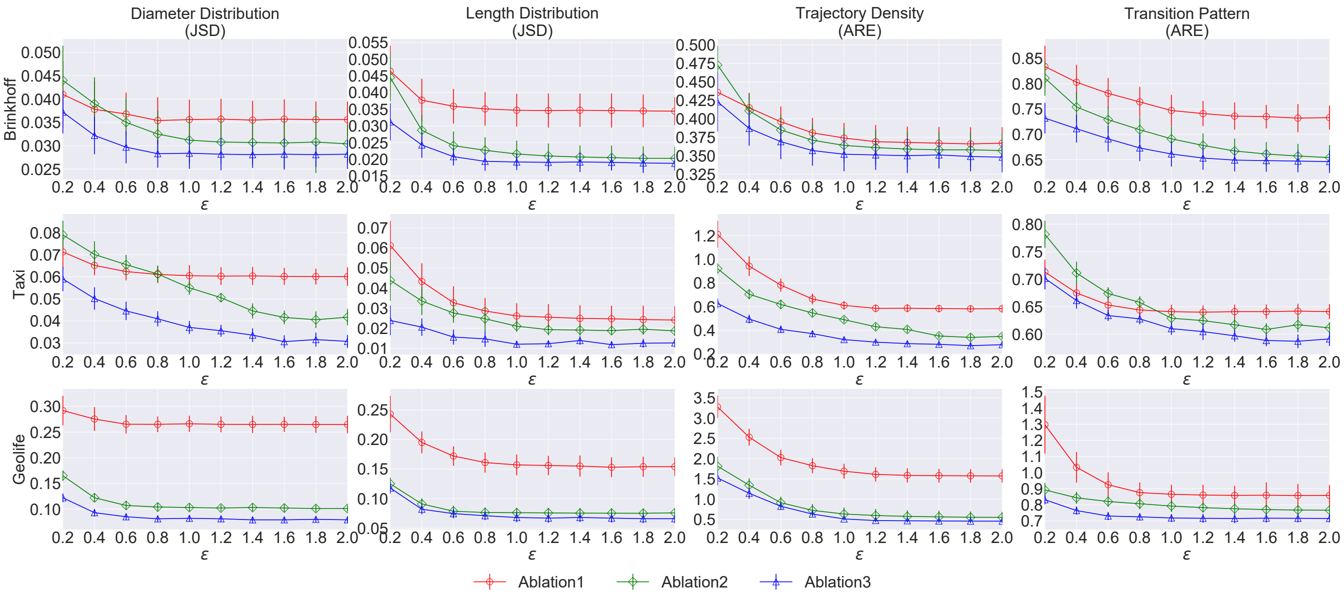

Figure 4: Effectiveness of component (a). Ablation1 stands for the first-order model, Ablation2 stands for PrivTrace with the second-order model, and Ablation3 stands for PrivTrace with the adaptive model. All three methods are without components (b) and (c).

lute error values of DPT are similar for all datasets, while AdaTrace performs extremely poorly on Geolife. After carefully checking the characteristics of all datasets, we find that the trajectory is concentrated in a small area on Geolife, while the trajectories in the other two datasets are more evenly distributed. This observation indicates that AdaTrace is not good at handling the datasets with trajectories concentrating in a small area on the map. This can be explained by the fact that AdaTrace only uses the first-layer cells as states, making most of the trajectories discretized into very short sequences of states. Furthermore, since most of the trajectories are concentrated in a small number of cells, the transition patterns of trajectories lack diversity. In this case, the Markov chain model and the trip distribution cannot learn useful information.

Visualization Results. To better illustrate the superiority of our proposed method PrivTrace, we further provide a visualization comparison of the trajectory density in Figure 3. We experiment on the Taxi dataset and divide the map into an 80×80 uniform grid. We use heatmaps to visualize the number of trajectories passing through each cell in the grid. Darker cells mean there are more trajectories passing through them. The visualization results show that the trajectories generated by PrivTrace have higher fidelity, while AdaTrace and DPT cannot preserve the density information.

5.3 Ablation Study

There are three main components in PrivTrace: (a) the adaptive Markov models (b) taking the second-layer cells as states, and (c) the trip distribution estimation. We conduct three ablation studies to evaluate the effectiveness of each of them in an incremental way, i.e., we first evaluate the effectiveness of (a), and then with the best option for (a), we evaluate (b), and then (c).

Effectiveness of the Adaptive Markov Models. First, we verify the effectiveness of the adaptive model with three variants called Ablation1, Ablation2, and Ablation3, which stand for three versions of PrivTrace differing in component (a). Specifically, Ablation1, Ablation2, and Ablation3 use the first-order Markov model, the second-order Markov model, and

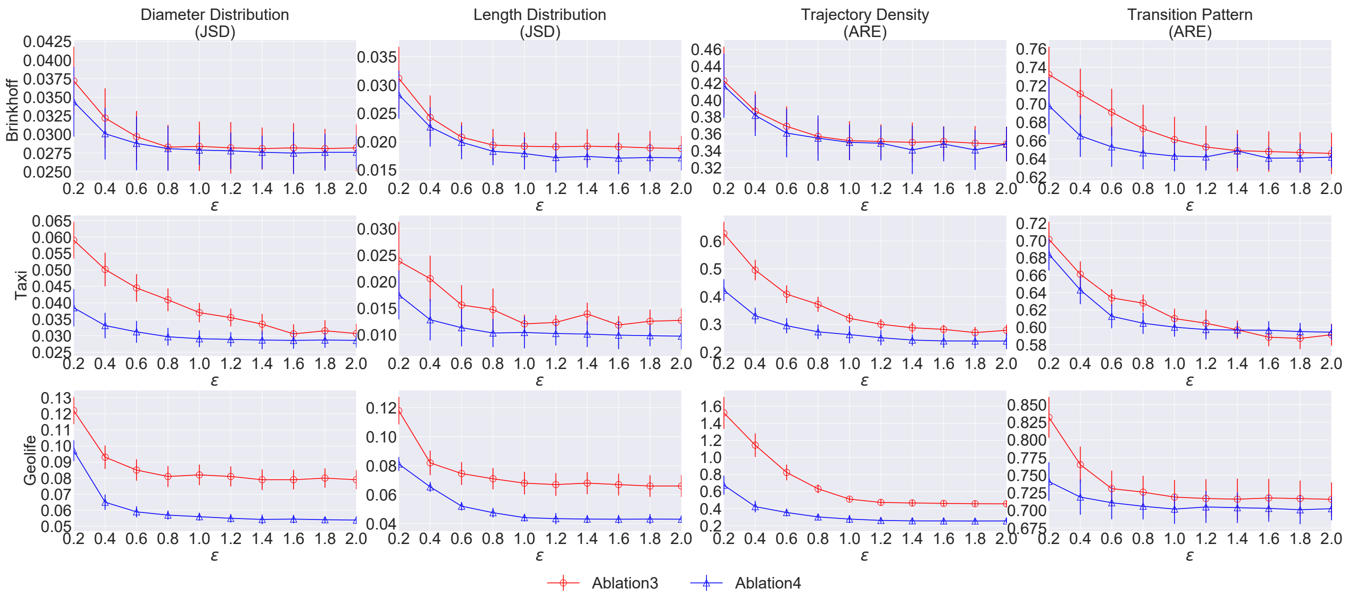

Figure 5: Effectiveness of taking the second layer cells as states of PrivTrace. Ablation3 stands for PrivTrace without second layer cells, and Ablation4 stands for PrivTrace with second layer cells. Both methods assume the adaptive model in component (a) but without component (c).

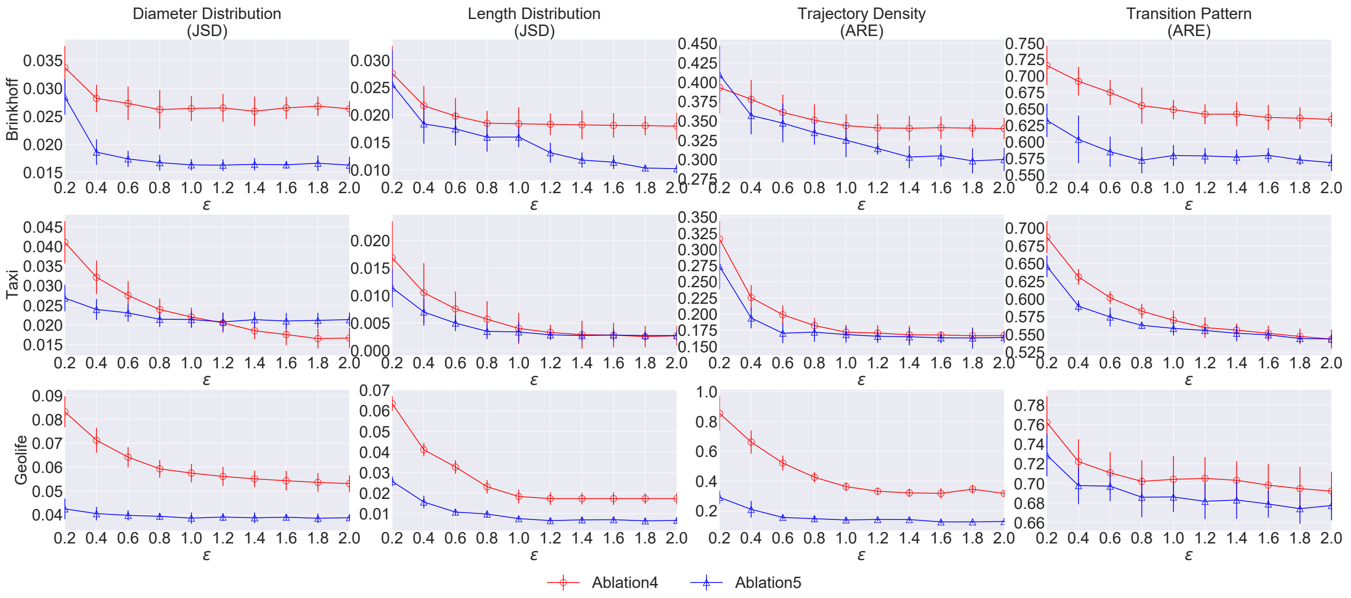

Figure 6: Effectiveness of the trip distribution estimation of PrivTrace. Ablation1 stands for PrivTrace with adaptive model and (b), but without (c). Ablation5 stands for PrivTrace with the adaptive model, (b), and (c).

the adaptive model, respectively. Meanwhile, all Ablation1, Ablation2, and Ablation3 are without (b) and (c). The results in Figure 4 show that Ablation3 outperforms the other two algorithms, which indicates that the adaptive model is the best choice among the three ways of using the Markov models.

Effectiveness of Second-layer States. Next, using the adaptive model in (a), we study the effectiveness of (b): How much do the second-layer cells help? We have two methods, Ablation3 (without the second layer, from the last experiment) and Ablation4 (with the second layer), and the results are shown in Figure 5. The results show that (b) is effective since Ablation4 outperforms Ablation3 for most datasets and metrics.

Effectiveness of Trip Distribution. The last evaluation is for component (c). Similar to the last group, Ablation5 differs from Ablation4 by considering component (c) trip distribution estimation. The results are shown in Figure 6. From the

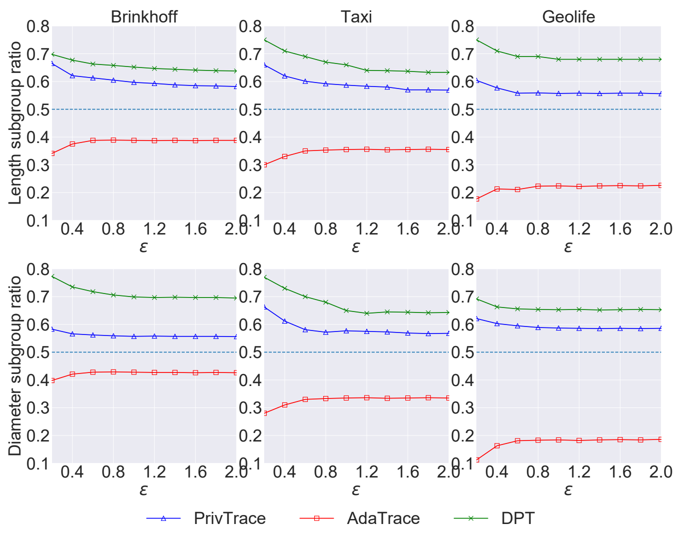

Figure 7: Ratio of subgroup in different generation algorithms for different levels of ε. The lines closer to 0.5 indicates less bias.

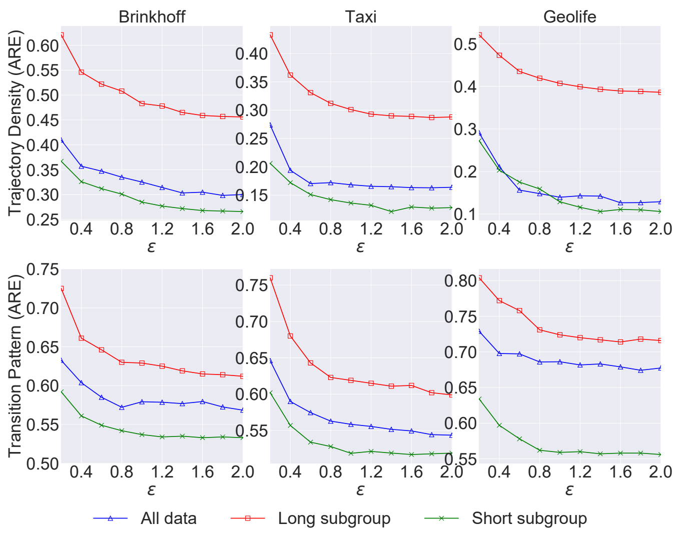

Figure 8: Errors of the long subgroup and the short subgroup on the trajectory density and transition pattern metric.

results, we can conclude that (c) has a contribution to the accuracy since the performance of Ablation5 is better than Ablation4 in general.

For better reference, we include all the ablation variants in Table 3.

5.4 Impact on Subgroups

Georgi et al. [23] focus on the impact of DP on the subgroups, and show that DP can affect the distribution and data utility of subgroups (more explanations about [23] can be found in Appendix D of [46]). Following the settings of [23], we conduct two groups of evaluation to show the impact of DP trajectory data generation algorithms on subgroups.

Subgroup Ratio. First, we evaluate the impact of DP on the subgroup distribution for PrivTrace, AdaTrace, and DPT. In trajectory data, there is no data attribute or other demographic information such as gender that can be used in dividing subgroups directly. Thus, we partition the trajectory data based on two common attributes: length and diameter. We first calculate the median of the trajectory length in the original dataset, and then partition the dataset into the long subgroup $R_{l>l_m}$ and short subgroup $R_{l\leq l_m}$ based on the median value l_m (thus 50% trajectories are long and 50% are short). The diameter subgroups are defined similarly.

We compare the ratio of the short-length subgroup and the ratio of the short-diameter subgroup in the whole dataset for the synthetic data generated by PrivTrace, AdaTrace, and DPT. Figure 7 illustrates the experimental results (each experiment is repeated 10 times). In general, we observe that PrivTrace has the least bias for the ratio of the subgroup.

Subgroup Utility. Second, we evaluate the impact of PrivTrace on the data utility of subgroups. We use the median of length to divide subgroups and get four subgroups: The long subgroup from the original dataset $R_{l>l_m}$, the short subgroup from the original dataset $R_{l\leq l_m}$, the long subgroup from the synthetic dataset $S_{l>l_m}$, and the short subgroup from the original dataset $S_{l\leq l_m}$. We choose two metrics, i.e., the trajectory density metric and the transition pattern metric, to show the data utility. In Section 5.2, these two metrics compare synthetic data with the original data. However, it does not make sense if we compare the data in a subgroup with the whole original data directly. Instead, we compare the subgroup from the original data and the synthetic data (such as $R_{l>l_m}$ and $S_{l>l_m}$), regarding the subgroup from the original dataset as the original data. For a fair comparison, we sample the larger subgroup before comparison to make the two subgroups equal in size.

We conduct the experiments on two metrics and the conclusions for other metrics are consistent. The experimental results in Figure 8 show that the utility of the long subgroup is worse than that of the short subgroup. We suspect this is due to the fact that long trajectories are more complicated and thus are more challenging to generate correctly. The unbalanced utility between different subgroups is a limitation of PrivTrace, as well as the limitation of AdaTrace and DPT (we defer the corresponding results to Appendix D of [46] due to space limitation).

It is worth noting that our evaluations are only for the utility of subgroups, while the privacy-protection level is the same since all subgroups have the same privacy guarantee provided by DP.

6 Discussion

Time Consumption. Theoretical analysis (Table 4 in Appendix A) shows that the time complexity of PrivTrace is

$O(m|\mathcal{D}| + m^3)$, which is larger than AdaTrace ($O(m|\mathcal{D}| + m^2)$). This is a limitation of PrivTrace when m is extremely large. However, our empirical experiments show that the time consumption of PrivTrace on real-world trajectory datasets is acceptable, e.g., less than 10 minutes (see Table 5 in Appendix A for more details).

Practical Implementation. With PrivTrace (or other models), we can ensure the trajectories published/shared are protected by differential privacy. It is possible that given such a system, the data-miner/advertiser may make more predictions. This is not a limitation of the technique; instead, it is a consequence of users being attracted to share more data.

7 Related Work

Location Density Estimation. There are a number of previous studies focusing on estimating the density distribution of a location dataset while satisfying DP. Most approaches rely on recursive partitioning, which recursively performs binary partitioning on the map. Xiao et al. [52] propose to use KD-tree, which recursively partitions the map along some dimensions. In order to minimize the non-uniformity error, the authors use the heuristic to choose the partition point such that the two sub-regions are as close to uniform as possible. To improve the estimation accuracy, Cormode et al. [16] propose several alternatives based on KD-trees. Instead of using a uniformity heuristic, they partition the nodes along the median of the partition dimension. The height of the tree is predetermined, and the privacy budget is divided among the levels. The key challenge of the recursive partitioning-based approaches lies in choosing the right partition granularity to balance the noise error and the non-uniformity error. To address this issue, Qardaji et al. [40] propose a uniform-grid approach, which applies an equal-width grid of a certain size over the data domain and then issues independent count queries on the grid cells.

Trajectory Data Publication. Several studies have been done on finishing certain tasks using trajectory data or publishing trajectory data via perturbing locations. Chen et al. [13] propose to use differentially private prefix tree to publish transition data. They established a method to generate transition data using a differentially private prefix tree. They evaluated its mechanism by studying the impact of its protection on range query and sequential pattern mining. Acs et al. [3] propose a mechanism to publish spatio-temporal trajectory density data, which is counts of active users within small areas for given time windows. Jiang et al. [30] propose to protect ships' trajectories by location perturbing. They preserve the endpoints of trajectories while the intermediate locations are altered by adding some noise satisfying differential privacy guarantees. Also, there are some recent works focusing on generating synthetic trajectory data following other privacy requirements instead of different privacy. One of the most famous works is SGLT [7]. SGLT uses plausible deniability as the privacy requirement. It captures both geographic and semantic features of real traces and uses the real traces are seeds to generate synthetic trajectories.

Our paper focuses on a more general paradigm that publishes a synthetic trajectory dataset while satisfying DP.

Differentially Private Tabular Data Synthesis. For the general tabular data, there exists a number of studies following the same paradigm of our paper, i.e., generating tabular data while satisfying DP. They can be broadly classified into three categories: graphical model-based methods, game-based methods, and deep generative model-based methods.

The graphical model-based methods aim to estimate a graphical model that approximates the distribution of the original dataset in a differentially private manner and generate synthetic data by sampling from it. For instance, PrivBayes [54] and BSG [8] adopt Bayesian network to approximate the data distribution, while PGM [36] and JTree [14] use Markov random field to approximate the data distribution. PrivSyn [57] utilizes a number of marginal tables to capture as much as correlated information in the dataset, which can be regarded as a dense graphical model. The core idea of game-based methods is to formulate the dataset synthesis problem as a zero-sum game [21, 27, 45]. Assume there are two players, a data player and a query player. MWEM [27] method solves the game by having the data player use a no-regret learning algorithm, and the query player repeatedly best responds. Dual Query [21] switches the role of the two players. The deep generative model-based approaches rely on the DP-SGD framework [1] (adding noise in the optimization procedure) to train a generative model, and use this deep generative model to generate a synthetic dataset. The most commonly used deep generative model is the Generative Adversarial Network (GAN) [2, 6, 20, 43, 55].

8 Conclusion

In this paper, we propose a differentially private algorithm to generate trajectory data. By employing the first-order and second-order Markov chain models, we achieve a middle ground between richness of information and amount of noise. Besides, we propose an optimization-based method to estimate the trip distribution, which is important information for generating synthetic trajectory data. Extensive experiments on real-world and generated datasets are conducted to illustrate the superiority over the current state of the art.

Acknowledgments

We thank the anonymous shepherd and reviewers for their constructive feedback. This work is supported by "New Generation Artificial Intelligence" major project of China under Grant 2018AAA0101605, National Natural Science Foundation of China (NSFC) under grant NO.61833015, NSF CNS-2220433, CNS-2213700, CCF-2217071, and the Helmholtz Association within the project "Trustworthy Federated Data Analytics" (TFDA) (funding number ZT-I-OO1 4).

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] Nazmiye Ceren Abay, Yan Zhou, Murat Kantarcioglu, Bhavani Thuraisingham, and Latanya Sweeney. Privacy Preserving Synthetic Data Release Using Deep Learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 510–526. Springer, 2018.

[3] Gergely Acs and Claude Castelluccia. A Case Study: Privacy Preserving Release of Spatio-temporal Density in Paris. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1679–1688, 2014.

[4] Gowtham Atluri, Anuj Karpatne, and Vipin Kumar. Spatio-temporal Data Mining: a Survey of Problems and Methods. *ACM Computing Surveys (CSUR)*, 51(4):1–41, 2018.

[5] Faisal I Bashir, Ashfaq A Khokhar, and Dan Schonfeld. Object Trajectory-based Activity Classification and Recognition Using Hidden Markov Models. *IEEE transactions on Image Processing*, 16(7):1912–1919, 2007.

[6] Brett K Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, Ran Lee, Sanjeev P Bhavnani, James Brian Byrd, and Casey S Greene. Privacy-preserving Generative Deep Neural Networks Support Clinical Data Sharing. *Circulation: Cardiovascular Quality and Outcomes*, 12(7):e005122, 2019.

[7] Vincent Bindschaedler and Reza Shokri. Synthesizing Plausible Privacy-preserving Location Traces. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 546–563. IEEE, 2016.

[8] Vincent Bindschaedler, Reza Shokri, and Carl A Gunter. Plausible Deniability for Privacy-Preserving Data Synthesis. *Proceedings of the VLDB Endowment*, 10(5), 2017.

[9] Thomas Brinkhoff. A Framework for Generating Network-based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.

[10] United States Census Bureau. `https://www.census.gov/programs-surveys/decennial-census/decade/2020/planning-management/process/disclosure-avoidance/differential-privacy.html`.

[11] Shan Chang, Chao Li, Hongzi Zhu, Ting Lu, and Qiang Li. Revealing Privacy Vulnerabilities of Anonymous Trajectories. *IEEE Transactions on Vehicular Technology*, 67(12):12061–12071, 2018.

[12] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. When Machine Unlearning Jeopardize Privacy. In *ACM CCS*, 2021.

[13] Rui Chen, Gergely Acs, and Claude Castelluccia. Differentially Private Sequential Data Publication Via Variable-length N-grams. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 638–649, 2012.

[14] Rui Chen, Qian Xiao, Yu Zhang, and Jianliang Xu. Differentially Private High-dimensional Data Publication via Sampling-based Inference. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 129–138, 2015.

[15] Yihang Cheng, Yuanyuan Qiao, and Jie Yang. An Improved Markov Method for Prediction of User Mobility. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 394–399. IEEE, 2016.

[16] Graham Cormode, Cecilia Procopiuc, Divesh Srivastava, Entong Shen, and Ting Yu. Differentially Private Spatial Decompositions. In *2012 IEEE 28th International Conference on Data Engineering*, pages 20–31. IEEE, 2012.

[17] Linkang Du, Zhikun Zhang, Shaojie Bai, Changchang Liu, Shouling Ji, Peng Cheng, and Jiming Chen. AHEAD: Adaptive Hierarchical Decomposition for Range Query under Local Differential Privacy. In *ACM CCS*, 2021.

[18] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private data Analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[19] Shai Fine, Yoram Singer, and Naftali Tishby. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine learning*, 32(1):41–62, 1998.

[20] Lorenzo Frigerio, Anderson Santana de Oliveira, Laurent Gomez, and Patrick Duverger. Differentially Private Generative Adversarial Networks for Time Series, Continuous, and Discrete Open Data. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 151–164. Springer, 2019.

[21] Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Zhiwei Steven Wu. Dual Query: Practical Private Query Release for High Dimensional Data. In *International Conference on Machine Learning*, pages 1170–1178. PMLR, 2014.

[22] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. Next Place Prediction Using Mobility Markov Chains. In *Proceedings of the first workshop on measurement, privacy, and mobility*, pages 1–6, 2012.

[23] Georgi Ganev, Bristena Oprisanu, and Emiliano De Cristofaro. Robin Hood and Matthew Effects–Differential Privacy Has Disparate Impact on Synthetic Data. *arXiv preprint arXiv:2109.11429*, 2021.

[24] Chong Yang Goh, Justin Dauwels, Nikola Mitrovic, Muhammad Tayyab Asif, Ali Oran, and Patrick Jaillet. Online Map-matching Based on Hidden Markov Model for Real-time Traffic Sensing Applications. In *2012 15th International IEEE Conference on Intelligent Transportation Systems*, pages 776–781. IEEE, 2012.

[25] Mehmet Emre Gursoy, Ling Liu, Stacey Truex, Lei Yu, and Wenqi Wei. Utility-aware Synthesis of Differentially Private and Attack-resilient Location Traces. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 196–211, 2018.

[26] Mehmet Emre Gursoy, Ling Liu, Stacey Truex, Lei Yu, and Wenqi Wei. Adatrace-Github. `https://github.com/git-disl/AdaTrace`, 2019.

[27] Moritz Hardt, Katrina Ligett, and Frank McSherry. A Simple and Practical Algorithm for Differentially Private Data Release. In *Proceedings of the 25th International Conference on Neural Information Processing Systems-Volume 2*, pages 2339–2347, 2012.

[28] Xi He, Graham Cormode, Ashwin Machanavajjhala, Cecilia M Procopiuc, and Divesh Srivastava. DPT: Differentially Private Trajectory Synthesis using Hierarchical Reference Systems. *Proceedings of the VLDB Endowment*, 8(11):1154–1165, 2015.

[29] Tomoharu Iwata and Hitoshi Shimizu. Neural Collective Graphical Models for Estimating Spatio-temporal Population Flow from Aggregated Data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3935–3942, 2019.

[30] Kaifeng Jiang, Dongxu Shao, Stéphane Bressan, Thomas Kister, and Kian-Lee Tan. Publishing Trajectories with Differential Privacy Guarantees. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2013.

[31] Biing Hwang Juang and Laurence R Rabiner. Hidden Markov Models for Speech Recognition. *Technometrics*, 33(3):251–272, 1991.

[32] Mostafa Karimzadeh, Zhongliang Zhao, Florian Gerber, and Torsten Braun. Pedestrians Complex Behavior Understanding and Prediction with Hybrid Markov Chain. In *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 200–207. IEEE, 2018.

[33] Meng Li, Liehuang Zhu, Zijian Zhang, and Rixin Xu. Achieving Differential Privacy of Trajectory Data Publishing in Participatory Sensing. *Information Sciences*, 400:1–13, 2017.

[34] Jianhua Lin. Divergence Measures Based on the Shannon Entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[35] Qiujian Lv, Yuanyuan Qiao, Nirwan Ansari, Jun Liu, and Jie Yang. Big Data Driven Hidden Markov Model Based Individual Mobility Prediction at Points of Interest. *IEEE Transactions on Vehicular Technology*, 66(6):5204–5216, 2016.

[36] Ryan McKenna, Daniel Sheldon, and Gerome Miklau. Graphical-model Based Estimation and Inference for Differential Privacy. In *International Conference on Machine Learning*, pages 4435–4444. PMLR, 2019.

[37] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. Predicting Taxi–passenger Demand Using Streaming Data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402, 2013.

[38] Sudha Morwal, Nusrat Jahan, and Deepti Chopra. Named Entity Recognition Using Hidden Markov Model (HMM). *International Journal on Natural Language Computing (IJNLC)*, 1(4):15–23, 2012.

[39] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. Knock Knock, Who's There? Membership Inference on Aggregate Location Data. *arXiv preprint arXiv:1708.06145*, 2017.

[40] Wahbeh Qardaji, Weining Yang, and Ninghui Li. Differentially Private Grids for Geospatial Data. In *2013 IEEE 29th international conference on data engineering (ICDE)*, pages 757–768. IEEE, 2013.

[41] Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating Next-cell Predictors with Extensive Wi-Fi Mobility Data. *IEEE transactions on mobile computing*, 5(12):1633–1649, 2006.

[42] Theresa Stadler, Bristena Oprisanu, and Carmela Troncoso. Synthetic Data–Anonymisation Groundhog Day. *arXiv preprint arXiv:2011.07018*, 2021.

[43] Uthaipon Tantipongpipat, Chris Waites, Digvijay Boob, Amaresh Ankit Siva, and Rachel Cummings. Differentially Private Mixed-type Data Generation for Unsupervised Learning. *arXiv preprint arXiv:1912.03250*, 2019.

[44] Keiichi Tokuda, Yoshihiko Nankaku, Tomoki Toda, Heiga Zen, Junichi Yamagishi, and Keiichiro Oura. Speech Synthesis Based on Hidden Markov Models. *Proceedings of the IEEE*, 101(5):1234–1252, 2013.

[45] Giuseppe Vietri, Grace Tian, Mark Bun, Thomas Steinke, and Steven Wu. New Oracle-efficient Algorithms for Private Synthetic Data Release. In *International Conference on Machine Learning*, pages 9765–9774. PMLR, 2020.

[46] Haiming Wang, Zhikun Zhang, Tianhao Wang, Shibo He, Michael Backes, Jiming Chen, and Yang Zhang. PrivTrace: Differentially Private Trajectory Synthesis by Adaptive Markov Model. *arXiv preprint arXiv:2210.00581*, 2022.

[47] Senzhang Wang, Jiannong Cao, and Philip Yu. Deep Learning for Spatio-temporal Data Mining: a Survey. *IEEE transactions on knowledge and data engineering*, 2020.

[48] Tianhao Wang, Joann Qiongna Chen, Zhikun Zhang, Dong Su, Yueqiang Cheng, Zhou Li, Ninghui Li, and Somesh Jha. Continuous Release of Data Streams under both Centralized and Local Differential Privacy. In *ACM CCS*, 2021.

[49] Tianhao Wang, Milan Lopuhaa-Zwakenberg, Zitao Li, Boris Skoric, and Ninghui Li. Locally Differentially Private Frequency Estimation with Consistency. In *NDSS'20: Proceedings of the NDSS Symposium*, 2020.

[50] Jianhao Wei, Yaping Lin, Xin Yao, and Voundi Koe Arthur Sandor. Differential Privacy-based Trajectory Community Recommendation in Social Network. *Journal of Parallel and Distributed Computing*, 133:136–148, 2019.

[51] H. Xi and M. Ashwin. Differentially Private Trajectory Synthesis. https://users.cs.duke.edu/~hexi88/project_dpt/index.html, 2015.

[52] Yonghui Xiao, Li Xiong, and Chun Yuan. Differentially Private Data Release Through Multidimensional Partitioning. In *Workshop on Secure Data Management*, pages 150–168. Springer, 2010.

[53] Changqiao Xu, Liang Zhu, Yang Liu, Jianfen Guan, and Shui Yu. Dp-ltod: Differential Privacy Latent Trajectory Community Discovering Services over Location-based Social Networks. *IEEE Transactions on Services Computing*, 2018.

[54] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private Data Release Via Bayesian Networks. *ACM Transactions on Database Systems (TODS)*, 42(4):1–41, 2017.

[55] Xinyang Zhang, Shouling Ji, and Ting Wang. Differentially Private Releasing Via Deep Generative Model (technical report). *arXiv preprint arXiv:1801.01594*, 2018.

[56] Zhikun Zhang, Tianhao Wang, Ninghui Li, Shibo He, and Jiming Chen. CALM: Consistent Adaptive Local Marginal for Marginal Release under Local Differential Privacy. In *ACM CCS*, 2018.

[57] Zhikun Zhang, Tianhao Wang, Ninghui Li, Jean Honorio, Michael Backes, Shibo He, Jiming Chen, and Yang Zhang. PrivSyn: Differentially Private Data Synthesis. *USENIX Security*, 2022.

[58] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining Interesting Locations and Travel Sequences from Gps Trajectories. In *Proceedings of the 18th international conference on World wide web*, pages 791–800, 2009.

A Computational Complexity Analysis

In this section, we theoretically analyze the computational complexity of different methods and then empirically evaluate their running time and memory consumption.

Here we assume a method discretizes the space into m_i cells in the i-th layer and the total number of cells $m = \sum m_i$. We also assume that the average length of the trajectory is a constant.

Time Complexity. The time complexity of PrivTrace can be analyzed by studying every step of the algorithm. In the discretization step, PrivTrace discretizes the geographical space into m cells and translates trajectories from location sequences to state sequences (i.e., find the cells where a trajectory has passed for all the trajectories). To find if a trajectory passes a certain cell takes $O(m|\mathcal{D}|)$. The process of calculating a Markov chain model requires counting the transitions in every trajectory. Thus, the time complexity of calculating Markov chain models is $O(|\mathcal{D}|)$. The trip distribution estimation is composed of calculating the shortest path and solving the convex optimization problem. The time

Table 4: Comparison of computational complexity for different methods.

Methods	Time Complexity	Space Complexity
AdaTrace [25]	$O(m\|\mathcal{D}\|+m^2)$	$O(m_1^2)$
DPT [28]	$O(m_g\|\mathcal{D}\|+m_g^h)$	$O(m_g^h)$
PrivTrace	$O(m\|\mathcal{D}\|+m^3)$	$O(m^3)$

complexities of them are $O(m^2 \ln(m))$ using the Dijkstra's algorithm and $O(m^2)$, respectively. The time complexity of generating and adding noise for the first-order and second-order models are $O(m^2)$ and $O(m^3)$. Finally, generating a trajectory depends on the transition count distributions in the Markov chain models, which has time complexity is $O(|\mathcal{D}|)$. Putting all steps together, the time complexity of PrivTrace is $O(m|\mathcal{D}|)+O(|\mathcal{D}|+m^2\ln(m))+O(|\mathcal{D}|)+O(m^2)+O(m^3) = O(m|\mathcal{D}|+m^3)$.

The analysis for AdaTrace is similar to PrivTrace: The time complexity of discretization is $O(m|\mathcal{D}|)$. The time complexity for learning the Markov chain model and estimating trip distribution are both $O(|\mathcal{D}|)$ since it involves scanning all trajectories. In the length distribution calculating, AdaTrace scans all trajectories and then chooses a proper distribution for all possible trips. The time complexity of length distribution calculating is $O(|\mathcal{D}|+m^2)$. In the steps above, the Laplacian noise is added to cell density, the first-order Markov model, and trip counts, respectively. The time complexity of Laplacian noise adding in AdaTrace is $O(m)+O(m^2)+O(m^2) = O(m^2)$. As the last step, the random walk process takes $O(m|\mathcal{D}|)$ to generate synthetic trajectories. The total time complexity of AdaTrace is $O(m|\mathcal{D}|+m^2)$.

For DPT, the discretization step uses several grids with different granularities (which are called hierarchical reference systems) to discretize the geographical space. In DPT, when the granularity of the division becomes finer, the number of states increases exponentially. We denote g as the number of different granularities and m_g as the number of states in the most fine-grained grid. The time complexity of discretization step is $O(m_g|\mathcal{D}|)$. In the prefix tree-building step, the algorithm takes $O(|\mathcal{D}|)$ for scanning all trajectories. After the prefix trees-building, the Laplacian noise will be injected to every node of the trees. The noise adding takes time complexity as $O(m_g^h)$, where h is the height of the tree with the most fine-grained grid. In the data generation process, DPT employs random walk on prefix tree and takes $O(|\mathcal{D}|)$. The total time complexity is $O(m_g|\mathcal{D}|)+O(m_g^h) = O(m_g|\mathcal{D}|+m_g^h)$.

Space Complexity. The memory consumption of all methods mainly comes from the model building process. PrivTrace uses the first-order Markov chain model and the second-order Markov chain model in model building. The space complexity of PrivTrace is $O(m^3)$.

AdaTrace stores the first-order Markov chain model, trip distribution, and length distribution in memory. For the first-order Markov chain model, there are m_1^2 transition count values to be stored. For trip distribution and length distribution, information for all possible trips is stored and takes $O(m_1^2)$ space. The space complexity for AdaTrace is $O(m_1^2)$.

DPT stores prefix trees in memory. The number of nodes in the tree of the most fine-grained grid is m_g. The memory consumption of this tree is $O(m_g^h)$, where h is the height of this tree. Since the most fine-grained tree consumes the most memory, the space complexity of DPT is $O(m_g^h)$. Both the time and space complexity of different algorithms are shown in Table 4.

Table 5: Comparison of running time for different methods.

Datasets	Brinkhoff	Taxi	Geolife
AdaTrace [25]	2 min 30 s	4 min 44 s	1 min 45 s
DPT [28]	18 min 47 s	43 min 51 s	34 min 27 s
PrivTrace	6 min 21 s	8 min 33 s	7 min 24 s

Table 6: Comparison of memory consumption of different methods. The unit is Megabytes.

Datasets	Brinkhoff	Taxi	Geolife
AdaTrace [25]	0.05	0.04	0.53
DPT [28]	1003.77	111.22	1755.23
PrivTrace	12.96	63.30	193.75

Empirical Evaluation. Table 5 and Table 6 illustrate the running time and memory consumption for all methods on the three datasets. The empirical running time in Table 5 shows that AdaTrace has the best running time performance. This is because AdaTrace has relatively low time complexity as shown in Table 4. PrivTrace is slower than AdaTrace since it contains a graph-based method and convex optimization in the trip distribution estimation. For memory consumption in Table 6, we observe that DPT consumes the most memory since storing the reference system is space-consuming.

B More Details

B.1 Details of Normalization in Section 4.2

Denote the number of occurrences of trajectory S_i in cell j as δ_{ij}, we can represent the trajectory density as a vector $(\sum_i \delta_{i1}, \sum_i \delta_{i2}, \dots \sum_i \delta_{iK^2})$. The sensitivity of the trajectory density is the impact of one trajectory on this vector, which is unbounded since one trajectory could appear in any number of cells. To bound the sensitivity, we propose to use the length-normalized density, which is defined by $\delta'_{ij} = \delta_{ij}/\sum_j \delta_{ij}$ instead of δ_{ij}. Denote the length-normalized density query vector $(\sum_i \delta'_{i1}, \sum_i \delta'_{i2}, \dots \sum_i \delta'_{iK^2})$ as $\eta(\mathcal{D})$, we can see

$$\begin{aligned}\mathsf{GS}_\eta &= \max_{S_k \in \mathcal{D}} ||\eta(\mathcal{D}) - \eta(\mathcal{D} - S_k)||_1 \\ &= \max_{S_k \in \mathcal{D}} ||(\delta'_{k1}, \delta'_{k2}, \ldots, \delta'_{kK^2})||_1 \\ &= \max_{\forall S_k \in \mathcal{D}} |\sum_{j=1}^{K^2} \delta_{kj} / \sum_{j=1}^{N^2} \delta_{kj}| = 1\end{aligned}$$

B.2 Details of Bounding Sensitivity in Section 4.3

In Markov chain model, denote σ_i as a state corresponding to C_i, and any trajectory is transformed into a state sequence. For example, if a trajectory S passes through the following cells in order: C_2, C_3, C_7, the state sequence it is transformed into is $(\sigma_2, \sigma_3, \sigma_7)$.

We denote the length-normalized count for transition P as $N'_{\mathcal{D}}(P) = \sum_{\forall T \in \mathcal{D}} \frac{N_T(P)}{|T|}$, where $N_T(P)$ means the appearance of P in trajectory T. And the set of length-normalized transition counts is defined as $\gamma^k(\mathcal{D}) = \{N'_{\mathcal{D}}(P), \forall P \in \Sigma^{k+1}\}$. Applying the length-normalization technique, the kth-order Markov chain model can be built easily by calculating γ^k.

Here the core idea of bounding sensitivity is to divide the transition count by the length of the trajectory. By doing this, the sensitivity of the Markov chain model counting query can be bounded by 1.

Theorem 2. *Define $N'_{\mathcal{D}}(r) = \sum_{\forall T \in \mathcal{D}} \frac{N_T(r)}{|T|}$ where r is any length-$k+1$ sequence, the sensitivity of outputing all $N'_{\mathcal{D}}(r)$, denoted by $\gamma^k(\mathcal{D}) = \{N'_{\mathcal{D}}(r), \forall r \in \Sigma^{k+1}\}$, is 1.*

The theorem can be proved by first showing that a single sequence T's contribution γ^k is limited by $|T|$. The following lemma tells us that $\sum_{r \in \Sigma^l} N_T(r)$ cannot be larger than $|T|$.

Lemma 3. *Given a set of all possible subsequences of length l, denoted by Σ^l, and a sequence T, $\sum_{r \in \Sigma^l} N_T(r) \leq |T|$, where $|T|$ is the length of sequence T.*

Proof. The calculating of $\sum_{r \in \Sigma^l} N_T(r)$ is to count all subsequences of length l in T. Considering the sequence can only have no more than $|T|$ continuous subsequences of the same length, $\sum_{r \in \Sigma^l} N_T(r) \leq |T|$. □

With that, we can prove Theorem 2:

Proof. We denote a dateset of sequences as $\mathcal{D}$. The sensitivity of γ^k is $\max_{\forall T' \in \mathcal{D}} \gamma^k(\mathcal{D}) - \gamma^k(\mathcal{D} - T')$. We give an order to all transitions P of length $k+1$ and γ^k is represented as a vector following the order of transitions. For a transition P_i, the ith element of $\gamma^k(\mathcal{D}) - \gamma^k(\mathcal{D} - T')$ is $N'_{\mathcal{D}}(P_i) - N'_{\mathcal{D}-T'}(P_i) = N'_{T'}(P_i)$. Thus, the sensitivity is,

$$\begin{aligned}\mathsf{GS}_{\gamma^k} &= \gamma^k(\mathcal{D}) - \gamma^k(\mathcal{D} - T') \\ &= \max_{\forall T' \in \mathcal{D}} ||(N'_{T'}(P_1)), \ldots, (N'_{T'}(P_{m^{k+1}}))||_1 \\ &= \max_{\forall T' \in \mathcal{D}} |\sum_{P_i} N'_{T'}(r)| \\ &= \max_{\forall T' \in \mathcal{D}} \sum_{P_i} \frac{N_{T'}(r)}{|T'|}\end{aligned}$$

According to Lemma 3, $\sum_{r \in \Sigma^l} N_r(T) \leq |T|$, $\mathsf{GS}_{N'} \leq 1$ □

B.3 Details of Trip Distribution Estimation in Section 4.4

Calculation for b_i. Considering the sets of states $\sigma_1, \ldots, \sigma_m$, by calculating γ^1 we will get $N'_{\mathcal{D}}((\sigma_{start}, \sigma_i))$ and $N'_{\mathcal{D}}((\sigma_j, \sigma_{end}))$ for all σ_i and σ_j. For presentation purpose, we denote $N'_{\mathcal{D}}((\sigma_{start}, \lambda_i))$ and $N'_{\mathcal{D}}((\lambda_j, \sigma_{end}))$ as b'_i and q'_j, respectively. b'_i and q'_j are values of b_i and q_j before adding Laplacian noise. Relation between ϕ_{ij} and b_i can be built through b'_i.

Without loss of generality, we first analyze the calculation of b'_i (q'_j can be calculated in a similar way). We denote the set of all trajectories starting at σ_i and ending at σ_j by ϕ_{ij}. $\frac{|\phi_{ij}|}{\sum_{T \in \phi_{ij}} 1/|T|}$ is the harmony average of the sequence length of trajectory in ϕ_{ij}. With the assumption that people tend to travel the shortest path, we have l_{ij} is very close to the harmony average length. In later calculation, we use l_{ij} as the harmony average length. Here we provide how we compute the relation $b'_i = \sum_j \frac{t_{ij}}{l_{ij}}$: b'_i is the length-normalized transition count of transition $P^s_i = (\sigma_{start}, \sigma_i)$, which is calculated by $b'_i = N'_{\mathcal{D}}(P^s_i) = \sum_{\forall T \in \mathcal{D}} \frac{N_T(P^s_i)}{|T|} = \sum_j \sum_i \sum_{\forall T \in \phi_{ij}} \frac{N_T(P^s_i)}{|T|}$. Considering that every trajectory can only belongs to one ϕ, we can divide $\mathcal{D}$ into two sets:ϕ_{ij} and $\mathcal{D} - \phi_{ij}$. By this division, we have $b'_i = \sum_j \sum_{\forall T \in \phi_{ij}} \frac{N_T(P^s_i)}{|T|} + \sum_j \sum_{v \neq i} \sum_{\forall T \in \phi_{vj}} \frac{N_T(P^s_i)}{|T|}$. It can be concluded that $N_T(P^s_i) = 1$ if T starts at σ_i and $N_T(P^s_i) = 0$ otherwise, since a trajectory only has one start state. Thus, for T in $\phi_{ij}, N_T(P^s_i) = 1$. By the definition of the harmony average and replacing $|\phi_{ij}|$ by t_{ij} , we have $\sum_{\forall T \in \phi_{ij}} \frac{1}{|T|} = \frac{t_{ij}}{l_{ij}}$. Replacing $\sum_{\forall T \in \phi_{ij}} \frac{1}{|T|}$ by $\frac{t_{ij}}{l_{ij}}$, we have $b'_i = \sum_j \sum_{\forall T \in \phi_{ij}} \frac{1}{|T|} + 0 = \sum_j \frac{t_{ij}}{l_{ij}}$. Since b is b' together with the Laplacian noise, we have $b_i \simeq \sum_{j=1}^{m} \frac{t_{ij}}{l_{ij}}$.

Estimating l_{ij}. Observing that the shortest length trajectory is similar to the minimum-weight shortest path in a graph, we design a graph-based method to estimate l_{ij}. We construct a graph whose nodes are states in the Markov chain model. An edge exists between two nodes if the two cells corresponding to the two states are neighbors. The weight of an edge is the geographical distance between the two cells. Then, we use the minimum-weight shortest path algorithm to calculate l_{ij}.

B.4 Details of Evaluation Metrics in Section 5.1

- **Length Distribution.** The length of a trajectory measures the summation of distances between all two adjacent points. We bucketize the length into 50 bins and count the number of trajectories falling into each bin to calculate the length distribution. We use the Jensen-Shannon divergence (JSD) to measure the error between $\mathcal{D}_s$ and $\mathcal{D}_o$.
- **Diameter Distribution.** The diameter indicates the maximum distance between any two points in a trajectory. Similar to the length distribution, we bucketize the diameter into 50 bins to obtain the diameter distribution and use JSD to measure the error.
- **Trajectory Density.** It measures the number of trajectories passing through a specific area. We first generate 500 random circle areas with random radius in the map, and count the number of trajectories passing through each area. We use the average relative error (ARE) to measure the error. Denote the set of density queries as Q, the ARE of all queries is calculated as:

$$\mathsf{ARE}_{TD} = \frac{1}{|Q|}\sum_i \frac{|Q_i(\mathcal{D}_o) - Q_i(\mathcal{D}_s)|}{max(Q_i(\mathcal{D}_o), \varphi)}$$

  where $|Q|$ is the cardinality of Q, φ is a factor to bound the impact of a query of small real value.
- **Transition Pattern.** It captures the frequency of transiting from one place to another. We use a 20×20 uniform grid to discretize the geographical space and count the frequencies of all transition patterns. In most downstream tasks, only frequent patterns are considered. We use the ARE of the top μ frequent patterns to measure the error. The ARE of the top-μ frequent transition patterns P is calculated as:

$$\mathsf{ARE}_{TP} = \frac{1}{\mu}\sum_{P \in \mathcal{P}_{top}} \frac{|N_P(\mathcal{D}_o) - N_P(\mathcal{D}_s)|}{max(N_P(\mathcal{D}_o), \varphi)}$$

  Here N_P means the number of pattern P's occurrences and $\mathcal{P}_{top}$ means the set of the top-μ frequent transition patterns. In our experiments, we only consider the transition patterns with length between 2 to 5, i.e., the transition patterns whose number of states is between 2 to 5. Then we rank these counts and find the top-μ transition patterns since users usually focus on the frequent patterns. In the experiments, we set μ as 200.

B.5 Details of Competitors in Section 5.1

The AdaTrace code[1] chooses between three candidate distributions by comparing the candidate distributions with distribution in the original dataset without adding noise (in LengthDistribution.java, Line 133 to Line 135). Meanwhile, AdaTrace extract mean and median from original dataset using the same privacy budget (in LengthDistribution.java, line 81 and 86). For a fair comparison, we fix this problem and report the results of the fixed version of AdaTrace. That is, we divide the privacy budget for length distribution extraction in AdaTrace into three parts equally. The process of extracting mean and median and choosing a distribution from the candidate consumes one part, respectively.

[1] https://github.com/git-disl/AdaTrace

C Detailed Proof of PrivTrace's DP Guarantee

Proof. PrivTrace consists of four components: Geographical space discretization, Markov chain models learning, trip distribution estimation, and trajectories generation. In the following, we check the privacy budget consumption of each component.

- **Geographical Space Discretization.** We first count the occurrences of trajectories passing through the cells in the coarse-grained grid. We then add Laplacian noise to the counts in those cells. To control sensitivity, we normalize the trajectories so that each trajectory contributes at most 1 to the counts (the sensitivity is bounded to 1, see Appendix B.1), and the privacy budget consumed is ε_1. We further partition dense cells (cells whose noisy counts are above a threshold) by a more fine-grained grid, but that step, by the postprocessing property of DP (see Section 4.2), consumes no privacy budget.

- **Markov Chain Models Learning.** PrivTrace estimates the count of state transition to build the first- and the second-order Markov models (the states of the Markov models are the cells from the previous step). Similar to the previous step, we normalize the trajectory so that sensitivity is 1 (refer to Appendix B.2), and then add Laplace noise. The privacy budget consumed for the first- and the second-order Markov models are ε_2 and ε_3, respectively.

- **Trip Distribution Estimation.** PrivTrace constructs an optimization problem to estimate the trip distribution. The data used in the optimization problem is gathered in the differentially private Markov models; thus the trip distribution estimation is a postprocessing operation, and no privacy budget is consumed.

- **Trajectories Generation** Trajectory generation uses data estimated in the previous components; it is also a postprocessing operation. Concretely, the information that we use to choose between the first- and second-order Markov models for next step prediction is the noisy counts in the first-order Markov model; thus, mixing the two Markov models for prediction is a postprocessing operation and does not consume extra privacy budget.

By the sequential composition property of DP as discussed in Section 2.4, PrivTrace satisfies ε-DP, where $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$. □