A Deep Generative Model for Trajectory Modeling and Utilization

Yong Wang
Tsinghua University, China
wangy18@mails.tsinghua.edu.cn

Guoliang Li
Tsinghua University, China
liguoliang@tsinghua.edu.cn

Kaiyu Li
Tsinghua University, China
ky-li18@mails.tsinghua.org.cn

Haitao Yuan
Tsinghua University, China
yht16@mails.tsinghua.edu.cn

ABSTRACT

Modern location-based systems have stimulated explosive growth of urban trajectory data and promoted many real-world applications, *e.g.,* trajectory prediction. However, heavy big data processing overhead and privacy concerns hinder trajectory acquisition and utilization. Inspired by regular trajectory distribution on transportation road networks, we propose to model trajectory data privately with a deep generative model and leverage the model to generate representative trajectories for downstream tasks or directly support these tasks (*e.g.,* popularity ranking), rather than acquiring and processing the original big trajectory data. Nevertheless, it is rather challenging to model high-dimensional trajectories with time-varying yet skewed distribution. To address this problem, we model and generate trajectory sequence with judiciously encoded spatio-temporal features over skewed distribution by leveraging an important factor neglected by the literature –the underlying road properties (*e.g.,* road types and directions), which are closely related to trajectory distribution. Specifically, we decompose trajectory into map-matched road sequence with temporal information and embed them to encode spatio-temporal features. Then, we enhance trajectory representation by encoding inherent route planning patterns from the underlying road properties. Later, we encode spatial correlations among edges and daily and weekly temporal periodicity information. Next, we employ a meta-learning module to generate trajectory sequence step by step by learning generalized trajectory distribution patterns from skewed trajectory data based on the well-encoded trajectory prefix. Last but not least, we preserve trajectory privacy by learning the model differential privately with clipping gradients. Experiments on real-world datasets show that our method significantly outperforms existing methods.

PVLDB Reference Format:
Yong Wang, Guoliang Li, Kaiyu Li, and Haitao Yuan. A Deep Generative Model for Trajectory Modeling and Utilization. PVLDB, 16(4): 973 - 985, 2022.
doi:10.14778/3574245.3574277

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/wangyong01/MTNet_Code.

Proceedings of the VLDB Endowment, Vol. 16, No. 4 ISSN 2150-8097.
doi:10.14778/3574245.3574277

1 INTRODUCTION

Advances in vehicle-embedded sensors and mobile location-based services have generated big trajectory data and boosted urban trajectory data applications [13, 47].

Data analysts such as urban planners rely on accessible trajectory data (*e.g.,* data published by online car-hailing platforms [1]) to generate insights for strategic planning. However, heavy trajectory maintenance and processing overhead and privacy concerns hinder these applications.

Specifically, data analysts do not have too much computing resource for maintaining and processing big trajectory data, thus they want to conduct lightweight analysis. Compression-based and sampling-based methods are commonly adopted to alleviate this issue [7, 30, 53, 58]. Nevertheless, the former still suffers heavy trajectory decompression and processing overhead, while the latter is time-consuming for a large sampling ratio and has poor performance for a small sampling ratio. Furthermore, these methods raise privacy concerns because urban trajectories contain sensitive individual information (*e.g.,* unique moving patterns) and are not safe for publishing. Although there are some privacy-preserving trajectory data publishing methods [18, 19], their trajectory modeling and privacy-preserving approaches have practical limitations. First, they model raw trajectories into geo-grids sequence and neglect that trajectories are constrained on road networks. Second, they preserve privacy by adding noises to trajectory prefixes, while trajectory length is varying, which significantly reduces data utility (especially long trajectory with more accumulative noise). Besides, anonymization methods are vulnerable to attackers with background knowledge [45].

Inspired by the fact that trajectories are regularly generated on a road network with underlying moving patterns, we propose to model trajectory data with a deep generative model such that: (a) the model is lightweight, (b) the model is of high utility to support the downstream trajectory applications, and (c) the model preserves individual privacy, to address the above limitations. However, there are two challenges to model map-matched trajectories. (C1) It is hard to model real-world trajectories because their spatial route distribution is time-varying. External spatio-temporal features such as road network points of interests (POIs) and weather information are often adopted to enhance trajectory modeling [47]. Nevertheless, the underlying road network properties (*e.g.,* road types and road directions) which are critical for capturing trajectory routing patterns are overlooked in existing trajectory works. We show that road properties are closely related to and thus important to

[1] https://movement.uber.com; https://outreach.didichuxing.com

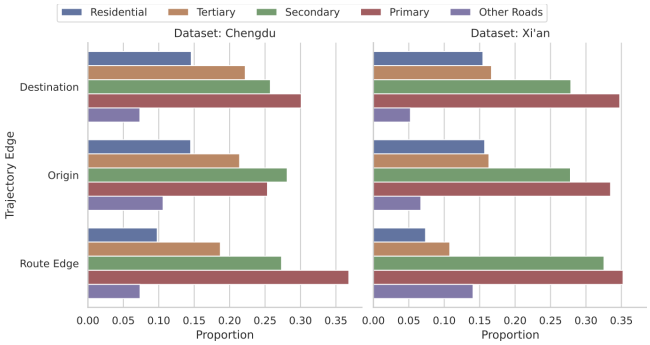


Figure 1: Trajectory spatial distribution is closely related to road types. Trajectory origins and destinations are more likely to be on residential or tertiary roads, while trajectory route pass more higher-level roads, *e.g.,* primary roads.

model trajectory distribution with an example of trajectory spatial distribution on different road types in Figure 1. (C2) It is hard to model real-world trajectories because trajectory data are high dimensional with respect to the number of route edges and have very skewed distribution on the road network [60]. Although prevalent spatio-temporal graph convolutional models [39] are shown to be effective on modeling common graph node or edge attributes, they are not effective enough for modeling high-dimensional yet skewed trajectories (will be shown in experiments).

To address the above challenges, we propose a deep generative model named MTNet to effectively model high-dimensional yet skewed trajectory data by fully leveraging the underlying road properties. Firstly, in addition to spatial route edge and temporal departure and travel time features of trajectory, we also collect the important road properties from OpenStreetMap and encode them into road meta knowledge to enhance spatio-temporal trajectory representation, to address the challenge C1. Specifically, MTNet decomposes a trajectory into a map-matched road sequence with corresponding departure time and travel time of each passing road. We (i) embed road, departure time slot and time-invariant influencing factors into hidden feature vectors; (ii) encode the underlying road properties into road meta knowledge to capture inherent route planning patterns; (iii) encode spatial correlations among edges with graph embedding techniques [17]; and (iv) encode daily and weekly temporal periodicity information with temporal graph[52]. Secondly, MTNet employs a knowledge-based meta-learning module to generate high-dimensional trajectory step by step by learning generalized trajectory distribution patterns from skewed trajectories, to address the challenge C2. Specifically, given an encoded trajectory prefix and the corresponding road meta knowledge, MTNet generates a possible subsequent road with selection probability and its travel time. By recursively taking as input the preceding road, MTNet extends the encoded trajectory prefix and outputs a subsequent road, until a complete trajectory is finally generated. Last but not least, we preserve the privacy of trajectories for learning MTNet by adopting differentially private training techniques with clipping gradients [1], where we control gradients from a single trajectory as a whole and improve the utility.

Leveraging well-encoded spatio-temporal trajectory features and inherent road meta knowledge based on the above designs, MTNet can not only generate representative trajectories with close-to-real-data statistics in intended granularities, but also directly support multiple real-world trajectory applications [5, 48], without further maintaining and processing a large scale trajectory data. Specifically, we support (i) Trajectory popularity ranking [8, 49]. Given a candidate set of trajectories, MTNet can estimate the time-dependent appearance probability (or popularity) for each of them. (ii) Trajectory suffix prediction[28]. Given an input trajectory prefix, MTNet can predict the most probable trajectory suffix. (iii) Time-dependent travel time estimation [57]. Given a trajectory and a departure time, MTNet can predict the travel time on each passed road segment. Note that MTNet is designed for publishing trajectory privately and periodically, which aims to support downstream applications on historical trajectories conveniently and effectively. We need to retrain the model to adapt to the new patterns when confronted with sudden and drastic shifts such as COVID-19.

Table 1: Notations

Notation	Definition
$\langle lat, lng, tstamp\rangle_{1:L_r}$	Raw trajectory sequence
$\langle x, td, tc\rangle_{1:L}$	Map-matched trajectory sequence
x, td, tc	Route edge, departure time and travel time
$\tau(td)$	Discrete departure time slot of td
W_x, W_τ, W_{tc}	Trajectory embedding matrices
xe, te	Edge and departure time slot embedding vector
N_τ	Number of departure time slots
L_r, L	Raw and map matched trajectory sequence length
t	Sequence generation step, $t \in [1, L]$
z	Gaussian noise sampled from $\mathcal{N}(0, 1)$
d_{xe}, d_{te}, d_z	Dimension of xe, te and z
W^{soft}, W^{sig}	Neural weights output from `MetaLearner`
h, h^x, h^{tc}	hidden state of LSTMs, x and its travel time
d_h, d_o	Dimension of h and of $W^{soft}, W^{sig}, h^x, h^{tc}$

In summary, we make the following contributions with MTNet.
(1) We formalize map-matched trajectory modeling problem with both spatio-temporal information, and formulate trajectory generative model for supporting trajectory applications (Section 2).
(2) We encode trajectory sequence with spatio-temporal features, underlying road meta knowledge, spatial correlations among road edges and temporal periodicity information, to enhance trajectory modeling ability (Section 3.1 and 3.2).
(3) We propose a deep generative model named MTNet to model skewed trajectory data based on the above well-encoded trajectory features (Section 3.3). We leverage MTNet to generate synthetic yet realistic trajectories for downstream applications and directly support multiple trajectory applications (Section 3.4).
(4) We illustrate how to learn MTNet differential privately with clipping gradients (Section 4).
(5) We conducted comprehensive experiments on two real-world datasets. The results show that MTNet significantly outperforms existing methods (Section 5).

2 PRELIMINARIES

2.1 Background

Raw Trajectory. A raw trajectory is a sequence of GPS points with corresponding timestamps, denoted as $\langle lat, lng, tstamp\rangle_{1:L_r} = (\langle lat, lng, tstamp\rangle_1, \langle lat, lng, tstamp\rangle_2, \cdots, \langle lat, lng, tstamp\rangle_{L_r})$, in which L_r is the trajectory length. We use $lat_{1:L_r}, lng_{1:L_r}, tstamp_{1:L_r}$ to denote latitude, longitude, timestamp sequences respectively.

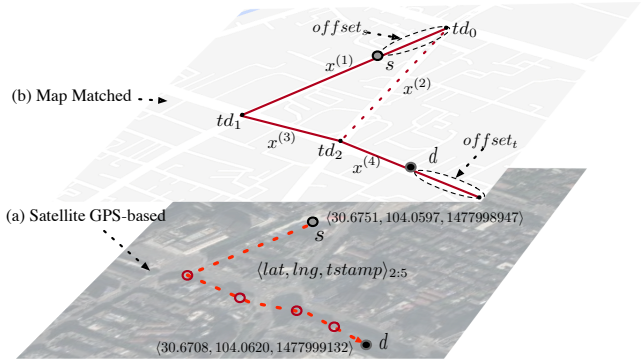


Figure 2: Satellite GPS based and Map-matched Trajectories

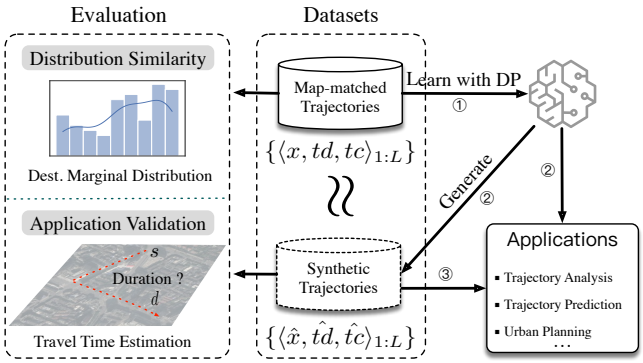


Figure 3: Trajectory Modeling and Utilization

Example 1:[Raw Trajectory] Consider trajectory (⟨30.6751, 104.0597, 1477998947⟩, ⋯, ⟨30.6708, 104.0620, 1477999132⟩) from origin s to destination d in Figure 2 (a). Each $\langle lat, lng, tstamp\rangle_i$ is a coordinate with the timestamp when vehicle passes. The travel time from s to d is timestamp at d minus timestamp at s, *i.e.,* 185 seconds. □

Road Networks. A road network is a directed graph $G = \langle V, E\rangle$, where V is the vertex set, and E is the edge set. Each $x \in E$ is a directed edge, *i.e.,* a directed road segment. Here we use undirected edges as an example for ease of presentation, and our method and experiments are based on directed ones.

Map-matched Trajectory. A raw trajectory represented by timestamped GPS sequence can be matched to road networks, called map-matched trajectory. A map-matched trajectory can be represented as $\langle x, td, tc\rangle_{1:L} = (\langle x, td, tc\rangle_1, \cdots, \langle x, td, tc\rangle_L)$, where x_i is a road segment, td_i is the departure time at the start point of x_i, tc_i denotes the travel time (*i.e.,* $tc_i = td_{i+1} - td_i$), and L is the trajectory sequence length. We denote the sequence of trajectory routes, departure time, and travel time by $x_{1:L}$, $td_{1:L}$ and $tc_{1:L}$, respectively.

Example 2:[Map-matched Trajectory] Consider a map-matched trajectory $(\langle x^{(1)}, td_0, tc_0\rangle, (\langle x^{(3)}, td_1, tc_1\rangle, \langle x^{(4)}, td_2, tc_2\rangle)$ in Figure 2(b). The road network has four connected edges, *i.e.,* $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$. Departure time td_t and travel time tc_t, $t \in \{0, 1, 2\}$ for each passed edge in Figure 2 (b) are computed based on linear interpolations with coordinates and timestamps in Figure 2 (a). Trajectory origins and destinations are aligned to corresponding edge joints, with offsets $\texttt{offset}_\texttt{s}$ and $\texttt{offset}_\texttt{d}$. □

Differential Privacy. Differential privacy (DP) is a mathematical property of a randomized algorithm to describe the level of privacy preservation of individuals from the input dataset. Intuitively, DP ensures that a user can only get aggregate information of the input data, but cannot learn anything on a particular individual. Assume any two neighboring trajectory datasets with only one different trajectory. A randomized algorithm A achieves (ϵ, δ)-DP if for all output subset S of A and for any two neighboring trajectory datasets $D \in \mathcal{D}$, $D' \in \mathcal{D}$ with only one different trajectory:

$$Pr[A(D) \in S] \leq e^{\epsilon} Pr[A(D') \in S] + \delta \tag{1}$$

where privacy level is measured by ϵ and δ, *i.e.,* the smaller ϵ and δ are, the stronger the privacy is. Gaussian mechanism is widely used to implement DP. For example, given a function $f : \mathcal{D} \to R^d$, it adds noise from Gaussian distribution $\mathcal{N}(0, S_f^2\sigma^2)$, *i.e.,* $A(D) = f(D) + \mathcal{N}(0, S_f^2\sigma^2)$, where $S_f = \max_{\text{neighboring } D,D'} ||f(D) - f(D')||_2$ is the L_2 sensitivity of f and σ is the noise scale. For $\epsilon \in (0, 1)$, if $\sigma \geq \sqrt{2\ln(1.25/\delta)}/\epsilon$, then A satisfies (ϵ, δ)-DP [10].

2.2 Trajectory Modeling and Utilization

We study the problem of trajectory modeling and utilization using deep generative models with DP guarantee. We first model trajectory distribution to quantify the utility of generated data. We then propose trajectory generative model to model and support trajectory generation. Finally, we discuss how to achieve DP guarantee.

Modeling Trajectory Distribution. Owing to various mixed influencing factors (hidden variables), urban trajectories have complex inherent patterns. Intuitively, a trajectory is generated step by step from an origin edge to a destination edge with the corresponding temporal information, where each edge is dependent on its trajectory prefix. Consequently, given a trajectory dataset $\{\langle x, td, tc\rangle_{1:L}\}$, its trajectory joint probability density function (PDF) can be formulated as follows:

$$f(\langle x, td, tc\rangle_{1:L}) = \underbrace{f(\langle x, td, tc\rangle_1)}_{\text{origin edge}} \prod_{t=2}^{L} \underbrace{f(\langle x, td, tc\rangle_t | \langle x, td, tc\rangle_{1:t-1})}_{\text{prefix dependent step } t}$$

$$= Pr(x_1) f(td_1|x_1) f(tc_1|x_1, td_1) \cdot \prod_{t=2}^{L} Pr(x_t|x_{1:t-1}, td_t) f(tc_t|x_t, td_t)$$

where $Pr(x_1)$ is the probability distribution of the first edge x_1, $f(td_1|x_1)$ is the PDF of departure time td_1 at x_1, $f(tc_1|x_1, td_1)$ is the PDF of travel time tc_1 on x_1 when depart at td_1, $Pr(x_t|x_{1:t-1}, td_t)$ and $f(tc_t|x_i, td_i)$ are the conditional distribution of t-th edge x_t and its travel time tc_t, respectively. For $\forall t \in [2, L]$, we have x_t topologically constrained by it preceding edge x_{i-1}, and $td_t = td_{t-1} + tc_{t-1}$. It is very hard to directly model the above unknown (hidden) joint distribution $P_d = f(\langle x, td, tc\rangle_{1:L})$, as the distribution space grows exponentially with respect to L. Thus, we propose to implicitly approximate the real trajectory distribution with a deep generative model. Formally, we learn a tractable generative model $\mathcal{G}$ to fit the real urban trajectory generation P_d, such that minimizing the distribution distance between P_d and $\mathcal{G}$. In particular, we use a symmetric and smoothed measurement, Jensen-Shannon divergence (JSD), to compute the distribution distance, *i.e.,*

$$\min \frac{1}{2}\mathbb{E}_{X\sim P_d}[\log \frac{P_d(X)}{M}] + \frac{1}{2}\mathbb{E}_{X\sim \mathcal{G}}[\log \frac{\mathcal{G}(X)}{M}] \tag{2}$$

where X is a trajectory from the real dataset or generated by $\mathcal{G}$, and $M = \frac{1}{2}(P_d(X) + \mathcal{G}(X))$. It will be factorized as route edge and destination marginal distribution distances in Section 5.1. Figure 3 presents the trajectory modeling and utilization problem.

Formulating Trajectory Generative Model $\mathcal{G}$. Intuitively, real-world trajectory data are generated from various origins at specific departure time. Based on this observation, to flexibly support synthetic trajectory generation in different granularities and support multiple trajectory applications simultaneously, we formulate $\mathcal{G}$ as a conditional generative model with origin edge and departure time (of the following edge) as input conditions. We factorize trajectory sequence generation into step by step route edge and travel time generation process. Specifically,

$$\mathcal{G}(\langle \hat{x}, \hat{td}, \hat{tc} \rangle_{1:L} | x_0, td_1) = \prod_{t=1}^{L} \mathcal{G}(\hat{x}_t, \hat{tc}_t | \hat{x}_{0:t-1}, \hat{td}_{1:t}) \quad (3)$$

where $\hat{x}_0 = x_0$ and $\hat{td}_1 = td_1$, *i.e.,* input conditions. In each step $t \in [1, L]$, based on the current trajectory prefix $\langle \hat{x}, \hat{td}, \hat{tc} \rangle_{1:t-1}$, it produces (generates) the next edge $\hat{x}_t$ and the corresponding travel time $\hat{tc}_t$ stochastically, where $\hat{x}_{t-1}$ and $\hat{tc}_{t-1}$ are generated in step $t-1$, and $\hat{td}_t = \hat{td}_{t-1} + \hat{tc}_{t-1}$. Note that the generated trajectory length does not need to be exactly L, *i.e.,* the route edge generation process may stop earlier with some probability during each step, where the sequence is padded with a stop flag. Besides, random deviations during the generation process are simulated via multinomial sampling.

Example 3:[Trajectory Generation] Consider a trajectory starts from $x_0 = x^{(1)}$ and arrives at its stop point at td_1 in Figure 2. Suppose a model outputs $\mathcal{G}(\hat{x}_1 = x^{(3)} | (x^{(1)}), (td_1)) = 83\%$ with $\hat{tc}_1 = 52s$, and $\mathcal{G}(\hat{x}_1 = -|(x^{(1)}), (td_1)) = 17\%$ with $\hat{tc}_1 = 0s$, where '$-$' means the stop flag, and $\mathcal{G}$'s output varies stochastically. We make a selection by multinomial sampling, *e.g.,* generate $x^{(3)}$ with $\hat{tc}_1 = 52s$. □

Achieving DP. We train the model $\mathcal{G}$ with DP guarantee that privatizes the access of each trajectory during model training by clipping and perturbing model gradients with random noise, which will be described in details in Section 4.

3 MTNET

Trajectories are hard to model due to high dimensions, complex correlations among road edges and trajectories, and time-varying yet highly skewed trajectory distributions. Nevertheless, there exists inherent route planning patterns (hidden variables) on the underlying road network that influence trajectory generation, *e.g.,* moving patterns between residential areas and working areas and route planning patterns based on road capacities and geographical directions. Based on this observation, we propose a deep generative model named MTNet, which utilizes spatio-temporal feature embeddings and the inherent road knowledge of trajectory generation based on meta learning. Figure 4 presents an overview of MTNet, which is composed of the following four modules. (1) Trajectory representation module (Figure 4 (b)) is for embedding spatial route edge and temporal features, aiming to represent trajectory sequence effectively. (2) Road knowledge encoding module (Figure 4 (c)) is for encoding road edge properties, which introduces inherent route planning features for enhancing trajectory prefix encoding and suffix generation. (3) Trajectory prefix encoding module (part of Figure 4 (a)) is for encoding trajectory prefix, aiming to extract long-term spatio-temporal dependency from high-dimensional trajectory prefix. (4) Meta generator module (Figure 4 (d)) is for trajectory route edge and its travel time generation, aiming to effectively support trajectory modeling and generation over skewed trajectory data. Besides, the output is masked by road network topology constraints in Figure 4 (e), in order to improve generation efficiency and quality.

3.1 Trajectory Representation

For trajectory modeling, the first problem is how to represent trajectory sequence with spatio-temporal characteristics, *i.e.,* trajectory route edge and corresponding departure time and travel time information. Specifically, we embed each discretely labelled road edge into a low-dimensional vector to capture the geographical distribution features of road edges. We discrete departure time into time slots and further embed it to capture the temporal distribution features of trajectory data. We also embed time-invariant hidden factors of travel time for each edge. These embeddings are concatenated with encoded road knowledge as unified input (Figure 4 (a)).

Edge Embedding. Trajectory route sequence is originally represented as discrete sequence of road edge labels, which cannot be directly fed to MTNet for trajectory prefix encoding. For road network $G = \langle V, E \rangle$, we may directly use one-hot encodings, which generates a $|E|$ dimensional vector and takes the position of a specified edge as 1 and others as 0. However, it leads to the curse of dimensionality, as $|E|$ is quite large. To overcome this issue, we embed edges with matrix $W_x \in \mathbb{R}^{|E| \times d_{xe}}$, where each $x \in E$ is represented with a low-dimensional (d_{xe}) vector $xe = W_x[x]$.

Time Embedding. Trajectory distribution varies in different time slots, *e.g.,* urban trajectories depart from residential area to working area in the morning, and turn back in the evening. Besides, route edge travel time is also time dependent, *i.e.,* longer during peak hours and shorter during off-peak hours. To model the variable temporal distribution of trajectory data, we embed departure time and make it an input of MTNet. Specifically, we discrete departure time into N_τ time slots for each day, where each time slot $\tau(td) \in [0, N_\tau]$. Further, we produce $7 * N_\tau$ time slots for a week to capture the daily changes, which can also be applied to month or other periods. For example, we discrete 86400s in each day into 48 time slots, where each slot is half an hour in $[0, 47]$, and we get $7 * 48$ time slots for the whole week. For better representation ability, we embed departure time slots with matrix $W_\tau \in \mathbb{R}^{|N_\tau| \times d_{te}}$, where a departure time td is turned into low-dimensional vector $W_\tau[\tau(td)]$.

Spatio-temporal Correlations Encoding. We adopt a popular graph embedding method named node2vec [17] to pre-train and initialize all embeddings of W_x, to embed spatial correlations among edges. We adopt temporal graph [52, 57] to capture the temporal periodicity information, where each node denotes the embedding of either a neighboring time slot or a neighboring day. Besides, for travel time of each edge, there exists stable yet complicated factors, *e.g.,* capacity, popularity and traffic lights. We maintain embedding matrix $W_{tc} \in \mathbb{R}^{|N_\tau| \times d_{te}}$ to model these time-invariant factors for travel time generation, where we have travel time hidden state

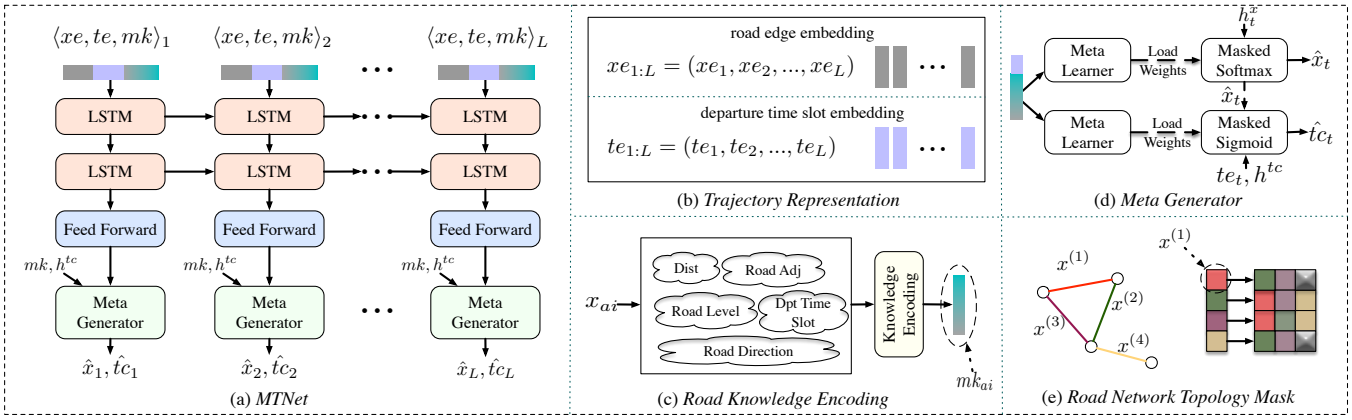


Figure 4: MTNet Overview

vector $h_x^{tc} = W_{tc}[x]$ for each $x \in E$. These embedding matrices are further learned together with MTNet (see Section 4).

Example 4: [Trajectory Representation] In step $t \in [1, L]$, based on $\hat{x}_{t-1}$ and $\hat{tc}_{t-1}$ output from step $t-1$, we compute $\hat{td}_t = \hat{td}_{t-1} + \hat{tc}_{t-1}$. Then we embed $\hat{x}_{t-1}$ and $\hat{td}_t$ are into $xe_t = W_x[\hat{x}_{t-1}]$ and $td_t = W_\tau[\tau(\hat{td}_t)]$ respectively, as input to MTNet (Figure 4 (a)). □

3.2 Trajectory Prefix Encoding

Road Knowledge Encoding. Despite embedded trajectory sequence itself, there exists semantic information of trajectory on the underlying road network, which contains significant trajectory distribution patterns. To discover these useful patterns, we collect road properties from the road maps and encode them into informative formats. Then, we employ the road knowledge encoding sub-module `MKEncoding` to concatenate encoded road edge properties and departure time information into road meta knowledge (or knowledge in short), which assembles the time varying routing patterns and serves to trajectory prefix encoding and the meta generator module. Specifically, we describe and encode the following road edge properties, a few of which are neglected in the literature (*e.g.,* road types and directions). (1) Road length, which mainly affects the travel time and routing strategies to avoid traffic lights, is normalized into Z-score format. (2) Road type (or level), chosen from *{residential street, tertiary road, secondary road, primary road, trunk road and motorway}*, which correlates to route edge planning, is encoded into one-hot format. (3) Geographical direction, from 0 to 2π, which mainly affects the following route edge selection, is also normalized into Z-score format. Besides, trigonometric value format, *e.g.,* sin, is appended for more effective representation. (4) Max speed, which corrects trajectory speed errors causes by GPS biases, is normalized into Z-score format. (5) Other properties, *e.g.,* oneway and living-street tags, are encoded into one-hot format.

Example 5:[Road Knowledge Encoding] In step $t \in [1, L]$ with current route edge $\hat{x}_{t-1}$ and departure time $\hat{td}_t$, suppose adjacent edge set of $\hat{x}_{t-1}$ is $\{x_{a1}, x_{a2}, \cdots, x_{ak}\}$, $x_{a1:ak}$ in short. It outputs knowledge mk_t = `MKEncoding` $(W_x[x_{t-1}],\ W_\tau[\tau(\hat{td}_t)])$ for encoding trajectory prefix, and mk_{ai} = `MKEncoding` $(W_x[x_{ai}],\ W_\tau[\tau(\hat{td}_t)])$, $x_{ai} \in x_{a1:ak}$, for the meta generator (Figure 4 (d)). □

Trajectory Prefix Encoding. Based on the spatio-temporal trajectory embedding and road knowledge for each route edge, we further encode the trajectory prefix to capture the long-term spatio-temporal correlation between the trajectory prefix and the trajectory suffix to be generated. Particularly, we adopt long short-term memories (LSTMs) – a typical kind of recurrent neural network that can effectively encode long-term dependency of sequence data, to encode the critical trajectory prefix. Besides, we use fully connected feedforward network following LSTMs to extract useful hidden features from encoded prefix for the next edge generation. Note that we adopt LSTMs for its effectiveness and succinctness, and more sophisticated recurrent neural modules that support step by step encoding and generation can also be applied here to further improve the encoding performance.

Example 6:[Trajectory Prefix Encoding] In step $t \in [1, L]$, based on $h_{t-1} \in \mathbb{R}^{d_h}$ for prefix $\hat{x}_{0:t-2}$ with $\hat{td}_{1:t-1}$, and input $\langle xe, te, z, mk \rangle_t$, `Recurrent` steps from h_{t-1} to h_t, *i.e.,* brings spatio-temporal features of $\hat{x}_{t-1}$ and td_t into the current state. With the following feedforward network, it outputs $h_t^x \in \mathbb{R}^{d_o}$ for $\hat{x}_t$ generation. □

3.3 Trajectory Generation with Meta Generator

There are hidden route planning patterns behind real-world urban trajectory generation on the underlying road network, where each road holds its own turning patterns according to its properties (or knowledge). Based on this observation, we design the core functional output module, *i.e.,* meta generator module (Figure 4 (c)). Leveraging the encoded road knowledge from Section 3.2, we propose a meta learning sub-module, named `MetaLearner`, to serve a family of neural weights for trajectory generation, where weights are optimized for each edge selection at any departure time slot.

Meta Learner. `MetaLearner` captures road network route planning patterns for each road edge based on the spatio-temporal knowledge from `MKEncoding`, and adapts and serves optimized settings (neural weights) for the output neural network, for each of the next-edge candidates. From the meta learning perspective, whether to select a specific edge candidate as the next edge to generate at a specific departure time slot is seen as an individual sub-task. There are different optimized output neural networks (with different neural weights) for each sub-task, *i.e.,* for each road edge and departure time slot, where experiences (patterns) learned from edges with

abundant real samples can be easily adapted to edges with less samples and improve the corresponding generation performance. MetaLearner is implemented by a single-layer fully connected feedforward network.

Example 7:[Meta Learner] For encoded spatio-temporal knowledge mk_{ai} from MKEncoding, meta learner sub-module outputs optimized (adapted) neural weights W_{ai} = MetaLearner(mk_{ai}). □

Output Neural Network. This sub-module is for generating route and travel time with MaskedSoftmax and MaskedSigmoid functions respectively (*i.e.,* softmax and sigmoid functions with topology mask in Figure 4 (e)). Specifically, for each step $t \in [1, L]$, suppose the next route edge candidates are $x_{a1:ak}$ and the departure time is $\hat{td}_t$. With prefix feature vector h_t^x and travel time hidden state vectors $h_{a1:ak}^{tc}$ of $x_{a1:ak}$, it works as follows.

- Route edge generation. For each candidate edge $x_{ai} \in x_{a1:ak}$, we get a score by the dot product of optimized neural weights W_{ai}^{soft} and prefix feature vector h_t^x. Then, all scores for adjacent edge candidates are turned into selection probabilities based on MaskedSoftmax function. Based on multinomial sampling over these probabilities, we randomly choose one candidate as the next generated edge, where there exists a stop flag (virtual edge) used for trajectory early stop.
- Travel time generation. Travel time of currently selected edge is produced by the dot product of the corresponding optimized neural weights W_{ai}^{sig} and travel time state vector h_{ai}^{tc}, where the result is passed through a sigmoid function to generate a rescaled travel time $tc_t \in [0, 1]$, for numerical stability.

Example 8:[Output Neural Network] For time step $t \in [1, L]$, it generates $\hat{x}_t$ = MaskedSoftmax ($W_{a1:ak}^{soft}, h_t^x$), *i.e.,* sampling based on softmax score $h_t^x \cdot W_{ai}^{soft}, i \in [1, k]$. It also generates $\hat{tc}_t$ = MaskedSigmoid($W_{a1:ak}^{sig}, h_{a1:ak}^{tc}, \hat{x}_t$), where $h_{a1:ak}^{tc} = W_{tc}[x_{a1:ak}]$. □

Example 9:[Trajectory Generation Dataflow] We show a dataflow example in Figure 5. $x^{(3)}$ with $\hat{tc}_1$, and h_1 for prefix ($x^{(1)}$) and $\hat{td}_{1:1}$ are given for $t = 1$. For $t = 2$, we input $\langle xe, te \rangle_2$ and mk_2 for $x^{(3)}$ and $\hat{td}_2$. $mk_{a1:a3}$ are computed for adjacent edges (Figure 5 (a)). h_2^x for prefix ($x^{(1)}, x^{(3)}$) and $\hat{td}_{1:2}$ is computed, where we update state to h_2 (Figure 5 (b)). Then, $W_{a1:a3}^{soft}$ and $W_{a1:a3}^{sig}$ are computed (Figure 5 (c)). Finally, $\hat{x}_2$ and $\hat{tc}_2$ are generated (Figure 5 (d)). □

3.4 Supporting Trajectory Applications

There are four mainstream applications on map-matched trajectories [47], *i.e.,* trajectory clustering (*e.g.,* popularity ranking), prediction (*e.g.,* trajectory suffix prediction), planning (*e.g.,* travel time estimation) and historical trajectory querying (*e.g.,* anomaly detection). MTNet can support the first three types of trajectory analysis applications except for querying historical trajectories, because the generative model learns the inherent statistical patterns rather than purely remembers historical trajectories, *e.g.,* it overlooks the occasional historical anomalies.

Trajectory Generation. MTNet learned from real big trajectory data can be used to generate synthetic yet representative trajectories that preserve the statistics in the original trajectory data. Thus we do not need to store and maintain the original large volume of trajectory data in normal use cases. We support trajectory generation in different granularities based on the superior model formulation in Equation 3, *i.e.,* generate samples for any subset of origin edges and any period of departure time. For example, given a trajectory dataset, MTNet can generate synthetic trajectories for each fixed origin, which achieves similar performance as the original data on real-world travel time estimation task (empirically evaluated in Section 5).

Directly Supporting Trajectory Applications. We formulate MTNet to be able to directly support statistical trajectory analysis without generating trajectories. We use the following applications as examples to show how to use MTNet to support real applications. (1) Trajectory popularity ranking. Given a set of trajectories, MTNet can be used to rank them by their time dependent popularity. It inputs trajectory route candidates $\{x_{0:L}\}$ and departure time condition td_1. It computes probabilities MTNet($\hat{x}_t = x_t | x_{0:t-1}, \hat{td}_{1:t}$), $\forall t \in [1, L]$, as rank values to compare time-dependent popularities. (2) Trajectory suffix prediction. Given a trajectory prefix (*e.g.,* an origin with departure time), MTNet searches the most promising suffix by maintaining promising candidates set according to generation probability in each step during the suffix generation process. (3)Time-dependent travel time prediction. Given a trajectory with a departure time, MTNet can directly estimate the travel time of each route edge. Specifically, the input is a route $x_{1:L}$ and departure time td_1 from x_1, and $x_{1:L}$ is padded with x_0 that is adjacent to x_1, as input condition. In step $t = 1$, we input x_0 and td_1, MTNet outputs $\hat{tc}_{a1:ak}$ of x_1, Then in step $t = 2$, we input x_1 and $\hat{td}_2 = td_1 + \hat{tc}_1$ and get $\hat{tc}_2$. We repeat above steps until we finish step $t = L$, and get the whole accumulated estimated travel time.

4 DIFFERENTIALLY PRIVATE MTNET

We introduce how to learn differentially private MTNet to effectively and efficiently model noisy trajectory data while preserving individual trajectory privacy. We set up the training loss function with two parts: learning trajectory route distribution and the corresponding travel time distribution. For trajectory route, we use negative log-likelihood (NLL) loss, shown in Equation 4, which is the accumulated loss for each time step $t \in [1, L]$. For travel time, we use mean absolute loss (MAE) in Equation 5, which is the accumulated travel time absolute error for each step $t \in [1, L]$. Note that MAE (rather than mean square error (MSE)) is used here for improving the resistance to travel time outliers because MAE pays more attention (gradients) to outliers that deviated from mean travel time, *e.g.,* when the vehicle idles or stops on roads for some reasons. We use a hyper-parameter λ to balance the influence of these two parts, in Equation 6, where θ denotes all trainable parameters, and we optimize θ to minimize the total loss $\mathcal{L}$.

$$\mathcal{L}_x = -\mathbb{E}_{x_{0:L}, td_{1:L} \sim p_{data}}[\sum_{t=1}^{L} \log \mathcal{G}_\theta(\hat{x}_t = x_t | x_{0:t-1}, td_{1:t})] \quad (4)$$

$$\mathcal{L}_t = \mathbb{E}_{x_{0:L}, td_{1:L}, tc_{1:L} \sim p_{data}}[\sum_{t=1}^{L} |tc_t - \mathcal{G}_\theta(\hat{tc}_t | x_{0:t}, td_{1:t})|] \quad (5)$$

$$\min_\theta \mathcal{L} = \mathcal{L}_x + \lambda \mathcal{L}_t \quad (6)$$

We adopt differentially private stochastic gradient decent method (DPSGD) [1] to learn MTNet, where real trajectory route and travel

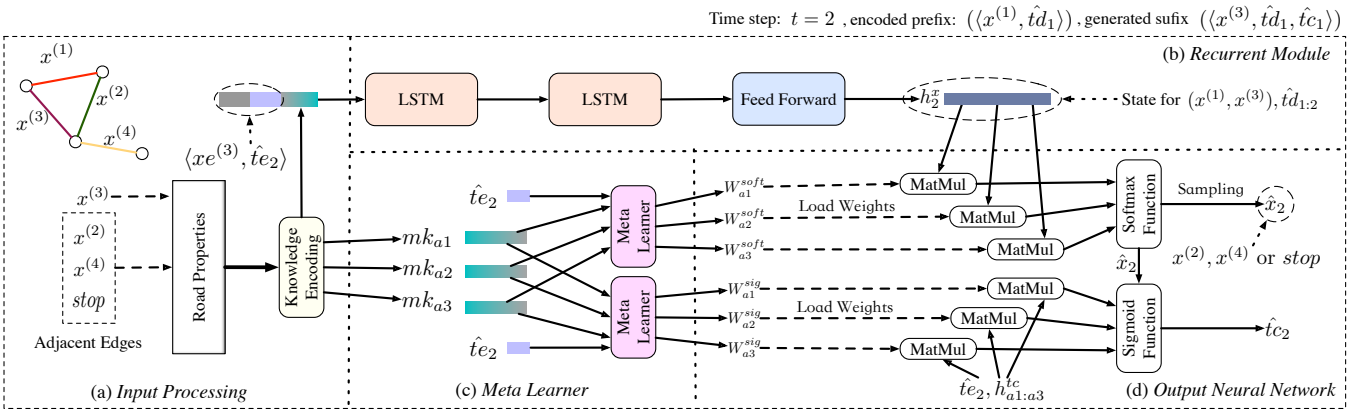


Figure 5: MTNet Trajectory Generation Dataflow Example

Algorithm 1: Learning MTNet Differential Privately

Input: Aligned trajectory routes: $\mathcal{X} = \{x_{0:L}\}$
Departure time sequences $\mathcal{TD} = \{td_{1:L}\}$
Rescaled trajectory travel time $\mathcal{TC} = \{tc_{1:L}\}$
Sample size B and gradient norm bound C

1 $X \leftarrow \langle \{x_{0:L-1}\}, \{td_{1:L}\} \rangle$;
2 $Y \leftarrow \langle \{x_{1:L}\}, \{tc_{1:L}\} \rangle$;
3 Initialize trainable parameters θ of MTNet;
4 **while** *stop criteria is not met* **do**
5 Uniformly sample $\langle X_B, Y_B \rangle$ from $\langle X, Y \rangle$;
6 $\mathcal{L} = \texttt{MTNet-Forward}(X_B, Y_B)$;
7 $g(i) \leftarrow \nabla_\theta \mathcal{L}[i]$, for $i \in [1, B]$; // compute gradient
8 $\bar{g}(i) \leftarrow g(i) / \max\left(1, \frac{||g(i)||_2}{C}\right)$; // clip gradient
9 $\tilde{g}(i) \leftarrow (\sum_i \bar{g}(i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}))/B$; // add noise
10 $\theta = \theta - \alpha \tilde{g}(i)$; // α is learning rate
11 **return** *MTNet;*

Algorithm 2: MTNet-Forward

Input: $X_B : \langle \{x_{0:L-1}\}, \{td_{1:L}\} \rangle$
$Y_B : \langle \{x_{1:L}\}, \{tc_{1:L}\} \rangle$

1 $\{xe_{1:L}\} = W_x[\{x_{0:L-1}\}]$; // embed route edge
2 $\{te_{1:L}\} = W_\tau[\{\tau(td_{1:L})\}]$; // embed departure time slot
3 $\{mk_{1:L}\} = \texttt{MKEncoding}(\{xe_{1:L}\}, \{te_{1:L}\})$;
4 Initialize $\mathcal{L}_x = \mathbf{0}$, $\mathcal{L}_t = \mathbf{0}$;
5 **for** $t \in [1, L]$ **do**
6 $\{h_t^x\} = \texttt{Recurrent}(\texttt{Concat}(\{\langle xe, te, mk \rangle_t\}))$;
7 $\{xe_{a1:ak}\} = W_x[\texttt{Adjacent}(\{x_{t-1}\})]$; // adjacent edges
8 $\{h_{a1:ak}^{tc}\} = \{W_{tc}[x_{a1:ak}]\}$;
9 $\{mk_{a1:ak}\} = \texttt{MKEncoding}\ (\{xe_{a1:ak}\}, te_t)$;
10 $\{W_{a1:ak}^{soft}\}, \{W_{a1:ak}^{sig}\} = \texttt{MetaLearner}(\{mk_{a1:ak}\})$;
11 $\{\hat{x}_t\} = \texttt{MaskedSoftmax}(\{W_{a1:ak}^{soft}\}, \{h_t^x\})$;
12 $\{\hat{tc}_t\} = \texttt{MaskedSigmoid}(\{W_{a1:ak}^{sig}\}, \{h_{a1:ak}^{tc}\}, \{\hat{x}_t\})$;
13 $\mathcal{L}_x = \mathcal{L}_x + \texttt{NLL}(\{\hat{x}_t\}, \{x_t\})$;
14 $\mathcal{L}_t = \mathcal{L}_t + \texttt{MAE}(\{\hat{tc}_t\}, \{tc_t\})$;
15 **return** $\mathcal{L}_x + \lambda \mathcal{L}_t$;

time are used privately to train and guide trajectory generation. Algorithm 1 shows the pseudo code. The input includes processed trajectory dataset, where each trajectory is aligned to length L, departure time for each route edge from the start point, and normalized travel time for each whole edge. Firstly, the input dataset is divided into input set X and output label set Y (lines 1-2). Then, all trainable parameters, except W_x and W_τ initialized with graph embedding, are initialized with Xavier initialization [15] (line 3). The next step is to learn MTNet privately with input set X and output label set Y until MTNet converges (lines 4-8). In each iteration, we uniformly sample a mini-batch B of trajectories and compute the forward loss (line 5-6). The gradients with respect to MTNet parameters are computed and clipped by L_2 norm with threshold C (lines 7-8). Gaussian noise is added to clipped gradients to privatize the access of trajectories, with privacy level parameter σ, before MTNet is updated with gradient decent (line 9-10). We return the learned privacy preserving MTNet (line 11). During the training process, we employ the advanced RDP accounting technique [34] for a tight estimation of privacy loss.

Algorithm 2 presents the explicit MTNet forward propagation procedure. Trajectory route edge labels are transformed into embedded representations (line 1). Departure time sequence is discretized into slots with function τ and also transformed into embedded representations (line 2). Spatio-temporal meta knowledge for input road edges are encoded (line 3). Mini-batch training loss $\mathcal{L}_x$ and $\mathcal{L}_t$ are initialized (line 4) for accumulation in each generation step (lines 5-14). In each step $t \in [1, L]$, the above three input parts are concatenated and passed through `Recurrent` module (line 6). The adjacent edges of current edge x_{t-1} are looked up and their travel time hidden states and corresponding meta knowledge are computed (lines 7-9). `MetaLearner` module produces corresponding meta weights (line 10) for generating the next edge and its travel time (lines 11-12), with the help of trajectory prefix feature vector h_t^x, h_t^{tc} from `Recurrent` module and meta knowledge of x_{t-1}'s adjacent edges. $\mathcal{L}_x$ and $\mathcal{L}_t$ are accumulated based on `NLL` and `MAE` loss respectively (lines 13-14). Finally, we get the mini-batch loss $\mathcal{L}$ by summing these two losses with a balancing factor λ (line 15).

Theorem 4.1. *Algorithm 1 guarantees differential privacy.*

Proof. For each training step (lines 4-10), the L2 gradient norm of a trajectory is bounded by C (*i.e.,* sensitivity) (line 8). Then, Gaussian noise guarantees (ϵ, δ)-DP with respect to the sampled mini-batch, *i.e.,* satisfies Equation 1, by setting $\sigma = \sqrt{2\ln(1.25/\delta)}/\epsilon$ [10]. Next, each sampling training step guarantees $(q\epsilon, q\delta)$-DP with respect to the whole data based on privacy amplification theorem [4], where q is the sampling ratio. Finally, the DP learning process of $\mathcal{G}$ guarantees DP trajectory generation and analysis, according to the post-processing property of DP [10]. Curious readers may refer to RDP [34] for a tight privacy loss accounting for overall training. □

Discussion. Intuitively, we protect individual privacy by learning population-level trajectory distribution statistics as more as possible and individual-level information as less as possible. By perturbing gradients with individual-level Gaussian noise, we randomize individual-level sensitive information without sacrificing the population-level statistics. Note that MTNet achieves high utility by applying DP over each whole trajectory sequence, while the literature perturbs coarse-grained location trajectory prefix modeled on geo-grids, which neglects road constraints and reduces data utility, especially for long trajectory with more accumulative noise.

5 EXPERIMENTS

5.1 Setup

Datasets. We used two real trajectory datasets: Chengdu and Xian[2], and there were 18.2 million and 11.6 million trajectories used for evaluation, respectively. Table 2 showed the details.

Validation Metrics. We used the aforementioned Jensen-Shannon divergence (JSD) measurement to capture and measure the distance between real test data and synthetic trajectory route distribution. Besides, we evaluated travel time generation error with MAE. Moreover, we evaluated the effectiveness of directly supporting trajectory applications with popularity ranking and suffix prediction tasks, and we evaluated the effectiveness of synthetic data based on travel time estimation task. Specifically, we formulated JSD measurement into route edge usage JSD (route JSD in short, Equation 7) and destination JSD (Equation 8).

- Route JSD. Origin conditioned route edge usage distribution quantifies detailed trajectory route edge distribution statistics. We formulated route usage distribution distance by comparing the usage percentage of each edge in real and synthetic trajectory data for each origin edge x_0, as shown in Equation 7. Note that the upper bound of route JSD was proportional to average trajectory sequence length rather than equal to 1, as there was more than one route edges in a trajectory. To evaluate route JSD, we generated equal-sized synthetic trajectories for each origin edge in test data, compared them with real ones based on JSD distance and computed average distance over all origin edges.
- Destination JSD. Destination distribution for each origin edge was also a crucial statistics for modeling trajectory distribution (*e.g.,* concerning traffic flow direction patterns), and synthetic data should keep the same destination distribution patterns with the input dataset. We used synthetic data from the above route JSD evaluation to further evaluate destination JSD, where we computed average destination JSD distance over all origin edges.

[2]https://gaia.didichuxing.com

Table 2: Details of the Datasets

Road Network	Chengdu	Xi'an
Timespan	10/01 - 11/30, 2016 & 2018	
# Edges	11,606	13,022
# Trajectories	18.2 Million	11.6 Million
Avg. Sequence Length	23	28

$$RouteJSD(x_0) = \frac{1}{2}[\mathbb{E}_{P_d(x \in x_{1:L}|x_0)} \log \frac{P_d(x \in x_{1:L}|x_0)}{M} + \mathbb{E}_{\mathcal{G}(x \in \hat{x}_{1:L}|x_0)} \log \frac{\mathcal{G}(x \in \hat{x}_{1:L}|x_0)}{M}] \quad (7)$$

$$DesJSD(x_0) = \frac{1}{2}[\mathbb{E}_{P_d(x_L|x_0)} \log \frac{P_d(x_L|x_0)}{M} + \mathbb{E}_{\mathcal{G}(\hat{x}_L|x_0)} \log \frac{\mathcal{G}(\hat{x}_L|x_0)}{M}] \quad (8)$$

where P_d is real trajectory distribution, $P_d(x_L|x_0)$ is origin conditioned probability that x_L is the destination, $P_d(x \in x_{1:L-1}|x_0, x_L)$ is OD conditioned probability that x is used in trajectories, and $M = \frac{1}{2}(P_d + \mathcal{G})$. As JSD measures the distance between two distributions, *the smaller the output value, the better the trajectory modeling performance.* Besides, x_L (or $\hat{x}_L$) above denoted the real destination edge rather than padded stop flag used for alignment.

Baselines. We compared MTNet with the following baselines:

- **Ngram** [42]. Trajectory route generation could be modeled by classic Markov chain [37]. We implemented a third-order Markov chain, named Ngram model. Note that continuous travel time information could not be supported by Ngram.
- **AutoSTG** [39]. AutoSTG was a spatio-temporal graph convolutional framework for traffic prediction. We adapted it to support trajectory generation with a 2-layer LSTMs, where the number of steps for diffusion convolution was set to 2 [27].
- **CSSRNN** [50]. CSSRNN was designed for map-matched trajectory modeling and generation task. It leveraged masked LSTMs for trajectory route edge generation. We extended it to support extra travel time generation.

Our Methods. We implemented two methods MTNet and TNet. The former used road knowledge and meta learning components and the latter did not use them.

- **MTNet**. MTNet was the full version of our proposed model with all optimization techniques as discussed in Section 3.
- **TNet**. TNet was implemented from MTNet by removing road knowledge and meta learning. The reason of evaluating TNet was to validate the superiority of spatio-temporal modeling approach with temporal graph and the effectiveness of our trajectory sequence representation learning over baselines.

Default Framework Settings. Trajectory sequence length was set as $L = 20$, according to the average trajectory length of the datasets. The number of training epochs was set to 30, with early-stop rule based on trajectory route JSD and patience of 5 epochs. In addition, we applied DP of fixed privacy budget ($\epsilon = 1, \delta = 10^{-5}$) to MTNet. Hyper-parameter λ was set to 0.1. The number of layers for LSTMs and feedforward network was set to 2 and 1, respectively. The dimension of parameters were set as follows: $d_z = 16$, $N_\tau = 48$, $d_{te} = 32$, $d_{xe} = 128$, $d_h = 128$ and $d_o = 32$.

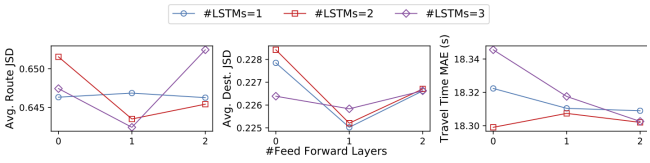


Figure 6: Varying Number of Layers (Chengdu)

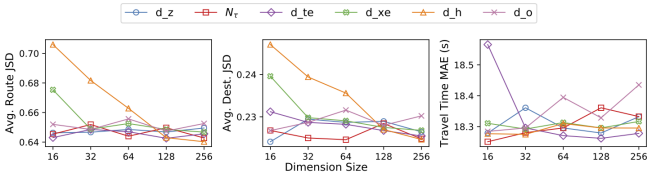


Figure 7: Varying Dimension of Parameters (Chengdu)

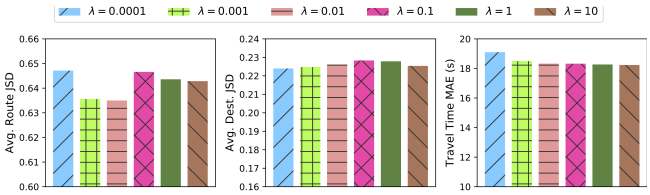


Figure 8: Varying Hyper-parameter λ (Chengdu)

Road Network Extraction and Processing. Road networks were obtained from OpenStreetMap. Firstly, nodes, edges and their properties such as geographical geometry and road level were parsed and modeled into a directed graph, where unreachable edges from the main connected component were removed. Road length and geographical direction were computed based on its GPS coordinates.

Trajectory Data Pre-Processing. Geo-coordinates and timestamps of raw trajectories were map matched to road network, based on the FMM algorithm [32, 51]. There are two issues for map matching. First, as trajectories may not start and end at edge joints (e.g., x_1 and x_L may not be at edge joints), we compute the corresponding travel time based on the offset ratio and omit offset details for ease of presentation. Second, timestamps of the raw trajectory may not be aligned with the road segments, and thus we cannot directly calculate departure time td_i and travel time tc_i. To handle this, we adopt linear interpolation based on road length to calculate the departure time and travel time for each route edge. Trajectories with lengths no larger than L were collected to compose the input dataset, where trajectories with sequence length smaller than L were padded with a stop flag. Continuous travel time values are are normalized into Z-score format for numerical stability, which can be scaled back to seconds. Trajectories were shuffled and split into training data (80%) and test data (20%).

Environment Settings. Experiments were conducted on a CentOS 7.7 server with Cuda 10.0 and a Tesla V100 GPU with 32GBs memory. Road network extraction and data pre-processing were implemented in C++11 with Clang compiler. All trajectory methods were implemented in Python 3.7 with PyPorch 1.2.

5.2 Evaluation on Different Settings

Evaluation on Number of Layers. We evaluated the impact of the number of layers for MTNet, *i.e.,* number of LSTMs and number of feedforward layers after LSTMs. We varied #LSTMs in $\{1, 2, 3\}$, and number of feedforward layers in $\{0, 1, 2\}$. Figure 6 showed the results. We observed that 2 layers of LSTMs were enough for trajectory sequence encoding. And one single layer of feedforward network achieved the best performance.

Table 3: Model Sizes Comparison

Dataset	Chengdu / Xi'an				
Metric	Ngram	AutoSTG	CSSRNN	TNet	MTNet
#Params (M)	-/-	1.6/1.8	1.6/1.8	1.8/2.0	1.6/1.8
Space (MBs)	5.8/4.0	400+	7.2/7.8	8.2/7.9	7.3/7.9

Evaluation on Dimension of Parameters. We evaluated the impact of parameter dimensions on modeling trajectory data. Based on the default settings, we varied optional Gaussian noise input with dimension d_z to simulate random deviations and to ease overfitting, departure time slot and travel time hidden state embedding dimension d_{te}, route edge embedding dimension d_{xe}, encoded trajectory prefix hidden state dimension d_h and output neural weight dimension d_o of `Recurrent` in $\{16, 32, 64, 128, 256\}$. We also evaluated number of departure time slots $N_\tau \in \{12, 24, 48, 96, 192\}$. Figure 7 showed the results (x-axis ticks were omitted for number of departure time slots). In summary, we set each parameter dimension: $d_z = 16$, $N_\tau = 48$, $d_{te} = 32$, $d_{xe} = 128$, $d_h = 128$ and $d_o = 32$.

Evaluation on Balancing Route and Travel Time Error. We evaluated the impact of hyper-parameter λ which balanced the loss between trajectory route and travel time during training, as shown in Equation 6. λ was varied in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$. Figure 8 showed the results. We observed that $\lambda = 10^{-1}$ was a good balance choice. Intuitively, larger λ leads to lower travel time `MAE`, but as the travel time was much simpler than trajectory route distribution modeling, $\lambda = 10^{-1}$ made a considerable choice.

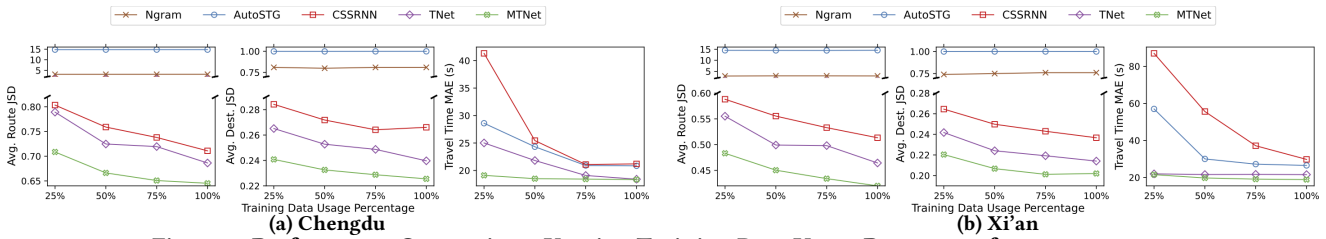
5.3 Performance Comparison with Baselines

To measure the effectiveness of modeling real trajectory data and generating synthetic data with close-to-real-data statistics, we used the origin edge and departure time conditions from the real data and conducted trajectory generation evaluations. We compared MTNet and TNet with baselines and evaluated the generation performance on trajectory route and destination JSD (*the smaller, the better*), and the corresponding travel time `MAE`. Table 3 showed the model sizes (or # parameters) used during the following evaluations, where all methods kept approximately equal sizes for fair comparisons. Note that AutoSTG with graph convolution needed to input and maintain graph adjacency information, thus it used larger space.

Performance Comparison. Based on the default settings, we evaluated performance and compared with baselines by varying sampled training data size in 25%, 50%, 75%, 100%. Figure 9 showed the results, where Ngram did not support travel time generation and hence did not show up in the corresponding sub-figure (also applied to following evaluations). We made the following observations.

(1) MTNet significantly outperformed other models on trajectory route and destination modeling. This was because MTNet effectively represented, encoded and modeled trajectory data with road knowledge based meta learning. Besides, TNet improved on trajectory route and destination distribution modeling compared to baselines, because trajectory route distributions were time dependent and thus introducing spatio-temporal representation of trajectory improved trajectory modeling ability compared to pure spatial modeling formulation in baselines.

(2) MTNet and TNet outperformed baselines on travel time `MAE`. This was because edge travel time was not constant but dependent on departure time, and hence spatio-temporal representation based and departure time conditioned modeling approaches of MTNet



(a) Chengdu (b) Xi'an

Figure 9: Performance Comparison: Varying Training Data Usage Percentage from 25% to 100%

Table 4: Trajectory Scalability for Varying Length L

Dataset		Chengdu/Xi'an			
Metric	L	20	40	60	80
Avg. Route JSD	Ngram	3.12/3.00	7.37/7.07	9.26/10.1	9.64/11.5
	AutoSTG	14.7/14.3	28.4/26.7	38.3/40.1	46.5/48.1
	CSSRNN	0.71/0.51	2.10/1.85	2.74/2.90	2.90/3.32
	TNet	0.68/0.46	2.05/1.79	2.70/2.83	2.88/3.26
	MTNet	**0.64**/**0.42**	**2.02**/**1.71**	**2.60**/**2.67**	**2.82**/**3.18**
Avg. Des. JSD	Ngram	0.81/0.76	0.89/0.85	0.90/0.88	0.90/0.89
	AutoSTG	1.00/0.99	1.00/1.00	1.00/1.00	1.00/1.00
	CSSRNN	0.27/0.24	0.44/0.46	0.45/0.53	0.45/0.54
	TNet	0.24/0.21	0.40/0.43	0.42/0.49	0.43/0.51
	MTNet	**0.22**/**0.20**	**0.39**/**0.41**	**0.42**/**0.48**	**0.43**/**0.50**
Travel Time MAE(s)	AutoSTG	20.5/25.6	19.4/20.0	19.4/18.8	19.2/18.6
	CSSRNN	21.2/29.7	20.7/21.1	20.1/19.2	19.6/18.8
	TNet	18.8/21.5	17.1/21.7	16.8/20.7	**16.7**/20.5
	MTNet	**18.3**/**18.7**	**17.1**/**17.2**	**16.8**/**16.6**	16.8/**16.4**

and TNet improved the quality. Besides, MTNet further reduced travel time MAE compared to TNet, especially when the training data volume was small, which showed that road knowledge was helpful on modeling travel time. This was because road properties such as length and capacity were effective to travel time modeling.
(3) All methods performed better with more training data but MTNet was much less data hungry and had better scalability on training data volume compared to other methods, *i.e.,* suffered less performance loss when reducing training data. It was because models learned more sufficiently and improved generalization ability with more training data, and MTNet with meta learning had highly adaptive modeling ability which learned inherent trajectory generation patterns that migrated among route edges. Hence, MTNet eased the data-hungry issue, *i.e.,* using less training samples and less computing resource while getting more stable performance.
(4) The performance difference across different methods was smaller on Xi'an dataset. This was because Xi'an dataset had simpler road network topology and less trajectory data, and thus it was easier to model compared to Chengdu dataset.
(5) Ngram and AutoSTG performed bad on modeling trajectory route and destination distribution. For Ngram with Markov property, the generation process depended on a few preceding edges, which was hard to support trajectory modeling with long-term correlations (or dependencies) among route edges. Although AutoSTG was proved effective on modeling time varying graph attributes owned by all nodes or edges, it failed on modeling map-matched trajectories because trajectories had highly skewed distribution, where graph convolutional models were hard to converge.

Trajectory Length Scalability Comparison. We varied trajectory length threshold L in $\{20, 40, 60, 80\}$, and evaluated modeling scalability on L for all methods and compared their modeling ability. Table 4 showed the results. We made the following observations.

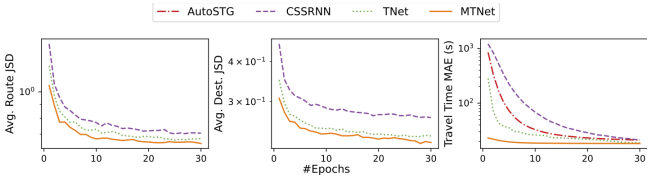


Figure 10: Training Efficiency Comparison (Chengdu)

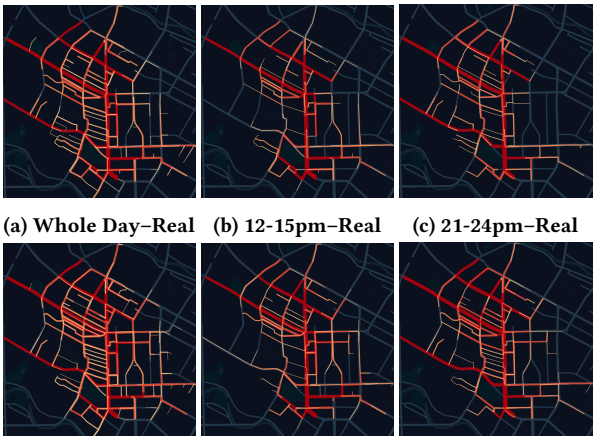
(1) MTNet outperformed other methods for all L settings on trajectory route and destination modeling and scaled well on varying trajectory length. This was because spatio-temporal trajectory representation, road knowledge and the meta learning module were based on single route edge rather than the whole trajectory sequence with had varying lengths.
(2) MTNet and TNet outperformed baselines on travel time MAE. This was because travel time was mainly influenced by hidden factors of each edge and departure time information which was captured by effective spatio-temporal trajectory representation strategies that both used in MTNet and TNet.
(3) Trajectory route and destination JSD increased along with increasing L. This was because trajectory distribution space grew exponentially with respect to trajectory sequence length, *i.e.,* there were more potential roads and destinations involved.
(4) Average travel time MAE decreased along with increasing L. This was because longer trajectory sequences tended to have more short road segments with small travel time, which reduced the average error of single road segment.

Training Efficiency Comparison. We visualized the average route JSD, destination JSD, and travel time MAE during training based on the more challenging Chengdu dataset (Xi'an dataset with similar tendency was omitted). Figure 10 showed the results, where AutoSTG that failed on modeling trajectory route and destination with highly skewed distribution was omitted. We made the following observations.
(1) MTNet learned much faster than other methods. The convergence rate of MTNet on route JSD, destination JSD, and corresponding travel time were all fastest. This was because road network knowledge based meta learning of MTNet was quite efficient, *i.e.,* it was able to extract effective routing and distribution patterns from well represented and encoded trajectory data.
(2) Evaluation loss of TNet also decreased faster than baselines on all three metrics. It verified that spatio-temporal trajectory representation with temporal graph to capture temporal periodicity information was more effective to model real-world trajectory data with time varying patterns.

Utility Loss Evaluation with Differential Privacy. We evaluated the utility loss brought by achieving DP with fixed privacy budget ($\epsilon = 1, \delta = 10^{-5}$). Table 5 showed the results. We observed that the utility loss brought by DP was negligible. This was because map-matching reduced privacy sensitivity of trajectories, *i.e.,* we had abundant map-matched trajectories with high redundancy

Table 5: Utility Loss of MTNet with ($\epsilon = 1, \delta = 10^{-5}$)-DP

Dataset	Chengdu / Xi'an		
Metric	Avg. Route JSD	Avg. Des. JSD	Travel Time MAE (s)
MTNet-DP	0.64/0.42	0.22/0.20	18.3/18.7
Without-DP	0.63/0.42	0.22/0.20	18.5/18.9



(a) Whole Day–Real (b) 12-15pm–Real (c) 21-24pm–Real

(d) Whole Day–Fake (e) 12-15pm–Fake (f) 21-24pm–Fake

Figure 11: Origin-fixed Trajectory Distribution (Chengdu)

for training and hence the statistical distribution could be fully captured with DP guarantee. Besides, Gaussian mechanism eased the overfitting of road travel time modeling.

Note that the above performance gain of MTNet based on JSD statistics may not look very high compared to the following case studies. This was because the marginal distribution (*i.e.,* on destination and route edge) naturally weakens the distribution differences as the distribution space is much smaller, while the real performance differences between methods were larger as errors were amplified (or accumulated) on prefix-dependent trajectory sequences.

5.4 Application Cases Study

Trajectory Generation in Different Granularities. We randomly chose an origin edge from Chengdu dataset and generated the same number of trajectories, whose departure time was consistent with real ones. We visualized trajectories during the whole day, $12-15pm$ and $21-24pm$ departure time slots for both the real data and synthetic samples generated by MTNet. Figure 11 showed the results, where edge usage density was described by edge color and line width, *i.e.,* from low density with light white and thin width towards higher density with deeper red and bolder width. We made the following observations.

(1) Synthetic trajectories by MTNet resembled the distribution of real ones, *i.e.,* real trajectories in the upper side had indistinguishable route density as synthetic ones in the lower side. Note that a few occasionally generated tail edges were brought by learned road turning patterns, which could be removed as their rank (*i.e.,* density) were orders of magnitude lower than normal ones.

(2) Trajectories had different spatio-temporal distributions for different departure time slots, which were accurately captured by MTNet. In particular, real trajectory distribution was different between $12-15pm$ and $21-24pm$, and synthetic trajectories by MTNet effectively captured the time varying distribution changes. For example, the street towards residential neighborhoods in the lower-left corner, was bold (*i.e.,* busy) at night hours $21-24pm$, but thin during daily time $12-15pm$.

Table 6: Popularity Ranking Accuracy (%)

	Ngram	AutoSTG	CSSRNN	TNet	MTNet
Chengdu	79.0	73.1	82.9	84.1	**87.3**
Xi'an	83.7	76.8	86.5	87.6	**90.1**

Table 7: Popularity Ranking Cases (Chengdu)

	Real Route Usage (%)/Popularity Rank			
Route	A	B	C	D
9 am	54/0.12	22/0.09	18/0.04	6/0.01
22 pm	81/0.29	14/0.07	0/0	5/0.03

Trajectory Popularity Ranking. We randomly chose 1,000 OD pairs and sampled 10 departure time slots for each OD pair from the test data. We randomly sampled two different trajectory routes for each fixed time slot and OD pair, with their occurrence frequency collected as popularity ground truth. We evaluated popularity ranking consistency with these 10,000 trajectory pairs and compared the average accuracy with baselines (trajectory popularity ranking was described in Section 3.4). Table 6 showed the results. We observed that MTNet achieved 4% higher ranking accuracy than baselines. This was because popularity ranking was affected by trajectory route distribution, and MTNet outperformed baselines on modeling route distribution of the original data (see Figure 9).

To reveal more details on time-dependent popularity ranking, we randomly chose an OD edge pair from Chengdu dataset and analyzed its viable routes. We estimated expected popularities based on learned MTNet, and showed the real route usage and corresponding rank values in Table 7. We observed that for time slots $9pm$ and $22pm$, the rank order was $A \rightarrow B \rightarrow C \rightarrow D$ and $A \rightarrow B \rightarrow D \rightarrow C$ respectively, consistent with the original route usage. And the rank value difference ratio between routes was related to the route usage ratio in real dataset. It was worth noting that if the route was not used in the original dataset, MTNet outputted a very small value closed to zero, *e.g.,* route C at $22pm$. Besides, it showed the time-varying of route popularity captured by MTNet. For morning peak $9am$, several routes for the same OD pair shared the traffic flow, which was reflected by close rank values of A, B and C. However, for late-night $22pm$, traffic mostly passed through the primary roadway A, with a rank value 0.29 that was much larger than other routes.

Trajectory Suffix Prediction. We randomly sampled 10,000 trajectory prefixes of length 5, and for each trajectory prefix with a fixed departure time slot, we collected the most promising suffix with the largest occurrence frequency from the test data. We evaluated the suffix prediction performance (as described in Section 3.4) and compared it with baselines, where beam search[12] was used to maintain promising edge candidates set during the generation process. Table 8 showed the results. We observed that MTNet achieved the best prediction accuracy on both datasets (5% higher than baselines), which was consistent with the route and destination JSD evaluation results.

In particular, we focused on routes C and D from Table 7 that both based on route prefix labeled as $\langle 3020, 3021, \cdots 3088\rangle$. For both departure time in $9am$ and $22pm$, we computed the expected suffix with the maximum probability based on MTNet, *i.e.,* $\langle 4722, 4711, 6314, -\rangle$ (*i.e.,* suffix of C) and $\langle 2980, 6314, -\rangle$ (*i.e.,* suffix of D), where $-$ was the stop flag. We observed that based on the long-term spatio-temporal encoding ability of MTNet, the predicted suffix was also time dependent and consistent with the rank values in Table 7.

Trajectory Travel Time Estimation. We evaluated synthetic trajectory data effectiveness on a popular trajectory application named

Table 8: Suffix Prediction Accuracy (%)

	Ngram	AutoSTG	CSSRNN	TNet	MTNet
Chengdu	35.7	56.3	61.6	62.8	**66.5**
Xi'an	25.7	60.1	65.3	66.6	**70.3**

Table 9: Travel Time Estimation Error (MAE)

	AutoSTG	CSSRNN	TNet	MTNet	Real Data
Chengdu	137s	157s	22.0s	**13.3s**	10.9s (27%)
Xi'an	144s	159s	55.1s	**14.1s**	11.2s (22%)

travel time estimation, which could also be directly supported with MTNet as described in Section 3.4. It input a trajectory route and estimated the travel time of each route edge. We split the real trajectory datasets into training part (80%) and test part (20%). We built a two-layer LSTM model for the travel time estimation task, where edge embedding and hidden state dimension were set to 128. We trained models on the task for 10 epochs: one with real training data and others with equal-sized synthetic data generated based on learned AutoSTG, CSSRNN, TNet and MTNet and respectively. We evaluated these learned models on the test data. Table 9 showed the results. We made the following observations.

(1) On synthetic data, MTNet outperformed baselines and achieved close-to-real data performance (with only 2 – 3 seconds difference). It showed that MTNet was more robust to model and recover trajectory (or its travel time) data than baselines because MTNet made full use of spatio-temporal road network knowledge based on deep meta learning, which led to outstanding generalization ability.

(2) AutoSTG and CSSRNN performed 3 – 10x worse compared to MTNet and TNet. It showed that spatio-temporal trajectory representation and modeling approach was much more effective for modeling continuous yet noisy time dependent travel time data.

6 RELATED WORK

6.1 Trajectory Modeling and Generation

Markov Models. Intuitively, sequential trajectories can be modeled and generated with Markov models, which includes Markov chains [43], hidden Markov models [3], and Markov decision process [58]. Although these methods are able to model short term transition probabilities which capture local sub-trajectories, they are limited for generating complicated length varying trajectories [44], because short-term memory is not sufficient for modeling complex trajectories with long-term dependencies.

Spatio-tmporal Graph Convolutional Networks. Graph convolutional network (GCN) is effective to model local correlations of graph node or edge features[24]. As road networks can be seen as graph with time varying features, spatio-tmporal graph convolutional networks (STGCN)[55] and graph attention techniques[59] has became prevalent on modeling and predicting time varying characteristics such as road traffic. In particular, AutoSTG[39] is shown promising on prediction tasks of time varying attributes for graph node or edge. However, skewed map-matched trajectories are high dimensional with respect to graph edges. It is hard for AutoSTG to model skewed trajectories like common edge attributes.

Recurrent Neural Networks (RNNs). RNN is effective in sequence generation problems [16], especially in NLP [13]. For example, Long-Short Term Memory [20] can maintain long-term dependency, which is useful for long trajectory sequence modeling. Consequently, it is adopted to model and generate trajectory in CSSRNN [50]. Nevertheless, CSSRNN neglects the essential temporal information to model time-varying distribution statistics and can only generate spatial trajectory road sequences. Furthermore, it overlooks the important underlying road knowledge (*e.g.,* road capacity and direction) which contains useful routing patterns to model high-dimensional trajectory data with skewed distribution.

Generative Adversarial Nets (GANs). GANs are popularly applied to spatio-temporal data [14], including trajectory generation [38], prediction [25], spatio-temporal events [23], time series imputation[33], etc. [29] proposed to use GAN-based techniques to generate location trajectories which preserve the summary properties of real trajectory data for analysis applications. However, as real map-matched trajectories are road network constrained, this approach is not applicable to generate map-matched trajectories.

Traffic Simulators. Extensive simulators on simulating traffic [2, 6, 9, 11, 22, 26, 31, 35, 56] are designed to simulate traffic with user-defined properties, which are used for benchmarking and analyzing traffic strategies and behaviors of moving objects with given conditions [36]. Different from traffic simulators, we model an existing trajectory dataset with close-to-real-data statistics, and further generate trajectories for or directly supporting downstream trajectory applications.

6.2 Meta Learning in Spatio-temporal Data

Meta learning is applied to spatio-temporal data recently [21, 46]. For example, ST-MetaNet [40, 41] predicts urban traffic with a meta learning based sequence-to-sequence architecture, using meta graph attention networks and meta RNNs. [54] studied the meta-learning paradigm for spatio-temporal data from multiple cities and utilized the long-period data for multiple city transfer learning tasks on traffic and water quality prediction. However, the regular mechanism of meta learning is splitting dataset into few-shot training and testing sub-datasets for meta and regular learning tasks respectively. For trajectory generation, high dimensional trajectory data is unique when compared to each other, *i.e.,* no explicit trajectory level patterns that can be directly transferred for classic meta learning. Thus we borrow the learning to learn idea of meta learning (*i.e.,* generalization), and leverage the underlying spatio-temporal road properties as meta knowledge to ideally support trajectory distribution patterns adaptation among route edges.

7 CONCLUSION

We proposed a deep meta learning based generative model, named MTNet, to support trajectory generation and analysis simultaneously with privacy guarantee. Extensive experiments showed that MTNet modeled real trajectory data with high utility, preserving original statistics and supporting real-world applications while keeping privacy. Future research directions include (1) transferring learned trajectory generation patterns to new road networks and (2) improving the adaptability of MTNet for new data.

ACKNOWLEDGMENTS

The corresponding author is Guoliang Li. This work was supported by NSF of China (61925205, 62232009, 62072261), BNRist, Huawei, and TAL education.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *SIGSAC*. 308–318.

[2] Michael Balmer, Marcel Rieser, Konrad Meister, David Charypar, Nicolas Lefebvre, and Kai Nagel. 2009. MATSim-T: Architecture and simulation times. In *Multi-agent systems for traffic and transportation engineering*. IGI Global, 57–78.

[3] Mitra Baratchi, Nirvana Meratnia, Paul JM Havinga, Andrew K Skidmore, and Bert AKG Toxopeus. 2014. A hierarchical hidden semi-Markov model for modeling mobility data. In *UbiComp*. 401–412.

[4] Amos Beimel, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. 2010. Bounds on the sample complexity for private learning and private data release. In *Theory of Cryptography Conference*. Springer, 437–454.

[5] Tolga Bektaş, Emrah Demir, and Gilbert Laporte. 2016. Green vehicle routing. In *Green transportation logistics*. Springer, 243–265.

[6] Thomas Brinkhoff. 2002. A framework for generating network-based moving objects. *GeoInformatica* 6, 2 (2002), 153–180.

[7] Chao Chen, Yan Ding, and et al. 2019. TrajCompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change. *TITS* 21, 5 (2019), 2012–2028.

[8] Zaiben Chen, Heng Tao Shen, and Xiaofang Zhou. 2011. Discovering popular routes from trajectories. In *ICDE*. IEEE, 900–911.

[9] Shaojie Dai, Yanwei Yu, Hao Fan, and Junyu Dong. 2022. Spatio-Temporal Representation Learning with Social Tie for Personalized POI Recommendation. *Data Sci. Eng.* 7, 1 (2022), 44–56. https://doi.org/10.1007/s41019-022-00180-w

[10] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407.

[11] Ju Fan, Guoliang Li, Lizhu Zhou, Shanshan Chen, and Jun Hu. 2012. SEAL: Spatio-Textual Similarity Search. *Proc. VLDB Endow.* 5, 9 (2012), 824–835. https://doi.org/10.14778/2311906.2311910

[12] Markus Freitag and Yaser Al-Onaizan. 2017. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806* (2017).

[13] Nan Gao, Hao Xue, Wei Shao, Sichen Zhao, Kyle Kai Qin, Arian Prabowo, Mohammad Saiedur Rahaman, and Flora D Salim. 2020. Generative Adversarial Networks for Spatio-temporal Data: A Survey. *arXiv preprint arXiv:2008.08903* (2020).

[14] Nan Gao, Hao Xue, Wei Shao, Sichen Zhao, Kyle Kai Qin, Arian Prabowo, Mohammad Saiedur Rahaman, and Flora D Salim. 2022. Generative adversarial networks for spatio-temporal data: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)* 13, 2 (2022), 1–25.

[15] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*. 249–256.

[16] Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* (2013).

[17] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. 855–864.

[18] Mehmet Emre Gursoy, Ling Liu, Stacey Truex, and Lei Yu. 2018. Differentially private and utility preserving publication of trajectory data. *TMC* 18, 10 (2018), 2315–2329.

[19] Xi He, Graham Cormode, and et al. 2015. DPT: differentially private trajectory synthesis using hierarchical reference systems. *PVLDB* 8, 11 (2015), 1154–1165.

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

[21] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. 2020. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439* (2020).

[22] Huiqi Hu, Guoliang Li, Zhifeng Bao, Jianhua Feng, Yongwei Wu, Zhiguo Gong, and Yaoqiang Xu. 2016. Top-k Spatio-Textual Similarity Join. *IEEE Trans. Knowl. Data Eng.* 28, 2 (2016), 551–565. https://doi.org/10.1109/TKDE.2015.2485213

[23] Guangyin Jin, Qi Wang, Xia Zhao, Yanghe Feng, Qing Cheng, and Jincai Huang. 2019. Crime-GAN: A Context-based Sequence Generative Network for Crime Forecasting with Adversarial Loss. In *IEEE BigData*. IEEE, 1460–1469.

[24] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[25] Vineet Kosaraju, Amir Sadeghian, Roberto Martín-Martín, Ian Reid, Hamid Rezatofighi, and Silvio Savarese. 2019. Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks. In *NIPS*. 137–146.

[26] Guoliang Li, Jianhua Feng, and Jing Xu. 2012. DESKS: Direction-Aware Spatial Keyword Search. In *ICDE*. 474–485. https://doi.org/10.1109/ICDE.2012.93

[27] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2017. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926* (2017).

[28] Qiang Liu, Shu Wu, and et al. 2016. Predicting the next location: A recurrent model with spatial and temporal contexts. In *AAAI*.

[29] Xi Liu, Hanzhou Chen, and Clio Andris. 2018. trajGANs: Using generative adversarial networks for geo-privacy protection of trajectory data (Vision paper). In *Location Privacy and Security Workshop*. 1–7.

[30] Cheng Long, Raymond Chi-Wing Wong, and HV Jagadish. 2013. Direction-preserving trajectory simplification. *PVLDB* 6, 10 (2013), 949–960.

[31] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. 2018. Microscopic traffic simulation using sumo. In *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE, 2575–2582.

[32] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. 2009. Map-matching for low-sampling-rate GPS trajectories. In *SIGSPATIAL*. 352–361.

[33] Yonghong Luo, Xiangrui Cai, Ying Zhang, Jun Xu, et al. 2018. Multivariate time series imputation with generative adversarial networks. In *NIPS*. 1596–1607.

[34] Ilya Mironov. 2017. Rényi differential privacy. In *CSF*. IEEE, 263–275.

[35] Kai Nagel, Paula Stretz, Martin Pieck, Rick Donnelly, and Christopher L Barrett. 1997. TRANSIMS traffic flow characteristics. *arXiv preprint adap-org/9710003* (1997).

[36] Johannes Nguyen, Simon T Powers, Neil Urquhart, Thomas Farrenkopf, and Michael Guckert. 2021. An overview of agent-based traffic simulators. *Transportation research interdisciplinary perspectives* 12 (2021), 100486.

[37] James R Norris. 1998. *Markov chains*. Number 2. Cambridge university press.

[38] Kun Ouyang, Reza Shokri, David S Rosenblum, and Wenzhuo Yang. 2018. A Non-Parametric Generative Model for Human Trajectories.. In *IJCAI*. 3812–3817.

[39] Zheyi Pan, Songyu Ke, Xiaodu Yang, Yuxuan Liang, Yong Yu, Junbo Zhang, and Yu Zheng. 2021. AutoSTG: Neural Architecture Search for Predictions of Spatio-Temporal Graph. In *Proceedings of the Web Conference 2021*. 1846–1855.

[40] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban traffic prediction from spatio-temporal data using deep meta learning. In *SIGKDD*. 1720–1730.

[41] Zheyi Pan, Wentao Zhang, Yuxuan Liang, Weinan Zhang, Yong Yu, Junbo Zhang, and Yu Zheng. 2020. Spatio-Temporal Meta Learning for Urban Traffic Prediction. *TKDE* (2020).

[42] Alexandre Papadopoulos, Pierre Roy, Jean-Charles Régin, and François Pachet. 2015. Generating all possible palindromes from ngram corpora. In *IJCAI*.

[43] Reza Shokri, George Theodorakopoulos, Jean-Yves Le Boudec, and Jean-Pierre Hubaux. 2011. Quantifying location privacy. In *symposium on SP*. IEEE, 247–262.

[44] Mudhakar Srivatsa, Raghu Ganti, Jingjing Wang, and Vinay Kolar. 2013. Map matching: Facts and myths. In *SIGSPATIAL*. 484–487.

[45] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *IJUFKS* 10, 05 (2002), 557–570.

[46] Joaquin Vanschoren. 2018. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548* (2018).

[47] Sheng Wang, Zhifeng Bao, J Shane Culpepper, and Gao Cong. 2020. A Survey on Trajectory Data Management, Analytics, and Learning. *arXiv preprint arXiv:2003.11547* (2020).

[48] Yong Wang, Guoliang Li, and Nan Tang. 2019. Querying shortest paths on time dependent road networks. *PVLDB* 12, 11 (2019), 1249–1261.

[49] Ling-Yin Wei, Wen-Chih Peng, and Wang-Chien Lee. 2013. Exploring pattern-aware travel routes for trajectory search. *TIST* 4, 3 (2013), 1–25.

[50] Hao Wu, Ziyang Chen, Weiwei Sun, Baihua Zheng, and Wei Wang. 2017. Modeling trajectories with recurrent neural networks. IJCAI.

[51] Can Yang and Gyozo Gidofalvi. 2018. Fast map matching, an algorithm integrating hidden Markov model with precomputation. *Int. J. Geogr. Inf. Syst.* 32, 3 (2018), 547–570.

[52] Sean Bin Yang, Chenjuan Guo, and Bin Yang. 2020. Context-Aware Path Ranking in Road Networks. *TKDE* (2020).

[53] Xiaochun Yang, Bin Wang, Kai Yang, Chengfei Liu, and Baihua Zheng. 2017. A novel representation and compression for queries on trajectories in road networks. *TKDE* 30, 4 (2017), 613–629.

[54] Huaxiu Yao, Yiding Liu, Ying Wei, Xianfeng Tang, and Zhenhui Li. 2019. Learning from multiple cities: A meta-learning approach for spatial-temporal prediction. In *WWW*. 2181–2191.

[55] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875* (2017).

[56] Haitao Yuan and Guoliang Li. 2021. A Survey of Traffic Prediction: from Spatio-Temporal Data to Intelligent Transportation. *Data Sci. Eng.* 6, 1 (2021), 63–85.

[57] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2020. Effective Travel Time Estimation: When Historical Trajectories over Road Networks Matter. In *SIGMOD*.

[58] Dongxiang Zhang, Mengting Ding, Dingyu Yang, Yi Liu, Ju Fan, and Heng Tao Shen. 2018. Trajectory simplification: an experimental study and quality analysis. *PVLDB* 11, 9 (2018), 934–946.

[59] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1234–1241.

[60] Jiangchuan Zheng and Lionel M Ni. 2013. Time-dependent trajectory regression on road networks via multi-task learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 27. 1048–1055.