

# Origin-Destination Travel Time Oracle for Map-based Services

Yan Lin<sup>1,3</sup>, Huaiyu Wan<sup>1,3</sup>, Jilin Hu<sup>2</sup>, Shengnan Guo<sup>1,3</sup>, Bin Yang<sup>2</sup>, Youfang Lin<sup>1,3</sup>, Christian S. Jensen<sup>2</sup>

<sup>1</sup>School of Computer and Information Technology, Beijing Jiaotong University, China

<sup>2</sup>Department of Computer Science, Aalborg University, Denmark

<sup>3</sup>Beijing Key Laboratory of Traffic Data Analysis and Mining, Beijing, China

{ylincs,hywan,guoshn}@bjtu.edu.cn,{hujilin,byang,cnj}@cs.aau.dk

## ABSTRACT

Given an origin (O), a destination (D), and a departure time (T), an Origin-Destination (OD) travel time oracle (ODT-Oracle) returns an estimate of the time it takes to travel from O to D when departing at T. ODT-Oracles serve important purposes in map-based services. To enable the construction of such oracles, we provide a travel-time estimation (TTE) solution that leverages historical trajectories to estimate time-varying travel times for OD pairs.

The problem is complicated by the fact that multiple historical trajectories with different travel times may connect an OD pair, while trajectories may vary from one another. To solve the problem, it is crucial to remove outlier trajectories when doing travel time estimation for future queries.

We propose a novel, two-stage framework called Diffusion-based Origin-destination Travel Time Estimation (DOT), that solves the problem. First, DOT employs a conditioned Pixelated Trajectories (PiT) denoiser that enables building a diffusion-based PiT inference process by learning correlations between OD pairs and historical trajectories. Specifically, given an OD pair and a departure time, we aim to infer a PiT. Next, DOT encompasses a Masked Vision Transformer (MViT) that effectively and efficiently estimates a travel time based on the inferred PiT. We report on extensive experiments on two real-world datasets that offer evidence that DOT is capable of outperforming baseline methods in terms of accuracy, scalability, and explainability.

### ACM Reference Format:

Yan Lin<sup>1,3</sup>, Huaiyu Wan<sup>1,3</sup>, Jilin Hu<sup>2</sup>, Shengnan Guo<sup>1,3</sup>, Bin Yang<sup>2</sup>, Youfang Lin<sup>1,3</sup>, Christian S. Jensen<sup>2</sup>. 2023. Origin-Destination Travel Time Oracle for Map-based Services. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 INTRODUCTION

The diffusion of smartphones and the ongoing digitalization of societal processes combine to enable a wide range of map-based services [9, 12, 30]. Many such services rely on the availability of origin-destination (OD) oracles that provide estimates of the travel times, distances, and paths between origin (O) and destination (D) locations, e.g., distance oracles [39] and path oracles [40]. Examples

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'17, July 2017, Washington, DC, USA

© 2023 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

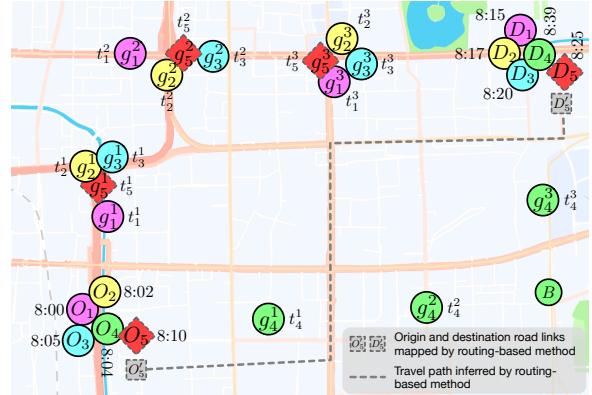


Figure 1: Motivation Example for ODT-Oracle.

include pricing in outsourced transportation services [2, 49], estimation of overall travel costs [31], transportation scheduling [7, 28], delivery services [38, 52], and traffic flow prediction [59]. For example, in flex-transport, taxi companies are paid by a public entity for making trips. The payments are based on pricing models that involve estimating the travel times of trips, but the driver is free to choose any travel path.

We study the problem of constructing an OD travel time oracle (ODT-Oracle) that takes an OD pair and a departure time  $T$  as input, and returns a travel time  $\Delta t$  needed to travel from O to D when departing at time  $T$ . ODT-Oracle can provide accurate estimation of travel times that is valuable for helping public entities, taxi companies, or other transportation service providers to plan their operations more effectively, while minimizing the need for detailed travel path information.

**EXAMPLE 1.** Figure 1 shows four trajectories for similar OD pair:  $\mathcal{T}_1 = \langle (O_1, 8:00), (g_1^1, t_1^1), (g_1^2, t_1^2), (g_1^3, t_1^3), (D_1, 8:15) \rangle$ ,  $\mathcal{T}_2 = \langle (O_2, 8:02), (g_2^1, t_2^1), (g_2^2, t_2^2), (g_2^3, t_2^3), (D_2, 8:17) \rangle$ ,  $\mathcal{T}_3 = \langle (O_3, 8:05), (g_3^1, t_3^1), (g_3^2, t_3^2), (g_3^3, t_3^3), (D_3, 8:20) \rangle$ , and  $\mathcal{T}_4 = \langle (O_4, 8:04), (g_4^1, t_4^1), (g_4^2, t_4^2), (g_4^3, t_4^3), (D_4, 8:39) \rangle$ , where each element  $(g, t)$  denotes a GPS point  $g$  and timestamp  $t$ . Thus, we have four data instances for ODT-Oracle:  $[O_1, D_1, 8:00] \rightarrow 15$ ,  $[O_2, D_2, 8:02] \rightarrow 15$ ,  $[O_3, D_3, 8:05] \rightarrow 15$ , and  $[O_4, D_4, 8:04] \rightarrow 35$ , where the travel time is the arrival time minus the departure time, e.g.,  $\mathcal{T}_1$  has travel time  $8:15 - 8:00 = 15\text{ min}$ .

We observe that  $\mathcal{T}_4$  is very different from the other three trajectories since it goes via place B, which makes it an outlier. When given a query  $Q = [O_5, D_5, 8:10]$ , the ODT-Oracle is to estimate the travel time from  $O_5$  to  $D_5$  at 8:10. Suppose we have an unseen trajectory  $\mathcal{T}_5 = \langle (O_5, 8:10), (g_5^1, t_5^1), (g_5^2, t_5^2), (g_5^3, t_5^3), (D_5, 8:25) \rangle$ . Then, we can use the travel time of  $\mathcal{T}_5$ , 15min, as the ground truth result for the query  $Q = [O_5, D_5, 8:10]$ . The closer the result is to 15min, the more accurate the ODT-Oracle is.

Studies exist that have attempted to solve this problem can be classified into two main categories [58]: **Non-machine learning-based method** [48] and **machine learning-based method** [34].

We can further classify **non-machine learning-based methods** into *historical trajectory-based methods* and *path-based methods*. TEMP [48] is a representative *historical trajectory-based method* that estimates the travel time of a given ODT-Input by averaging the travel time of historical trajectories that have a similar origin, destination and departure time. It suffers from poor accuracy when very different trajectories exist for the same O and D. For example, Figure 1 has four trajectories with similar departure times, where three similar trajectories ( $\mathcal{T}_1$ ,  $\mathcal{T}_2$ , and  $\mathcal{T}_3$ ) have travel time 15 minutes, while the latter ( $\mathcal{T}_4$ ) goes via point B and takes 35 minutes. In this case, the *historical trajectory-based method* returns  $(15 \times 3 + 35)/4 = 20$  minutes, which is inaccurate due to outlier  $\mathcal{T}_4$ .

The *path-based methods* effectively solve a shortest path problem. They first map the GPS coordinates of the origin and destination onto a road network using map-matching [20], i.e.,  $O \rightarrow O'$  and  $D \rightarrow D'$ . Then, they calculate the shortest path from  $O'$  to  $D'$  and return the travel time of this path as the estimated OD travel time. However, this type of method may also have poor accuracy due to two reasons. First, the map-matching results may be inaccurate. Second, the weights in the road network are not accurate. For example, in Figure 1, the query origin  $O_5$  and destination  $D_5$ , are mapped to  $O'_5$  and  $D'_5$ , respectively. Then, a *path-based method* finds the shortest path from  $O'_5$  to  $D'_5$ , which is the dashed line in Figure 1. However, the underlying trajectories relevant to the query  $[O_5, D_5, 8:10]$  do not use the shortest path, which makes the travel time estimate of *path-based methods* inaccurate.

When considering the **machine learning-based methods**, most existing studies [22, 24, 29, 34] model an **ODT-Oracle** as a regression problem without considering historical trajectories. Still, figure 1 has four historical trajectories with similar OD pairs, which are four independent training data instances for regression models. When regression models are trained with this data by using the least squares method or mean squared error (MSE) through backpropagation, they are likely to output 20 minutes when fed the same OD pair. Therefore, the regression-based methods also experience poor accuracy.

The recent DeepOD [58] attempts to alleviate this problem by introducing an auxiliary loss. Given a historical trajectory, it first learns an OD representation with the given OD pair, departure time, and external features. It also tries to learn a representation of the given trajectory. The next step is to match the two representations, i.e., the OD and affiliated trajectory representations, which is the auxiliary loss. Then the OD representation is used to estimate the travel time that is compared with the ground truth, which is the main loss. Finally, DeepOD is trained with the combination of main loss and the auxiliary loss, which enables performance improvements. However, in the example in Figure 1, outlier  $\mathcal{T}_4$  is still used for training in DeepOD, which is the key reason for the reduced accuracy of existing solutions for **ODT-Oracles**.

In this paper, we propose a new trajectory format, namely that of a Pixelated Trajectory (PiT). We represent a PiT using an image format, which is denoted as  $\mathbb{R}^{N \times M \times C}$ , where  $N$  and  $M$  are the height and width of the PiT, respectively, and  $C$  denotes the

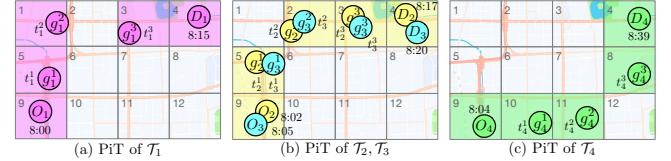


Figure 2: Examples of Pixelated Trajectories (PiTs).

number of features in the PiT, which consists of a Mask, a Time of the day (ToD), and a Time offset. Figure 2 shows examples of PiTs for  $\mathcal{T}_1$ ,  $\mathcal{T}_2$ ,  $\mathcal{T}_3$ , and  $\mathcal{T}_4$ . In the example, we use  $N = 3$ ,  $M = 4$ , and  $C = 1$ , so the PiT has the format  $X \in \mathbb{R}^{3 \times 4 \times 1}$ , where  $X[\cdot, \cdot, 1]$  is the mask that captures whether the cells are traversed by the trajectory or not. Although  $\mathcal{T}_1$ ,  $\mathcal{T}_2$ , and  $\mathcal{T}_3$  are different, their corresponding PiTs are very similar. This similarity helps the model learn common patterns and characteristics from them. On the other hand, PiT $_4$  is quite different from the other PiTs, which allows for easier identification and removal of this PiT as an outlier. This demonstrates the effectiveness of the proposed PiT representation in handling different trajectories and identifying outliers.

To remove the impact of outliers, we propose a novel, two-stage ODT-Oracle framework, called *Diffusion-based Origin-destination Travel Time Estimation* (DOT).

In the first stage, we propose a *PiT inference model* to infer the PiT for a query  $(O, D, T)$ . Given the example in Figure 1 and the query  $Q = [O_5, D_5, 8:10]$ , we aim to generate a PiT that is similar to  $PiT_1$ ,  $PiT_2$ , and  $PiT_3$ , but is different from  $PiT_4$ . To do this, we first propose a conditional diffusion model to generate a PiT based on the origin, destination, and departure time. Then, we leverage historical trajectory data to train this model such that it can generate a PiT when  $(O, D, T)$  is given.

In the second stage, called *PiT travel time estimation*, we aim to estimate travel times based on the generated PiTs. For example, given the PiTs in Figure 2, we generate a PiT that is similar to  $PiT_1$ ,  $PiT_2$ , and  $PiT_3$  but is different from  $PiT_4$ . So, given the query  $Q = [O_5, D_5, 8:10]$ , the outlier  $\mathcal{T}_4$  is disregarded during estimation.

Since the inferred PiT is formed by cells with spatial-temporal features, we attempt to model the global correlations in PiTs based on self-attention and the vision Transformer (ViT) [8] to improve the estimation accuracy. Considering that a PiT only occupies very few cells in the whole image, we further propose a Masked Vision Transformer (MViT) equipped with an efficient self-attention masking scheme to speed up the estimation process. The efficiency of training and estimation are improved substantially compared to the original ViT.

In summary, we make the following contributions:

- (1) We propose a novel two-stage framework for enabling accurate ODT-Oracles.
- (2) We introduce a PiT inference model that identifies a PiT conditioned on a query  $(O, D, T)$ , making it possible to reduce the impact of outlier trajectories.
- (3) We propose a Masked Vision Transformer to model global spatial-temporal correlation in the inferred PiT, enabling more effective and efficient travel time estimation.
- (4) We report on extensive experiments on two real-world datasets that offer evidence that the proposed method is capable of outperforming existing methods.

The remainder of the paper is organized as follows. Section 2 covers related work. Section 3 introduces preliminaries and formalizes the problem. Sections 4 and 5 detail PiT inference and travel time estimation, respectively. Section 6 reports on the empirical study, and Section 7 concludes.

## 2 RELATED WORK

### 2.1 Travel Time Estimation

Travel time estimation (TTE) has important applications. In location-based services, TTE can provide users with information on how to plan their trip or on when their packages will arrive. In urban planning, many data analyses, such as flow prediction and congestion prediction, depending on the results of TTE. TTE solutions can be classified into path-based TTE and ODT-Oracles, where the biggest difference is whether a path is given.

*Path-based travel time estimation.* Early studies implement regression techniques [18] or decomposition methods [50] to estimate the travel time of trajectories. The accuracies of these methods are limited due to their limited modeling capacities. More recently, deep learning approaches that are capable of modeling complex spatial-temporal correlations in trajectories have been gaining attention. WDR [51], DeepTTE [47], DeepETA [53], TADNM [56], and CompactETA [10] all utilize recurrent neural networks [5, 16] to model the sequential spatial-temporal information embedded in trajectories. Considering travel time as distributions rather than scalar values, DeepGTT [27] implements a variational encoder to model the travel time distributions of trajectories. To further improve the prediction performance, TAML [55], DRTTE [57], and WDDRA [11] implement a multitask learning framework to make the estimation of timestamps more accurate. DFTTE [43] proposes a fusion network to merge multiple sources of information for prediction. MetaTTE [46] incorporate meta-learning technique to generalize travel time estimation on multi-city scenarios. STDGCN [21] employs the neural architecture search techniques to identify the optimal network structure for estimation.

*Path-routing methods.* The superiority of path-based TTE methods is enabled by their affiliated trajectories. But in the scenario of an ODT-Oracle, paths are unknown to the model, rendering path-based methods inapplicable. One plausible solution to this issue involves leveraging path-routing methods to infer the potential paths between a given origin-destination pair. Classical algorithms such as Dijkstra's algorithm [23] and various alternative routing methods [32] primarily focus on determining the paths associated with minimal travel costs, yet their calculated routes may deviate from the actual paths drivers would choose. DeepST [26] makes use of historical travel behavior derived from trajectory data, thereby enhancing the accuracy of generated paths. It's noteworthy that route recovery methods [6, 54] and map-matching algorithms [3], though bearing structural similarity to path-routing methods, often demand specific details such as arrival time or comprehensive GPS sequences, which are impractical in the context of the ODT-Oracle.

*ODT-Oracles.* The motivation for building an ODT-Oracle comes from problems related to path-based travel time estimation. During estimation, only origin and destination locations and departure

times are given, and it is challenging to build an accurate ODT-Oracle since the underlying trajectories are unknown. TEMP [48] averages the travel times of historical travels and does not contain any learnable parameter. ST-NN [22] proposes to learn a non-linear mapping of origin-destination location pairs to the corresponding travel times using neural networks. MURAT [29] extends the input features with embeddings from road segments, spatial cells, and temporal slots. Nevertheless, the underlining trajectory of a trip is highly related to the travel time. All the above methods ignore the correlations between origin-destination location pairs and travel trajectories in historical data, so their prediction accuracies are limited. DeepOD [58] addresses this problem by incorporating historical trajectories during training and making the embeddings of origin-destination pairs and travel trajectories close in the latent space. However, it is sensitive to outliers in historical trajectories and is unable to provide explainable travel times.

### 2.2 Diffusion Models

The diffusion model is a generative model recently popularized in computer vision [15, 36, 42]. Coming from nonequilibrium thermodynamics [19, 41], diffusion models have a strong theoretical background and have been proven to perform excellently in image generation. There are two Markov processes in diffusion models: the forward diffusion process and the reverse denoising diffusion process. The forward diffusion process adds noise to a clean image through a pre-defined noise schedule until it turns into a Gaussian noise. The reverse denoising diffusion process removes noise from the noisy image step-by-step until the clean image is recovered. After training, one can utilize the reverse process to randomly generate images that follow the distribution of the training dataset.

Due to the effectiveness and flexibility of diffusion models, many ongoing efforts aim to migrate them into other domains and tasks, such as time series imputation [44], audio synthesis [25], shape generation [61] and language modeling [1]. In this study, we exploit diffusion models as a powerful means of modeling correlations from historical trajectories. Specifically, we propose a conditioned diffusion model for accurate and explainable ODT-Oracle.

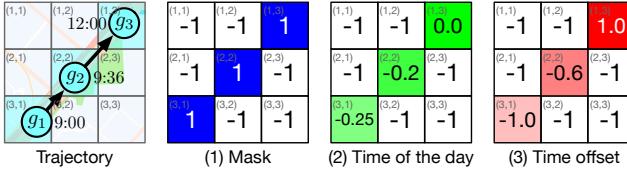
## 3 PRELIMINARIES

### 3.1 Definitions

**DEFINITION 1 (TRAJECTORY).** A trajectory  $\mathcal{T}$  is a sequence of timestamped GPS points:  $\mathcal{T} = \langle (g_1, t_1), (g_2, t_2), \dots, (g_{|\mathcal{T}|}, t_{|\mathcal{T}|}) \rangle$ , where  $g_i = (\text{lng}_i, \text{lat}_i)$ ,  $i = 1, \dots, |\mathcal{T}|$  denotes  $i$ -th GPS point, and  $|\mathcal{T}|$  denotes the total number of GPS points in the trajectory.

**DEFINITION 2 (PIXELATED TRAJECTORY).** Given an area of interest on the map, usually, the area covering all historical trajectories, we split the longitude and latitude equally with a total number of  $L_G$  segments, resulting in a total of  $L_G^2$  spatial cells. A Pixelated Trajectory (PiT),  $X \in \mathbb{R}^{L_G \times L_G \times C}$ , is represented as a tensor, where  $C$  is the number of channels. Each trajectory has a corresponding PiT.

In one PiT, the feature  $X[x, y, k]$  records the value of  $k$ -th feature in cell  $(x, y)$ . We utilize three feature channels, i.e., Mask, Time of the day (ToD), and Time offset. If the GPS points in a trajectory  $\mathcal{T}$  never fall into the cell  $(x, y)$ , then all three channels of the corresponding cell are set to -1. If a GPS point  $(g_i, t_i)$  in  $\mathcal{T}$  is the earliest

**Figure 3: Constructing the channels of PiT from GPS points.**

one that falls into the cell  $(x, y)$ , we calculate the values of three channels as follows.

- (1) *Mask*. Indicates whether the trajectory contains GPS points located in this cell or not.  $X[x, y, 1] = 1$  means there are one or more GPS points in the trajectory that are located in cell  $(x, y)$ .
- (2) *ToD*. A normalized value with range  $[-1, 1]$  that denotes when the cell is visited. It can be calculated as  $X[x, y, 2] = 2 \times (t_i \% 86400) / 86400 - 1$ , where  $t_i$  denotes the Unix timestamp of  $i$ -th GPS point in the trajectory  $\mathcal{T}$  and 86400 is the total number of seconds in 24 hours.
- (3) *Time offset*. Also, a normalized value with range  $[-1, 1]$  indicates the visiting order of this cell in the trajectory. It can be calculated as  $X[x, y, 3] = 2 \times (t_i - t_1) / (t_{|\mathcal{T}|} - t_1) - 1$ , where  $t_i$  denotes the Unix timestamp of  $i$ -th GPS point in the trajectory  $\mathcal{T}$ .

**EXAMPLE 2.** Figure 3 demonstrates an example of PiT construction. Suppose we have a trajectory  $\mathcal{T} = \langle (g_1, 9:00), (g_2, 9:36), (g_3, 12:00) \rangle$ , and we split the area of interest into  $3 \times 3$  cells. GPS points  $g_1, g_2, g_3$  locate in cell  $(3, 1), (2, 2), (1, 3)$ , respectively. Thus, we set the mask channels of the three cells as 1. The ToD channels of the cells are calculated as  $X[3, 1, 2] = 2 \times 9 \times 24 \times 60 / 86400 - 1 = -0.25$ ,  $X[2, 2, 2] = 2 \times (9 \times 24 + 36) \times 60 / 86400 - 1 = -0.2$ ,  $X[1, 3, 2] = 2 \times (12 \times 24 \times 60) / 86400 - 1 = 0.0$ , respectively. The time offset channel of the cell  $(2, 2)$  is calculated as  $X[2, 2, 3] = 2 \times (9 \times 24 + 36) / (12 \times 24 - 9 \times 24) = -0.6$ , while  $X[3, 1, 3] = -1.0$ ,  $X[1, 3, 3] = 1.0$ . For other cells where no GPS point is located in, all their channels are set to  $-1$ . In this way, we obtain a PiT,  $X \in \mathbb{R}^{3 \times 3 \times 3}$ , corresponding to the trajectory  $\mathcal{T}$ .

By representing historical trajectories using PiT rather than using raw sequential features, our method can focus on differences in trajectories that will actually affect the travel time, rather than minor diversities that might cause disturbances. On the other hand, transforming sequential trajectories into equally-sized images is better suited for the two-stage framework we introduce later.

**DEFINITION 3 (ODT-INPUT).** The ODT-Input for an ODT-Oracle,  $odt$ , is a tuple that consists of three elements:  $odt = (g_o, g_d, t_o)$ , where  $g_o$  and  $g_d$  are the GPS coordinates of the origin and destination, respectively.  $t_o$  denotes the departure time.

### 3.2 Problem Formulation

**OD Travel Time Oracle.** Given the set of historical trajectories  $\mathbb{T}$ , we aim to learn ODT-Oracle  $f_\theta^{\mathbb{T}}$  to estimate the travel time  $\Delta t$  and the PiT  $X$  for any future ODT-Input,  $odt$ . To be noted,  $odt$  in the query does not require to appear in  $\mathbb{T}$ . Formally, we have:

$$odt \xrightarrow{f_\theta^{\mathbb{T}}(\cdot)} \Delta t, X \quad (1)$$

### 3.3 Method Overview

In this work, we propose a *Diffusion-based Origin-destination Travel Time Estimation* (DOT) method to build an accurate and explainable ODT-Oracle, which follows a two-stage framework. The first stage is the PiT inference stage, shown in Figure 4(a); the second stage is the PiT travel time estimation stage, shown in Figure 4(b).

In the PiT inference stage, we try to infer the PiT corresponds to a given ODT-Input. The ODT-Input is considered the conditional information incorporated into a conditioned PiT denoiser. The denoiser samples from standard Gaussian noise at the beginning. Then, it produces the inferred PiT conditioned on the ODT-Input through a multi-step conditioned denoising diffusion process.

In the PiT travel time estimation stage, we estimate the travel time based on the inferred PiT. The PiT is first flattened and mapped into a feature sequence to capture the global spatial-temporal correlation better. To improve the model's efficiency, we propose a Masked Vision Transformer (MViT) to estimate the travel time.

We explain the proposed method in detail in the following sections. The PiT inference stage is introduced in Section 4, and the PiT travel time estimation stage is introduced in Section 5.

## 4 MODELING HISTORICAL TRAJECTORIES THROUGH PiT INFERENCE

### 4.1 Diffusion-based PiT Inference

Given the origin-destination location pair of a future trip, the travel time is highly related to the route that a driver takes. However, the actual route is unavailable at the time of estimating. To achieve accurate ODT-Oracle, we aim to comprehensively learn the correlation between ODT-Inputs and travel trajectories from historical trips. Then, we can utilize the learned correlation to infer travel time based on possible trajectories given the future ODT-Inputs.

Suppose we have the historical trajectory, denoted as  $\mathcal{T}$ , and the corresponding ODT-Input,  $odt = (g_o, g_d, t_o)$  of the trip. We aim to learn a posterior probability  $p(\mathcal{T}|odt)$ , which represents the relationships between  $odt$  and  $\mathcal{T}$  from the set of historical trajectories  $\mathbb{T}$ . Since we represent a trajectory  $\mathcal{T}$  using the PiT, denoted as  $X$ , which is stated in Section 3.1, the probability can be written as  $p(X|odt)$ . However, this probability is unknown in reality, so we aim to learn it through a set of learnable parameters  $\theta$ . The estimated probability is then denoted as  $p_\theta(X|odt)$ .

There is no doubt that the learning model  $p_\theta$  is crucial to achieving accurate ODT-Oracle. If the inferred travel trajectory from  $p_\theta$  is very different from the expected trajectory given a future ODT-Input, then the estimated travel time cannot be good. In this paper, we build our learning model based on Denoising Diffusion Probabilistic Models (DDPM) [15]. Since DDPM models the unconditioned data distribution  $p(X)$ , the generated data is not restricted, which can be any signal from the underlying data distribution. For example, it can generate various PiTs, whose origins and destinations differ. However, in our case, we aim to model the data distribution conditioned on ODT-Input  $odt$ . Therefore, we propose a diffusion-based conditioned PiT inference framework, which consists of a diffusion process and a conditioned denoising diffusion process. We detail these two processes in the following Sections.

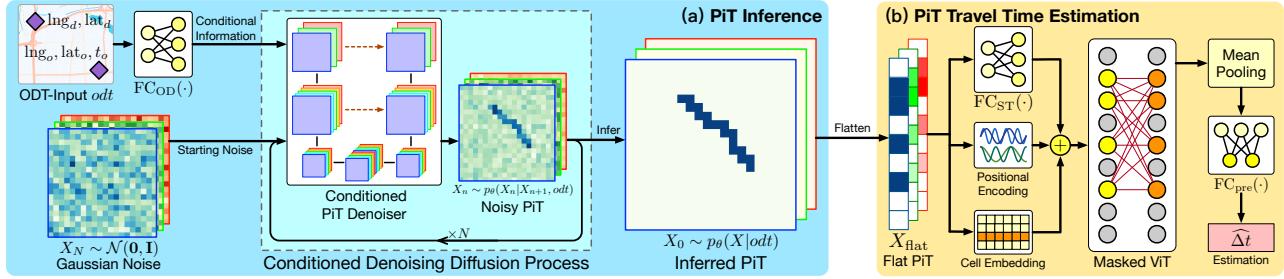


Figure 4: The two-stage framework of DOT.

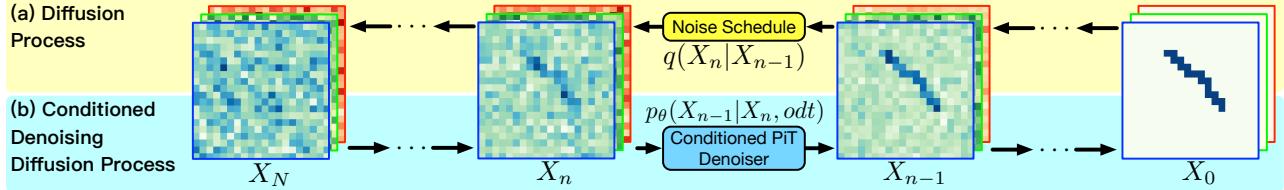


Figure 5: Two Markov processes in the diffusion-based conditioned PiT inference framework.

**4.1.1 The diffusion process for adding noise to PiTs.** Intuitively, the diffusion process is to add noise to a signal step by step until we reach a simple prior distribution, e.g., Gaussian distribution. Figure 5(a) gives a clear illustration of how the diffusion process works in our paper, i.e., we try to obtain a noisy PiT,  $X_N$ , by giving a clear PiT,  $X_0$ , in a total of  $N$  diffusion steps. A single step of this diffusion process can be formulated as follows.

$$q(X_n|X_{n-1}) = \mathcal{N}(X_n; \sqrt{1 - \beta_n}X_{n-1}, \beta_n\mathbf{I}), \quad (2)$$

where  $\mathcal{N}(\mu; \Sigma)$  is the Gaussian distribution with mean value  $\mu$  and covariance matrix  $\Sigma$ ,  $\beta_n$  is the coefficient used for controlling the noise level in the  $n$ -th step. In practice,  $\beta_n$  is often fixed for every step and follows a monotonic schedule so that the added noise level increases with  $n$ . We follow the linear schedule used in DDPM [15], where  $\beta_n$  scales linearly from 0.0001 to 0.02 with  $n$ .

Starting from the clear PiT  $X_0$ , we can get the noisy PiT in the  $n$ -th step by:

$$q(X_n|X_0) = \prod_{m=1}^n q(X_m|X_{m-1}) \quad (3)$$

Since the noise level in each step is fixed and the added noises follow Gaussian distribution, we can simplify Equation 3 into the following form by utilizing the property of Gaussian distribution.

$$q(X_n|X_0) = \mathcal{N}(X_n; \sqrt{\alpha_n}X_0, (1 - \bar{\alpha}_n)\mathbb{I}), \quad (4)$$

where  $\alpha_n = 1 - \beta_n$  and  $\bar{\alpha}_n = \prod_{m=1}^n \alpha_m$ . Therefore, we can sample the noisy PiT at any step in the diffusion process by adding a specific noise, i.e.,  $q(X_n|X_0)$ , to the original  $X_0$ , without adding noise step-by-step.

Finally, the noisy PiT in the last step of the diffusion process follows a standard Gaussian noise, which is represented as follows.

$$X_N \sim q(X_N) = \mathcal{N}(X_N; \mathbf{0}, \mathbf{I}) \quad (5)$$

Here,  $X_N$  is a good start for the following PiT inference process since it contains no prior information. In other words, we can randomly sample noise from the standard Gaussian distribution as the starting point for the inference process.

**4.1.2 The conditioned denoising diffusion process for PiT inference.** To infer a PiT given a future ODT-Input, we implement a reverse diffusion process under the prior information of ODT-Input. More specifically, we gradually remove noise from the noisy PiT, until we retrieve a noise-free PiT of the travel trajectory corresponding to the ODT-Input. The process is formally called the conditioned denoising diffusion process, which is illustrated in Figure 5(b). A single step of the conditioned denoising diffusion process can be formulated as follows.

$$p_\theta(X_{n-1}|X_n, odt) = \mathcal{N}(X_{n-1}; \mu_{n-1}, \Sigma_{n-1}), \quad (6)$$

where  $\mu_{n-1}$  and  $\Sigma_{n-1}$  are the mean and variance at the step of  $n-1$ , and  $X_{n-1}$  follows a conditional probability relies on the noisy PiT from the previous step  $X_n$  and the ODT-Input  $odt$ . However, we are unaware of the actual form of  $p$  in practice, so we implement two neural network modules to learn the mean and variance. Thus, Equation 6 can be reformulated as follows.

$$p_\theta(X_{n-1}|X_n, odt) = \mathcal{N}(X_{n-1}; \mu_\theta(X_n, n, odt), \Sigma_\theta(X_n, n, odt)), \quad (7)$$

where  $\mu_\theta(\cdot)$  and  $\Sigma_\theta(\cdot)$  denotes the mean and variance estimated by the neural network parameterized by  $\theta$ . The complete PiT inferring process can be formulated as follows.

$$p_\theta(X_0|X_N, odt) = \prod_{n=1}^N p_\theta(X_{n-1}|X_n, odt) \quad (8)$$

As is denoted in Equation 5,  $X_N$  can be sampled from a standard Gaussian distribution, and  $odt$  is the input from the user. Therefore, we can infer a PiT based on these two inputs. We detail the process of PiT inference in Algorithm 1.

Since we utilize a probabilistic model to infer the most plausible PiT given an ODT-Input, we can remove the impact of outliers. The remaining question is how to train the set of parameters  $\theta$  with the historical trajectories  $\mathbb{T}$ , which we discuss in the following section.

**4.1.3 Re-parameterization and training of the PiT inference model.** To simplify the training of  $\theta$ , we follow DDPM [15] to build the conditioned denoising diffusion process by keeping the variance

**Algorithm 1** Inferring PiT given ODT-Input

**Require:** A simulated probability  $p_\theta(X_{n-1}|X_n, odt)$ , a future ODT-Input  $odt$

- 1: Initialize  $n \leftarrow N, X_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 2: **while**  $n > 0$  **do**
- 3:     Sample  $X_{n-1} \sim p_\theta(X_{n-1}|X_n, odt)$
- 4:     Set  $X_n \leftarrow X_{n-1}, n \leftarrow n - 1$
- 5: **end while**
- 6: Obtain generation result  $X_0$  as the inferred PiT  $X$  representative of the travel trajectory of  $odt$ .

**Algorithm 2** Training PiT inference model

**Require:** A noise predictor  $\epsilon_\theta$  with a set of learnable parameters  $\theta$

- 1: **while** model not converged **do**
- 2:     Sample  $\mathcal{T} \in \mathbb{T}$ , calculate the ODT-Input  $odt$ , and the corresponding PiT  $X$  as the original image  $X_0$
- 3:     Sample  $n \sim \text{Uniform}(1, 2, \dots, N)$
- 4:     Sample  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 5:     Calculate  $X_n = \sqrt{\alpha_n}X_0 + \sqrt{1 - \bar{\alpha}_n}\epsilon$
- 6:     Take a gradient descent step on  $\nabla_\theta \|\epsilon - \epsilon_\theta(X_n, n, odt)\|^2$
- 7: **end while**

in Equation 7 fixed at any given step, i.e.,  $\Sigma_\theta(X_n, n, odt) = \sqrt{\beta_n}\mathbf{I}$ . Therefore, only the mean needs to be estimated.

Then, we re-parameterize the mean  $\mu_\theta(\cdot)$ . Instead of directly predicting  $\mu_\theta(\cdot)$ , we aim to predict the added noise at the  $n$ -th step in the diffusion process. We denote the predicted noise as  $\epsilon_\theta(X_n, n, odt)$ , such that  $\mu_\theta(\cdot)$  can be re-parameterized as follows.

$$\mu_\theta(X_n, n, odt) = \frac{1}{\sqrt{\alpha_n}}(X_n - \frac{\beta_n}{\sqrt{1 - \bar{\alpha}_n}}\epsilon_\theta(X_n, n, odt)) \quad (9)$$

Next, we re-parameterize Equation 7, and substitute  $\mu_\theta(\cdot)$  with Equation 9, which can be formulated as follows.

$$\begin{aligned} X_{n-1} &= \mu_\theta(X_n, n, odt) + \Sigma_\theta(X_n, n, odt)\epsilon \\ &= \frac{1}{\sqrt{\alpha_n}}(X_n - \frac{\beta_n}{\sqrt{1 - \bar{\alpha}_n}}\epsilon_\theta(X_n, n, odt)) + \sqrt{\beta_n}\epsilon, \end{aligned} \quad (10)$$

where  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ .

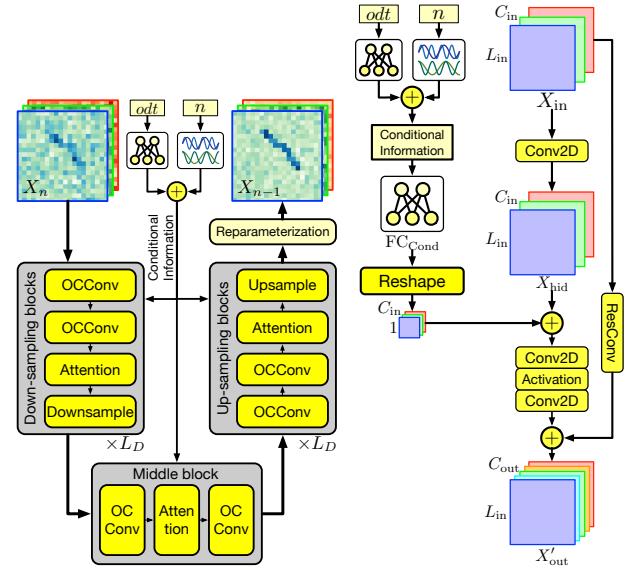
To train the set of parameters  $\theta$ , we must supervise all steps of the conditioned denoising diffusion process. In practice, we can sample  $n$  from a uniform distribution,  $U(1, N)$ , so that all steps can eventually be trained. At the  $n$ -th step, we use the mean squared error to minimize the difference between the ground truth and the predicted noise, and the loss function can be formulated as follows.

$$\begin{aligned} L_n &= \|\epsilon - \epsilon_\theta(X_n, n, odt)\|^2 \\ &= \|\epsilon - \epsilon_\theta(\sqrt{\alpha_n}X_0 + \sqrt{1 - \bar{\alpha}_n}\epsilon, n, odt)\|^2, \end{aligned} \quad (11)$$

where the noisy PiT,  $X_n$ , comes from the diffusion process and is calculated using the simplified form presented in Equation 4. The detailed training algorithm is presented in Algorithm 2.

## 4.2 Conditioned PiT Denoiser

In this section, in detail, we introduce the implementation of denoiser  $\epsilon_\theta(X_n, n, odt)$ . Two basic requirements need to be met by the denoiser. The first one is that the predicted noise, i.e., this denoiser's



(a) The overall architecture of the conditioned PiT denoiser.  
(b) The fuse of conditional information in the OCCConv module.

Figure 6: Conditioned PiT denoiser.

output, should be the same size as the input noisy PiT,  $X_n$ . The other one is that the denoiser takes input as the noisy PiT  $X_n$ , the step indicator  $n$ , and the ODT-Input  $odt$ . Bearing these requirements, we implement our denoiser based on Unet [37], a neural network widely used in computer vision. Unet is efficient yet powerful due to its bottleneck architecture and residual connection design. Yet, the naive Unet cannot take input as  $n$  and  $odt$ . Thus, we aim to cast these features into latent spaces and fuse them into our Unet-based conditioned PiT denoiser. The overall architecture of our denoiser is shown in Figure 6(a).

We first implement the positional encoding to encode the step indicator  $n$  into an embedding vector,  $\text{PE}(n) \in \mathbb{R}^d$ , which is commonly used in Transformer [45]. The detailed encoding operation is formulated as follows:

$$\begin{aligned} \text{PE}(n)[2i] &= \sin(n/10000^{2i/d}) \\ \text{PE}(n)[2i-1] &= \cos(n/10000^{2i/d}), \end{aligned} \quad (12)$$

where  $d$  is an even value, and  $i \in \{1, \dots, d/2\}$  is the dimension indicator of the feature vector.

We then implement a fully-connected layer to transform  $odt$  into a  $d$ -dimensional latent space, formulated as follows.

$$\text{FC}_{\text{OD}}(odt) : \mathbb{R}^5 \rightarrow \mathbb{R}^d \quad (13)$$

Next, we introduce how to fuse these latent representations into the proposed PiT denoiser. Following the Unet [37], our denoiser is formed by  $L_D$  layers of down-sampling blocks, one layer of middle block and  $L_D$  layers of up-sampling blocks, with residual connections between the down-sampling blocks and the up-sampling blocks. Each down-sampling and up-sampling block contains a sequential stack of two ODT-Input Conditioned Convolutional (OC-Conv) modules that are built based on ConvNeXt [33], a multi-head dot-product attention module, and a convolutional module for up-sampling or down-sampling. The middle block contains two

OCCConv modules, with an attention module in the middle. The down-sampling blocks down-sample the input spatially but expand it channel-wisely, while the up-sampling blocks do the opposite. The conditional information  $odt$  and  $n$  is incorporated into every OCCConv module in the denoiser, whose data flow in the OCCConv module is illustrated in Figure 6(b).

Specifically, the OCCConv module firstly takes input as an image, denoted as  $X_{\text{in}} \in \mathbb{R}^{L_{\text{in}} \times L_{\text{in}} \times C_{\text{in}}}$ , which goes through a 2D convolutional layer Conv2D with all dimensions unchanged, resulting in the hidden state  $X_{\text{hid}}$ .

$$X_{\text{hid}} = \text{Conv2D}(X_{\text{in}}), X_{\text{hid}} \in \mathbb{R}^{L_{\text{in}} \times L_{\text{in}} \times C_{\text{in}}} \quad (14)$$

Then, we utilize a fully-connected layer to transform the sum of  $\text{PE}(n)$  and  $\text{FC}_{\text{OD}}(odt)$  into a vector with a size of  $C_{\text{in}}$ , which is then added to every pixel in  $X_{\text{hid}}$ , formulated as follows.

$$\begin{aligned} \text{FC}_{\text{Cond}}(\cdot) : \mathbb{R}^d &\rightarrow \mathbb{R}^{C_{\text{in}}} \\ X'_{\text{hid}}[:, :, i] &= X_{\text{hid}}[:, :, i] + \text{FC}_{\text{Cond}}(\text{PE}(n) + \text{FC}_{\text{OD}}(odt))[i], \end{aligned} \quad (15)$$

where  $i = 1, 2, \dots, C_{\text{in}}$ .

Finally,  $X'_{\text{hid}}$  goes through a two-layer 2D convolutional network with activation and residual connection to the output state:

$$\begin{aligned} X_{\text{out}} &= \text{Conv2D}(\sigma(\text{Conv2D}(X'_{\text{hid}}))) \\ X'_{\text{out}} &= X_{\text{out}} + \text{ResConv}(X_{\text{in}}), \end{aligned} \quad (16)$$

where  $\sigma(\cdot)$  is the activation function, and we use the Gaussian Error Linear Units (GELU) [14] in this paper. ResConv is the residual connection implemented using 2D convolution. The output state  $X'_{\text{out}} \in \mathbb{R}^{L_{\text{in}} \times L_{\text{in}} \times C_{\text{out}}}$  is then fed into the following modules in the denoiser. Usually,  $C_{\text{out}} = 2 \cdot C_{\text{in}}$  for the down-sampling blocks,  $C_{\text{out}} = \lfloor C_{\text{in}}/2 \rfloor$  for the up-sampling blocks.

## 5 PIT TRAVEL TIME ESTIMATION

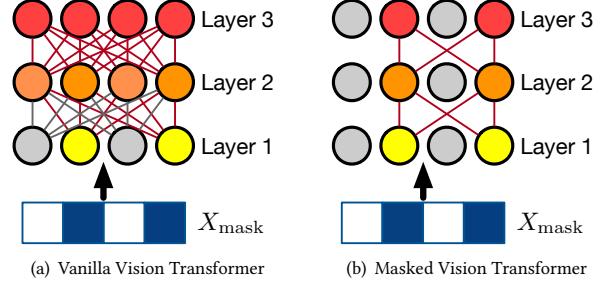
Once we finish the training of the PiT inference stage detailed in Section 4, we aim to propose a PiT travel time estimation module based on the inferred PiT, which is detailed in this Section.

Since the inferred PiT,  $X$ , is in the pixelated format, it is intuitive to come up with an estimator based on convolutional network networks (CNNs). Yet, CNNs focus on modeling local properties, which is not good at travel time estimation, where capturing the global correlation is essential. Vision Transformer (ViT) [8] that utilizes self-attention to consider global correlation among all pixels seems to be a good fit in our case. However, many pixels in PiT don't contain valid information, making the vanilla ViT perform poorly in our scenario. Therefore, we propose the Masked Vision Transformer (MViT) to improve the efficiency in both training and estimating significantly.

### 5.1 PiT Flatten and Feature Extraction

Given the inferred PiT  $X \in \mathbb{R}^{L_G \times L_G \times C}$ , we firstly flatten it to a sequence with a length of  $L_G^2$ :

$$\begin{aligned} X_{\text{flat}} &= \langle X[1, 1, :], X[1, 2, :], \dots, X[1, L_G, :], \\ &\quad X[2, 1, :], X[2, 2, :], \dots, X[2, L_G, :], \\ &\quad \dots \\ &\quad X[L_G, 1, :], X[L_G, 2, :], \dots, X[L_G, L_G, :] \rangle \end{aligned} \quad (17)$$



**Figure 7: Comparison of attention mask scheme between vanilla ViT and MViT.**

After flattening, the cell  $(x, y)$  in PiT becomes the  $(x + (y - 1) * L_G)$ -th item in the sequence.  $X_{\text{flat}}$  follows the arrangement of pixels rather than chronological order like normal temporal sequences.

We then further extract the spatial-temporal features from  $X_{\text{flat}}$  using three embedding modules:

- (1) *Cell embedding module  $E[\cdot]$* . We initialize an embedding matrix  $E \in \mathbb{R}^{L_G^2 \times d_E}$ , where the column  $E[x + (y - 1) * W]$  is the embedding vector of the cell  $(x, y)$ , that is, the embedding vector of item  $X[x, y, :]$  in the flatten sequence.  $d_E$  is the embedding dimension.  $E$  can be viewed as the innate spatial features of cells.
- (2) *Positional encoding module  $\text{PE}(\cdot)$* . Since self-attention is order-independent, we encode the position of items in a flattened sequence using the positional encoding introduced in Equation 12. Here, the encoding vector for item  $X[x, y, :]$  is  $\text{PE}(x + (y - 1) * W) \in \mathbb{R}^{d_E}$ .
- (3) *Latent casting module  $\text{FC}_{\text{ST}}(\cdot)$* . Recall that we design three feature channels for each cell in Section 3.1. We utilize a fully-connected layer to cast them into the latent space:  $\text{FC}_{\text{ST}}(X[x, y, :]) : \mathbb{R}^3 \rightarrow \mathbb{R}^{d_E}$ .

The outputs from three modules are summed up to form the latent input vector for each item in  $X_{\text{flat}}$ :

$$\begin{aligned} X_{\text{latent}}[x, y] &= E[x + (y - 1) * W] + \\ &\quad \text{PE}(x + (y - 1) * W) + \\ &\quad \text{FC}_{\text{ST}}(X[x, y, :]) \end{aligned} \quad (18)$$

Then, the latent sequence,  $X_{\text{latent}} = \langle X_{\text{latent}}[1, 1], X_{\text{latent}}[1, 2], \dots, X_{\text{latent}}[L_G, L_G] \rangle$  is the input to MViT.

### 5.2 Masked Vision Transformer

Many items in  $X_{\text{latent}}$  don't contain any valid information since their corresponding features in PiT are set to  $-1$ . In the vanilla vision Transformer (ViT), we can apply an attention mask so that the attention weights of these items are set to zero. Yet, this mask scheme can't improve computational efficiency since the attention weights are still calculated for all items, shown in Figure 7(a). This is particularly problematic in our scenario, where PiT covers the full spatial space, but most cells are not visited for one trajectory.

To speed up the calculation on flattened PiT sequences, we aim to implement a more efficient mask scheme. We propose the Masked Vision Transformer (MViT). First, we calculate a mask to determine whether one item in  $X_{\text{latent}}$  contains valid information, and we can

obtain such a mask through the first channel of PiT:

$$X_{\text{mask}}[x, y] = \begin{cases} \text{True} & \text{if } X[x, y, 1] \geq 0 \\ \text{False} & \text{if } X[x, y, 1] < 0 \end{cases}, \quad (19)$$

for  $x \in \{1, \dots, L_G\}$ ,  $y \in \{1, \dots, L_G\}$ .  $X_{\text{latent}}[x, y, :]$  contains valid information if  $X_{\text{mask}}[x, y] = \text{True}$ .

Following the Transformer [45] and ViT, our MViT is stacked by multiple layers of MViT layer, and each layer contains two modules, a self-attention and a feed-forward network, both with the residual connection. Unlike ViT, self-attention in MViT is only applied to items with valid information. It is equivalent to only retaining the items with valid information to form a masked sequence and only applying self-attention to the masked sequence, as demonstrated in Figure 7(b). Formally, one MViT layer is calculated as:

$$\begin{aligned} X_{\text{out-seq}} &= \text{FFN}(\text{Att}(\text{Mask}(X_{\text{in-seq}}, X_{\text{mask}}))) \\ \text{Mask}(X_{\text{in-seq}}, X_{\text{mask}}) &= \langle X_{\text{in-seq}}[x, y, :] \mid \\ &x \in \{1, \dots, L_G\}, y \in \{1, \dots, L_G\}, \\ &X_{\text{mask}}[x, y] = \text{True} \rangle, \end{aligned} \quad (20)$$

where  $X_{\text{in-seq}}$ ,  $X_{\text{out-seq}}$  represent the input sequence and output sequence of the MViT layer,  $\text{Mask}(X_{\text{in-seq}}, X_{\text{mask}})$  denotes the masked input sequence,  $\text{Att}(\cdot)$ ,  $\text{FFN}(\cdot)$  denote the multi-head attention and the feed-forward network, respectively. Since we only apply self-attention on items with valid information, the calculation cost depends on the total number of valid items, not the length of the full sequence. Therefore, we can improve the efficiency of flattening PiT sequences significantly.

Then, we stack a total of  $L_E$  MViT layers to form the MViT, and calculate the memory sequence using MViT.

$$X'_{\text{latent}} = \text{MViT}(X_{\text{latent}}, X_{\text{mask}}), \quad (21)$$

where the memory sequence  $X'_{\text{latent}}$  has the same dimension as  $X_{\text{latent}}$ , with length equal to the number of valid items in  $X_{\text{latent}}$ .

Next, we apply a mean pooling layer and a fully-connected layer to calculate the result of travel time estimation as follows.

$$\widehat{\Delta t} = \text{FC}_{\text{pre}}(\text{mean}(X'_{\text{latent}})), \quad (22)$$

where the pooling applies to the dimension of sequence length.

Finally, to train the PiT travel time estimation model, we use mean squared error as the loss function to make the prediction result as close to the ground truth as possible, formulated as follows.

$$L_{\text{pre}} = \|\Delta t - \widehat{\Delta t}\|^2 \quad (23)$$

The two stages, PiT inference and PiT travel time estimation, are trained separately. More specifically, after training of the PiT inference model detailed in Algorithm 2, the learnable parameters  $\theta$  are fixed. During training of the PiT travel time estimation model,  $p_\theta(\cdot)$  is only used for inferring, without further parameter update.

By properly utilizing the inferred PiT for travel time estimation, we can achieve high estimation accuracy without the real trajectories that are also not available.

## 6 EXPERIMENTS

To evaluate the effectiveness of the proposed DOT framework, we conduct extensive experiments on two real-world trajectory datasets and compare DOT with existing ODT-Oracle methods.

**Table 1: Dataset statistics.**

| Dataset  | Chengdu           | Harbin            |
|--|-------------------|-------------------|
| Time span  | 11.01–11.10, 2018 | 01.03–01.07, 2015 |
| Number of trajectories                           | 1,389,138         | 614,830           |
| Mean travel time (minutes)                       | 13.73             | 15.69             |
| Mean travel distance (meters)                    | 3,283             | 3,376             |
| Mean sample interval (seconds)                   | 29.06             | 44.42             |
| Area of interest (width*height km <sup>2</sup> ) | 15.32*15.19       | 18.66*18.24       |

### 6.1 Datasets

In our experiments, we utilize two real-world taxi trajectory datasets collected from the cities of Chengdu from Didi Chuxing<sup>1</sup> and Harbin [27] in China. We remove trajectories that traveled less than 500 meters or 5 minutes, or more than 1 hour during pre-processing. Then, we filter out sparse trajectories by setting the minimum sampling rate to 80 seconds. The statistics of datasets after pre-processing are listed in Table 1.

### 6.2 Comparison Methods

To prove the superiority of the proposed method, we compare it with twelve baselines. Three are routing methods, two are path-based methods, and the others are ODT-Oracle methods.

**6.2.1 Routing Methods.** These methods identify the optimal path on the road network from origin to destination. We provide them with a weighted road network, where the weights represent the average travel time of road segments that is calculated from historical trajectories. The travel time is the sum of the historical average travel time of the road segments in the identified path.

- Dijkstra Shortest Path (**Dijkstra**) [23]: calculates the path between origin and destination with the smallest weights.
- Deep Probabilistic Spatial Transition (**DeepST**) [26]: generates the most probable traveling path between origin and destination based on the learned historical travel behaviors.

**6.2.2 Path-based Methods.** These methods predicts the travel time given the travel path. Since the real travel path is absent in the scenario of ODT-Oracle, we feed these methods with the path generated by DeepST.

- Wide-Deep-Double Recurrent model with Auxiliary loss (**WDDRA**) [11]: utilizes multi-tasking auxiliary loss to improve the accuracy of travel time estimation.
- Automated Spatio-Temporal Dual Graph Convolutional Networks (**STDGCN**) [21]: implement the neural architecture search techniques to automatically identify the optimal network structure.

**6.2.3 ODT-Oracle Methods.** These methods aim to predict travel time based on ODT-Input, serving as direct comparison to the proposed method. Four of them are traditional methods, the others are neural network-based methods.

- Temporally weighted neighbors (**TEMP**) [48]: averages the travel times of historical trajectories that have a similar origin, destination and departure time.
- Linear Regression (**LR**): learns a linear map from ODT-Inputs to travel times from historical travels.

<sup>1</sup><https://gaia.didichuxing.com/>

**Table 2: Hyper-parameter range and optimal values.**

| Parameter | Range                         |
|-----------|-------------------------------|
| $L_G$     | 10, 15, <u>20</u> , 25, 30    |
| $N$       | 500, <u>1000</u> , 1500, 2000 |
| $L_D$     | 1, 2, <u>3</u> , 4            |
| $d_E$     | 32, 64, <u>128</u> , 256      |
| $L_E$     | 1, <u>2</u> , 3, 4            |

Underline denotes the optimal value.

- Gradient Boosted Machine (**GBM**): a non-linear regression method, which is implemented using XGBoost [4].
- Road Network Vertex Embedding (**RNE**) [17]: calculates the shortest path distances between vertices in the embedding space.
- Spatial Temporal Neural Network (**ST-NN**) [22]: jointly predicts the travel distance and time given origin and destination.
- Multi-task Representation learning for Arrival Time estimation (**MURAT**) [29]: jointly predicts the travel distance and travel time given origin, destination and departure time.
- Effective Travel Time Estimation (**DeepOD**) [58]: incorporates the correlation between ODT-Inputs and travel trajectories from history through an auxiliary loss during training.

### 6.3 Experimental Settings

For both datasets, we first sort trajectories by their departure date and time. Then, we split them into training, validation and testing sets by 8:1:1. The PiT inference model is trained on the training set for 50 epochs, then used for PiT inference on validation and testing sets. The PiT travel time estimation is trained on the training set, early-stopped on the inferred validation set, and calculated final metrics on the inferred testing set. We use root mean squared error (RMSE), mean absolute error (MAE) and mean absolute percentage error (MAPE) to evaluate the precision of ODT-Oracles.

All methods are implemented using Python and PyTorch [35]. Baselines are implemented following the parameters suggested in their original papers. For the hyper-parameters of the proposed method, we consider 5 key parameters in different modules, their range and optimal values are listed in Table 2. It's worth mentioning that all hyper-parameters are chosen based on MAE results on the validation set. We also demonstrate the effectiveness of these hyper-parameters on the testing set later.

During model training, we choose the Adam optimizer and an initial learning rate of 0.001 across the board. We run all experiments on Ubuntu 20.04 servers equipped with Intel(R) Xeon(R) W-2155 CPUs and nVidia(R) TITAN RTX GPUs.

### 6.4 Comparison with Baselines

**6.4.1 Overall effectiveness comparison.** Table 3 demonstrates the comparison between different methods on overall effectiveness. The proposed method consistently shows its performance is superior over the two datasets.

The performance of routing methods in estimating travel time is largely dependent on the accuracy of their calculated routes. Dijkstra primarily focuses on finding shortest paths, which results in less accurate travel time estimations. In contrast, DeepST learns

**Table 3: Overall travel time estimation performance of different approaches.**

| Datasets          | Chengdu/Harbin      |                     |                       |          |
|-------------------|---------------------|---------------------|-----------------------|----------|
|                   | Metric              | RMSE (minutes)      | MAE (minutes)         | MAPE (%) |
| Dijkstra          | 9.677/11.865        | 7.618/8.447         | 48.618/55.261         |          |
| DeepST            | 4.717/8.926         | 3.452/5.849         | 27.503/37.772         |          |
| WDDRA             | 4.581/8.836         | 3.210/5.705         | 24.553/35.617         |          |
| STDGCN            | 4.469/8.679         | 3.104/5.564         | 23.187/33.771         |          |
| TEMP              | 5.578/10.150        | 4.267/7.891         | 36.611/66.781         |          |
| LR                | 6.475/10.290        | 5.036/8.006         | 44.514/67.669         |          |
| GBM               | 4.999/9.069         | 3.655/6.748         | 29.636/54.413         |          |
| RNE               | 4.624/8.571         | 3.416/6.245         | 27.660/47.956         |          |
| ST-NN             | 3.961/8.492         | 2.803/6.114         | 21.532/45.891         |          |
| MURAT             | <u>3.646</u> /7.937 | 2.384/5.360         | 18.345/41.128         |          |
| DeepOD            | 3.764/7.859         | <u>1.789</u> /4.533 | 14.997/36.974         |          |
| <b>DOT (Ours)</b> | <b>3.177</b> /7.462 | <b>1.272</b> /3.213 | <b>11.343</b> /26.698 |          |

**Bold** denotes the best result, and underline denotes the second-best result.

travel behavior from historical trajectory data, leading to significantly improved estimation accuracy.

The travel time estimation accuracy of path-based methods heavily relies on the quality of the input travel paths. Both WDDRA and STDGCN exhibit slightly improved performance compared to their path provider, DeepST, as they are capable of learning complex correlations between travel paths and travel time. STDGCN achieves higher accuracy compared to WDDRA, taking advantage of the neural architecture search technique.

The four traditional ODT-Oracle methods, TEMP, LR, GBM, and RNE, tackle the ODT-Oracle problem through feature engineering and kernel design. These methods face difficulties in modeling the intricate correlations between ODT-Inputs and travel times, and they do not take historical trajectories into account, which is essential for accurate ODT-Oracle predictions. Among them, the linear method LR performs the worst, suggesting that the spatial-temporal features in ODT-Inputs and travel times do not have a linear correlation. The history average method TEMP gets the second-worst result, primarily due to the imbalance of historical trips under different ODT-Inputs and the presence of outliers in historical trajectories. GBM outperforms TEMP and LR, attributable to its relatively higher model capacity. RNE achieves the best performance among traditional ODT-Oracle methods, as it incorporates hierarchical embeddings to capture the distances between locations more effectively.

The four neural network-based ODT-Oracle methods consistently outperform their traditional counterparts. It indicates that neural network-based methods excel at extracting complex spatial-temporal correlations between ODT-Inputs and travel times, thanks to their higher model capacity. Therefore, they can learn more appropriate representations for spatial-temporal features. This conclusion is even more obvious when comparing GBM with ST-NN, whose inputs are the same, with only the origin and destination. We can observe that ST-NN has a much better result than GBM. We can also observe that MURAT outperforms ST-NN in both datasets on all metrics. MURAT considers more comprehensive factors, e.g., road network, spatial cells and temporal slots. However, they have

**Table 4: Scalability of methods on Chengdu, measured by MAPE(%).**

| Scale             | 20%           | 40%           | 60%           | 80%           | 100%          |
|-------------------|---------------|---------------|---------------|---------------|---------------|
| Dijkstra          | 57.231        | 54.802        | 53.261        | 52.218        | 48.618        |
| DeepST            | 32.635        | 29.700        | 28.864        | 27.848        | 27.503        |
| WDDRA             | 31.081        | 29.475        | 27.005        | 25.756        | 24.553        |
| STDGCN            | 30.305        | 28.269        | 26.987        | 25.409        | 23.187        |
| TEMP              | 56.451        | 49.361        | 46.392        | 41.461        | 36.611        |
| LR                | 90.412        | 77.206        | 61.451        | 48.652        | 44.514        |
| GBM               | 43.592        | 38.635        | 34.322        | 32.405        | 29.636        |
| RNE               | 38.386        | 31.129        | 29.700        | 28.838        | 27.660        |
| ST-NN             | 27.916        | 24.854        | 23.548        | 22.889        | 21.532        |
| MURAT             | 24.975        | 22.251        | 20.519        | 19.431        | 18.345        |
| DeepOD            | 18.003        | 17.253        | 16.128        | 15.380        | 14.997        |
| <b>DOT (Ours)</b> | <b>14.951</b> | <b>14.034</b> | <b>13.014</b> | <b>12.486</b> | <b>11.343</b> |

**Bold** denotes the best result, and underline denotes the second-best result.

not considered taking advantage of the historical trajectories until DeepOD. We can observe that DeepOD can achieve the second best in most cases but is still worse than our method. It indicates that inappropriate handling of outliers in historical trajectories can hurt the performance of the ODT-Oracle.

The proposed method DOT gets the best results on both datasets. We transform raw trajectories into Pixelated Trajectories (PiTs) to make our model more robust on localized, small differences in trajectories. During training, we explicitly model the spatial-temporal correlation between ODT-Inputs and PiTs from historical trips. During travel time estimation, we infer the most probable PiT given a future ODT-Input and utilize the inferred PiT for accurate travel time estimation. The performance superiority of DOT demonstrates the effectiveness of inferring a robust representative form of travel trajectories and avoiding the impact of outliers in historical trajectories.

**6.4.2 Scalability comparison.** To evaluate the scalability of various methods, we sampled the training set of Chengdu by 20%, 40%, 60%, 80% and 100%, then tested the MAPE of different methods trained on the sampled training set, whose results are demonstrated in Table 4.

Generally speaking, all methods benefit from larger-scale training data. Since larger data have higher density, we can improve the generalization and robustness of the trained models. When the scale of training data decreases, the performance downgrade over different methods demonstrates their scalability. The proposed method remains relatively stable and consistently achieves the best result compared with other methods under the same circumstances. We can also observe that our worst performance at scale 20%, 14.951, is even better than that of DeepOD at scale 100%, 14.997. It indicates that the proposed method can work in more data-scarce scenarios than other methods.

**6.4.3 Efficiency comparison.** To investigate the efficiency, we calculate the model size, training time and estimation speed of different methods on Chengdu. The model size demonstrates the required memory size when running these methods. The training time and estimation speed show the methods' efficiency during training

**Table 5: Efficiency of methods on Chengdu.**

| Property          | Model size (Bytes) | Training time (minutes/epoch) | Estimation speed (seconds/K queries) |
|-------------------|--------------------|-------------------------------|--------------------------------------|
| Dijkstra          | 3.16M              | -                             | 0.95                                 |
| DeepST            | 5.40M              | 2.33                          | 2.74                                 |
| WDDRA             | 6.79M              | 1.43                          | 2.42                                 |
| STDGCN            | 5.50M              | 2.97                          | 3.29                                 |
| TEMP              | 4.45M              | -                             | 5.73                                 |
| LR                | 0.59K              | 0.22                          | 0.21                                 |
| GBM               | 0.76K              | 1.23                          | 0.39                                 |
| RNE               | 0.78M              | 0.42                          | 0.34                                 |
| ST-NN             | 0.30M              | 0.34                          | 0.33                                 |
| MURAT             | 7.85M              | 1.41                          | 1.65                                 |
| DeepOD            | 6.24M              | 1.26                          | 1.62                                 |
| <b>DOT (Ours)</b> | <b>7.32M</b>       | <b>3.04/1.22</b>              | <b>1.85</b>                          |

and real-world travel time estimation. Note that the two stages of the proposed method are trained separately, so we give these two training times separately.

The storage of the weighted road network in Dijkstra takes up some memory. It doesn't require any training, and expedite techniques to speed up their routing process on road network. On the other hand, the data-driven routing method DeepST exhibits a relatively slow training and estimation speed, primarily due to the use of RNN sequential model [16]. RNNs cannot parallelize on path sequences, which consequently hinders their estimation speed. This same reason accounts for the slow estimation speed of the two path-based methods, WDDRA and STDGCN, as they also employ RNNs for processing the input path sequences.

Compared with other neural network methods, two traditional methods LR and GBM have smaller model sizes. GBM trains and predicts slower than LR, since it is a non-linear method and contains multiple decision trees for integrated learning. TEMP is an average history method that does not need training. Yet, it needs to cache all historical trips and calculate distances for every pair of trips between queries and training set during prediction. Thus, the model size and prediction speed of TEMP scale nearly linearly with the dataset size, which is problematic for large-scale datasets.

We can observe that ST-NN has the simplest design and smallest model size among neural network-based methods. So, it is the fastest in both training and predicting. RNE stores embeddings for all segments in the road network, resulting in slightly reduced efficiency compared to ST-NN. We can also observe that the proposed DOT is relatively slower during its training due to its two-stage framework. However, the prediction speed catches up with the state-of-the-art neural network-based methods since we propose an efficient estimation module, MViT. Since the training phase is always completed offline, we can claim that the efficiency of our method is on par with the existing inference methods.

## 6.5 Quantitative Analysis

**6.5.1 Effectiveness of outlier removal.** We assess the potential performance improvements that can be achieved by combining outlier removal methods with existing baselines. We implement the state-of-the-art outlier detection method DeepTEA [13] to eliminate outlier trajectories from the training set. Then, we re-train a select set

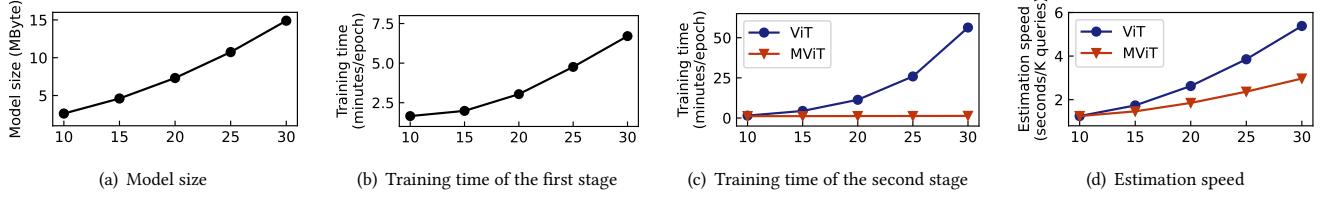
Figure 8: Efficiency impact of grid length  $L_G$ .

Table 6: Performance of baselines with outlier detection.

| Datasets          | Chengdu/Harbin              |                             |                               |
|-------------------|-----------------------------|-----------------------------|-------------------------------|
| Metric            | RMSE (minutes)              | MAE (minutes)               | MAPE (%)                      |
| Dijkstra+DeepTEA  | 9.641/11.862                | 7.582/8.396                 | 48.337/53.949                 |
| DeepST+DeepTEA    | 4.692/8.901                 | 3.416/5.821                 | 26.959/37.063                 |
| WDDRA+DeepTEA     | 4.497/8.584                 | 3.140/5.545                 | 23.537/34.723                 |
| STDGCN+DeepTEA    | 4.393/8.569                 | 3.056/5.501                 | 22.812/33.688                 |
| RNE+DeepTEA       | 4.627/8.403                 | 3.447/6.061                 | 28.239/45.345                 |
| ST-NN+DeepTEA     | 3.912/8.427                 | 2.740/5.994                 | 20.818/43.664                 |
| MURAT+DeepTEA     | <u>3.644</u> /7.899         | 2.367/5.181                 | 17.986/37.728                 |
| DeepOD+DeepTEA    | 3.763/7.817                 | <u>1.783</u> / <u>4.345</u> | <u>14.835</u> / <u>33.127</u> |
| <b>DOT (Ours)</b> | <b>3.177</b> / <b>7.462</b> | <b>1.272</b> / <b>3.213</b> | <b>11.343</b> / <b>26.698</b> |

**Bold** denotes the best result, and underline denotes the second-best result.

of baselines and evaluate their travel time estimation performance. The results are presented in Table 6.

All baselines experience performance improvements, with the exception of RNE on Chengdu. This demonstrates that proper handling of outliers in historical trajectories can be advantageous for achieving accurate travel time estimation. Nevertheless, the proposed method continues to outperform these baselines even after outlier removal, for two main reasons. First, as illustrated in Figure 2, the proposed PiT representation enables our method to more effectively identify outliers that significantly affect travel time estimation, while ignoring negligible differences in trajectories. Second, we design our PiT inference stage as a diffusion-based generative process, which is more robust at modeling the distribution of historical trajectories with outliers, separating outliers from normal trajectories. This approach has also been demonstrated to be effective in other studies such as [15, 44].

**6.5.2 Impact of grid length.** To study the impact of different resolutions of PiT, we vary the grid length,  $L_G$ , and investigate the efficiency at different grid lengths, i.e., the model size, the training efficiency at the first and second stages, and the inferring efficiency. From Figures 8(a) and 8(b), we can observe that the model size and the training time of the first stage increase with the increasing of  $L_G$ . It is because the PiT becomes larger when  $L_G$  increases, bringing increased kernel sizes and filters in the conditioned PiT denoiser. Figure 8(c) shows that the proposed MViT scales well with the increase of  $L_G$  compared with the vanilla ViT in training at the second stage. It indicates that the total number of grid cells PiT occupies is almost unchanged, but the proportion of occupied grid cells is becoming less, so MViT can perform much better than ViT. We can see from Figure 8(d) that MViT also improves the estimation speed significantly compared to ViT. We can also observe that there is no clear difference between MViT and ViT at  $L_G = 10$ . PiT occupies a much larger proportion of grids when the grid size

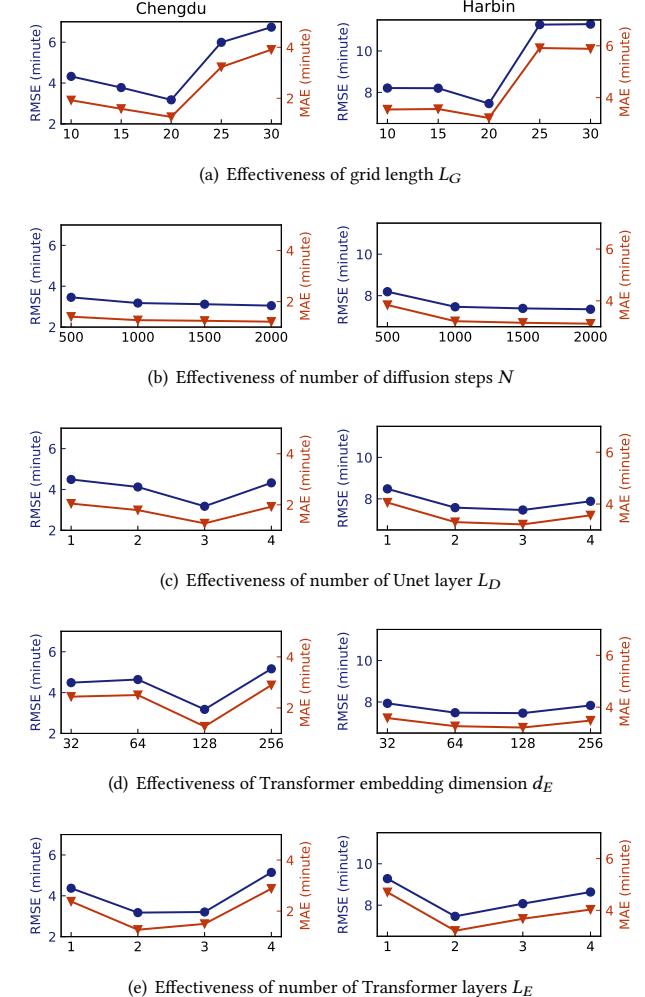


Figure 9: Effects of hyper-parameters on ODT-Oracle performance.

becomes larger. Yet, the total number of grid cells is the least at  $L_G = 10$ , resulting in the least training time for ViT. Finally, we can conclude that the proposed MViT can improve the framework's efficiency in both training and estimating.

**6.5.3 Impact of hyper-parameters.** The optimal hyper-parameters listed in Table 2 are selected with parameter experiments on validation sets. In this section, we further demonstrate the effectiveness

**Table 7: Effects of features and modules on ODT-Oracle performance.**

| Datasets      | Chengdu/Harbin     |                    |                      |
|---------------|--------------------|--------------------|----------------------|
| Metric        | RMSE (minutes)     | MAE (minutes)      | MAPE (%)             |
| Dijkstra+Est. | 9.182/11.869       | 6.871/8.246        | 41.462/50.488        |
| DeepST+Est.   | 4.587/8.879        | 3.170/5.689        | 23.437/33.769        |
| Infer.+WDDRA  | 3.773/7.958        | 1.801/4.171        | 18.937/31.514        |
| Infer.+STDGCN | 3.476/7.611        | 1.664/3.818        | 17.653/29.756        |
| No-t          | 4.325/8.798        | 1.926/4.345        | 16.820/35.973        |
| No-od         | 7.355/10.947       | 4.564/6.333        | 38.879/51.699        |
| No-odt        | 8.466/11.172       | 5.880/6.562        | 49.830/53.331        |
| No-CE         | 3.778/8.584        | 1.591/4.144        | 14.034/34.441        |
| No-ST         | 7.784/11.023       | 5.036/6.427        | 42.850/52.442        |
| Est-CNN       | 6.297/10.389       | 3.500/5.765        | 30.004/47.166        |
| Est-ViT       | <u>3.229/7.390</u> | <u>1.293/3.187</u> | <u>11.547/26.484</u> |
| <b>DOT</b>    | <b>3.177/7.462</b> | <b>1.272/3.213</b> | <b>11.343/26.698</b> |

**Bold** denotes the best result, and underline denotes the second-best result.

of these key hyper-parameters on testing sets, which is shown in Figure 9. We have the following observations.

- (1) As is indicated in Figure 9(a), it is optimal to set grid length  $L_G$  to 20. A smaller grid length results in coarse-grained PiTs, which is insufficient to estimate travel time accurately. On the other hand, a bigger grid length increases the sparsity of PiTs, resulting in the model being more sensitive to negligible differences in trajectories. Interestingly, increasing the grid length hurts the prediction performance more than decreasing it. This shows that it is better to model trajectories into PiTs, which can reduce the disturbance to accurate travel time estimation.
- (2) Figure 9(b) indicates that more diffusion steps lead to better results, which is intuitive that we can learn the noise better with more steps. However, the gain becomes less when  $N$  is larger than 1000. Therefore, we select  $N = 1000$  as a good trade-off between effectiveness and efficiency.
- (3)  $L_D, d_E, L_E$  determine the representation capacity of the PiT inference model and the PiT travel time estimation. Figures 9(c), 9(d) and 9(e) demonstrate their effectiveness respectively. All of them have optimal values. A too-small model struggles to extract the complex spatial-temporal correlations in datasets, while a too-big model leads to overfitting.

**6.5.4 Ablation study.** To verify the effectiveness of features and modules in the proposed method, we conduct an ablation study in the following variants of DOT methods and input features.

- (1) *Routing+Est.:* combine the routing methods listed in Section 6.2.1 with the PiT travel time estimation stage of our method.
- (2) *Infer+Path-based:* integrates the PiT inference stage with the path-based approaches presented in Section 6.2.2.
- (3) *No-t:* remove the departure time  $t_o$  from the ODT-Input *odt*.
- (4) *No-od:* remove the origin and destination coordinates from *odt*.
- (5) *No-odt:* remove the conditional information *odt* completely.
- (6) *No-CE:* remove the cell embedding module from the PiT travel time estimation.
- (7) *No-ST:* remove the latent casting module from the PiT travel time estimation.
- (8) *Est-CNN:* replace MViT with a CNN-based estimator.

**Table 8: PiT inference accuracy of our model.**

| Datasets           | Chengdu/Harbin |             |
|--------------------|----------------|-------------|
| Metric             | RMSE           | MAE         |
| Overall            | 0.196/0.181    | 0.027/0.023 |
| Channel 1 (Mask)   | 0.271/0.224    | 0.039/0.028 |
| Channel 2 (ToD)    | 0.128/0.183    | 0.016/0.024 |
| Channel 3 (Offset) | 0.159/0.123    | 0.025/0.016 |

(9) *Est-ViT:* replace MViT with the vanilla vision Transformer.

We compare the performance of these variants with the DOT method, and the experimental results are given in Table 7. We have the following observations:

- (1) Combining the routing methods with the proposed PiT travel time estimation stage failed to yield satisfactory results. This is mainly because the routes inferred by these methods are not accurate enough, which is also demonstrated in Table 9. Additionally, these routings methods do not generate temporal features, i.e., the second and third channels in PiTs, and these features are instead populated based on historical average travel times between cells. In contrast, the proposed method infers both spatial and temporal features of PiTs based on the learned spatio-temporal information from historical trajectories, which benefits travel time estimation accuracy.
- (2) Integrating the proposed PiT inference stage with the path-based methods enhances their estimation performance compared to the results in Table 3. The improvements can be attributed to the higher accuracy of the inferred routes compared to those produced by DeepST. However, the results still do not surpass those obtained by DOT, as the RNN sequential models employed in WDDRA and STDGCN are not as effective as the proposed MViT in modeling spatio-temporal correlations in PiTs, as also shown by other studies of time series mining [45, 60].
- (3) Removing features from the conditional ODT-Input reduces the estimation performance, since the inferred PiT cannot accurately correspond to the given ODT-Input.
- (4) Removing the embedding modules from the PiT travel time estimation also makes prediction worse, since the spatial-temporal information embedded in PiT is not comprehensively utilized.
- (5) Comparing results from Est-CNN and DOT proves that our Transformer-based MViT is more effective than CNN-based methods when dealing with PiT. Comparing results from Est-ViT and DOT shows that the estimation performance of MViT is very close to ViT. Combining the efficiency comparison in Figure 8, the proposed MViT can improve the efficiency over ViT while not compromising on estimation performance.

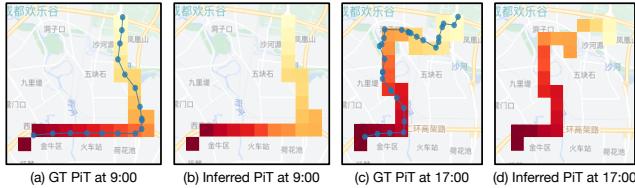
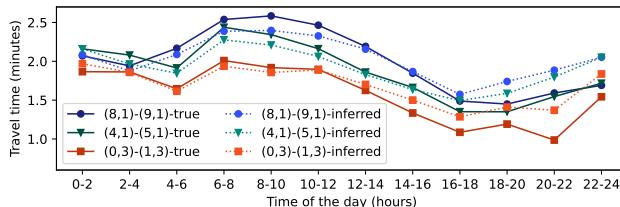
## 6.6 Analysis on the Explainability

To evaluate the explainability of the proposed method, we first conduct some quantitative experiments to demonstrate the accuracy of the inferred trajectories. We then conduct a case study to visualize the inferred trajectories.

**6.6.1 PiT inference accuracy.** We conduct experiments on both datasets to verify the accuracy of the inferred PiTs. Specifically,

**Table 9: Accuracy of route inferences.**

| Datasets          | Chengdu/Harbin       |                      |                      |
|-------------------|----------------------|----------------------|----------------------|
| Metric            | Pre(%)               | Rec(%)               | F1(%)                |
| Dijkstra          | 68.918/45.459        | 31.310/42.525        | 42.065/39.993        |
| DeepST            | 59.755/74.519        | 55.776/62.907        | 56.911/66.029        |
| <b>DOT (Ours)</b> | <b>87.890/88.190</b> | <b>88.684/88.982</b> | <b>88.280/88.584</b> |

**Figure 10: Trajectories for the same OD pair and departure at the same time of the day.****Figure 11: Trajectories for the same OD pair that depart at different times of the day.****Figure 12: Average travel time between spatial cells during a day.**

given the ODT-Inputs from the testing set, we calculate the RMSE and MAE between the inferred PiTs and the ground truth ones, whose results are listed in Table 8.

We also compare the accuracy of the inferred routes from DOT with the planned routes in Dijkstra and DeepST. Specifically, the routes planned by these methods are transformed into the same form as the first (mask) channel in PiT. We then compare the accuracy of the mask channels calculated by these methods with those inferred by our model, using precision (Pre), recall (Rec), and F1 scores as metrics. The results are presented in Table 9.

We can observe that we achieve a pretty accurate PiT inference and route inference on both datasets, which helps us to achieve good performance in the second stage of travel time estimation. Accurate route inference also means that our model can provide the users with the most probable route that is chosen by most drivers, given a future travel plan.

**6.6.2 Case study.** We conduct two case studies on the testing set of Chengdu.

For the first case study, we visualize the raw trajectories and the third channel (time offset) of both the ground truth and inferred PiTs under certain circumstances. We investigate the following circumstances with trajectories that travel from the same origin to the same destination.

- (1) *Trajectories depature during the same time:* In Figure 10(a), we can observe that two PiTs are almost the same when departing during the same time of the day, where the second PiT has extra cells compared to the first one, which is an outlier. Figure 10(b) shows the PiT inferred from the first stage by giving the ODT-Input with the same OD and T. We can observe that the inferred PiT matches the ground truth well, where the small portion of outliers in the second PiT is removed.
- (2) *Trajectories departure during different time:* Figure 11 demonstrates the case with the same origin-destination pair but a different departure time. It indicates that different travel trajectories can be chosen during different times of the day, which makes a big difference in the travel time estimation. Since the proposed method considers the departure time, it can infer PiTs at different departure times.

In the second case study, we examine the evolution of the average travel time between pairs of spatial cells during a day. We select the top-3 most frequently traveled pairs of cells and calculate the average travel time between them by dividing a day into two-hour intervals. Figure 12 display the average travel time calculated from ground truth trajectories and the inferred PiTs of DOT, where the cells are denoted by their coordinates specified in Definition 2. The travel time calculated from the inferred PiTs varies throughout the day, indicating that our method accounts for the evolving traffic conditions. Furthermore, it closely matches the travel time calculated from ground truth trajectories, signifying the accuracy of the temporal features in the inferred PiTs.

In conclusion, the proposed method's conditioned PiT inference model can learn from history and infer PiT given a future ODT-Input. It ensures the accurate performance of travel time estimation, which is highly dependent on the spatial-temporal information of trajectories. In addition, the inferred PiT gives an intuitive overview of the future trip, improving the explainability of the model, and providing users with useful information.

## 7 CONCLUSIONS

To build an accurate and explainable ODT-Oracle, we propose a two-stage framework called DOT. In the first stage, we propose a conditioned PiT denoiser to implement a PiT inference model. This model can learn from historical trajectories and can infer a PiT given an OD pair and a departure time, which reduce the impact of outlier historical trajectories. In the second stage, we propose an MViT to model the global correlation of the inferred PiT efficiently, which can give an accurate estimation of travel time. Comprehensive experiments are conducted on two real-world datasets to demonstrate the performance superiority of the proposed method.

## REFERENCES

- [1] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. Structured denoising diffusion models in discrete state-spaces. In *NeurIPS*, Vol. 34. 17981–17993.
- [2] Xiaoqiang Cai, Jian Chen, Yongbo Xiao, Xiaolin Xu, and Gang Yu. 2013. Fresh-product supply chain management with logistics outsourcing. *Omega* 41, 4 (2013). 752–765.
- [3] Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. 2020. A survey on map-matching algorithms. In *ADC*. Springer, 121–133.
- [4] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *ACM SIGKDD*. 785–794.
- [5] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP*. 1724–1734.
- [6] Theodoros Chondrogiannis, Johann Bornholdt, Panagiotis Bourou, and Michael Grossniklaus. 2022. History oblivious route recovery on road networks. In *SIGSPATIAL*. 1–10.
- [7] Teodor Gabriel Crainic and Gilbert Laporte. 1997. Planning models for freight transportation. *European journal of operational research* 97, 3 (1997). 409–438.
- [8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*.
- [9] Jie Feng, Yong Li, Chao Zhang, Funing Sun, Fanchao Meng, Ang Guo, and Depeng Jin. 2018. Deepmove: Predicting human mobility with attentional recurrent networks. In *WWW*. 1459–1468.
- [10] Kun Fu, Fanlin Meng, Jieping Ye, and Zheng Wang. 2020. Compacteta: A fast inference system for travel time prediction. In *ACM SIGKDD*. 3337–3345.
- [11] Yunchong Gan, Haoyu Zhang, and Mingjie Wang. 2021. Travel Time Estimation Based on Neural Network with Auxiliary Loss. In *SIGSPATIAL*. 642–645.
- [12] Qing Guo, Zhu Sun, Jie Zhang, and Yin-Leng Theng. 2020. An attentional recurrent neural network for personalized next location recommendation. In *AAAI*, Vol. 34. 83–90.
- [13] Xiaolin Han, Reynold Cheng, Chenhao Ma, and Tobias Grubemann. 2022. DeepTEA: effective and efficient online time-dependent trajectory outlier detection. *Proc. VLDB Endow.* 15, 7 (2022). 1493–1505.
- [14] Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).
- [15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. In *NeurIPS*, Vol. 33. 6840–6851.
- [16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997). 1735–1780.
- [17] Shuai Huang, Yong Wang, Tianyu Zhao, and Guoliang Li. 2021. A learning-based method for computing shortest path distances on road networks. In *ICDE*. 360–371.
- [18] Tsuyoshi Idé and Masashi Sugiyama. 2011. Trajectory regression on road networks. In *AAAI*, Vol. 25. 203–208.
- [19] Christopher Jarzynski. 1997. Equilibrium free-energy differences from nonequilibrium measurements: A master-equation approach. *Physical Review E* 56, 5 (1997). 5018.
- [20] Christian S. Jensen and Nerius Tradišauskas. 2009. Map Matching. In *Encyclopedia of Database Systems*. 1692–1696.
- [21] Guangjin Jin, Huan Yan, Fuxian Li, Yong Li, and Jincui Huang. 2021. Hierarchical neural architecture search for travel time estimation. In *SIGSPATIAL*. 91–94.
- [22] Ishan Jindal, Xuewen Chen, Matthew Nokleby, Jieping Ye, et al. 2017. A unified neural network approach for estimating travel time and distance for a taxi trip. *arXiv preprint arXiv:1710.04350* (2017).
- [23] Donald B Johnson. 1973. A note on Dijkstra's shortest path algorithm. *J. ACM* 20, 3 (1973). 385–388.
- [24] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *NeurIPS*, Vol. 30. 3146–3154.
- [25] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. 2020. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761* (2020).
- [26] Xiucheng Li, Gao Cong, and Yun Cheng. 2020. Spatial transition learning on road networks with deep probabilistic models. In *ICDE*. 349–360.
- [27] Xiucheng Li, Gao Cong, Aixin Sun, and Yun Cheng. 2019. Learning travel time distributions with deep generative model. In *WWW*. 1017–1027.
- [28] Yaguang Li, Dingxiang Deng, Ugur Demiryurek, Cyrus Shahabi, and Siva Ravada. 2015. Towards fast and accurate solutions to vehicle routing in a large-scale and dynamic environment. In *SSTD*. 119–136.
- [29] Yaguang Li, Kun Fu, Zheng Wang, Cyrus Shahabi, Jieping Ye, and Yan Liu. 2018. Multi-task representation learning for travel time estimation. In *ACM SIGKDD*. 1695–1704.
- [30] Yan Lin, Huaiyu Wan, Shengnan Guo, and Youfang Lin. 2021. Pre-training context and time aware location embeddings from spatial-temporal trajectories for user next location prediction. In *AAAI*, Vol. 35. 4241–4248.
- [31] Todd Litman. 2009. Transportation cost and benefit analysis. *Victoria Transport Policy Institute* 31 (2009). 1–19.
- [32] Huiping Liu, Cheqing Jin, Bin Yang, and Aoying Zhou. 2017. Finding top-k shortest paths with diversity. *IEEE Trans. on Know. and Data Eng.* 30, 3 (2017). 488–502.
- [33] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A convnet for the 2020s. In *IEEE CVPR*. 11976–11986.
- [34] Brian K Nelson. 1998. Time series analysis using autoregressive integrated moving average (ARIMA) models. *Academic emergency medicine* 5, 7 (1998). 739–744.
- [35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 8024–8035.
- [36] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *IEEE CVPR*. 10684–10695.
- [37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*. 234–241.
- [38] Sijie Ruan, Zi Xiong, Cheng Long, Yiheng Chen, Jie Bao, Tianfu He, Ruiyuan Li, Shengnan Wu, Zhongyuan Jiang, and Yu Zheng. 2020. Doing in One Go: Delivery Time Inference Based on Couriers' Trajectories. In *ACM SIGKDD*. 2813–2821.
- [39] Jagan Sankaranarayanan and Hanan Samet. 2009. Distance Oracles for Spatial Networks. In *ICDE*. 652–663.
- [40] Jagan Sankaranarayanan, Hanan Samet, and Houman Alborzi. 2009. Path Oracles for Spatial Networks. *Proc. VLDB Endow.* 2, 1 (2009). 1210–1221.
- [41] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*. 2256–2265.
- [42] Yang Song and Stefano Ermon. 2019. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, Vol. 32. 11895–11907.
- [43] Fuyong Sun, Ruipeng Gao, Weiwei Xing, Yaoxue Zhang, Wei Lu, Jun Fang, and Shui Liu. 2022. Deep Fusion for Travel Time Estimation Based on Road Network Topology. *IEEE Intelligent Systems* 37, 3 (2022). 98–107.
- [44] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. 2021. CSID: Conditional score-based diffusion models for probabilistic time series imputation. In *NeurIPS*, Vol. 34. 24804–24816.
- [45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*, Vol. 30. 5998–6008.
- [46] Chenxing Wang, Fang Zhao, Haichao Zhang, Haiyong Luo, Yanjun Qin, and Yuchen Fang. 2022. Fine-Grained Trajectory-Based Travel Time Estimation for Multi-City Scenarios Based on Deep Meta-Learning. *IEEE Trans. on Intelli. Trans. Sys.* 23, 9 (2022). 15716–15728.
- [47] Dong Wang, Junbo Zhang, Wei Cao, Jian Li, and Yu Zheng. 2018. When will you arrive? Estimating travel time based on deep neural networks. In *AAAI*, Vol. 32. 2500–2507.
- [48] Hongjian Wang, Xianfeng Tang, Yu-Hsuan Kuo, Daniel Kifer, and Zhenhui Li. 2019. A simple baseline for travel time estimation using large-scale trip data. *ACM Trans. on Intelli. Sys. and Tech.* 10, 2 (2019). 1–22.
- [49] Yuandong Wang, Hongzhi Yin, Tong Chen, Chunyang Liu, Ben Wang, Tianyu Wo, and Jie Xu. 2021. Passenger Mobility Prediction via Representation Learning for Dynamic Directed and Weighted Graphs. *ACM Trans. on Intelli. Sys. and Tech.* 13, 1 (2021). 1–25.
- [50] Yilun Wang, Yu Zheng, and Yexiang Xue. 2014. Travel time estimation of a path using sparse trajectories. In *ACM SIGKDD*. 25–34.
- [51] Zheng Wang, Kun Fu, and Jieping Ye. 2018. Learning to estimate the travel time. In *ACM SIGKDD*. 858–866.
- [52] Haomin Wen, Youfang Lin, Fan Wu, Huaiyu Wan, Shengnan Guo, Lixia Wu, Chao Song, and Yinghui Xu. 2021. Package pick-up route prediction via modeling couriers' spatial-temporal behaviors. In *IEEE ICDE*. 2141–2146.
- [53] Fan Wu and Lixia Wu. 2019. DeepETA: a spatial-temporal sequential neural network model for estimating time of arrival in package delivery system. In *AAAI*, Vol. 33. 774–781.
- [54] Hao Wu, Jiangyun Mao, Weiwei Sun, Baihua Zheng, Hanyuan Zhang, Ziyang Chen, and Wei Wang. 2016. Probabilistic robust route recovery with spatio-temporal dynamics. In *SIGKDD*. 1915–1924.
- [55] Jiajia Xu, Sajju Xu, Rui Zhou, Chengfei Liu, An Liu, and Lei Zhao. 2021. TAML: A Traffic-aware Multi-task Learning Model for Estimating Travel Time. *ACM Trans. on Intelli. Sys. and Tech.* 12, 6 (2021). 1–14.
- [56] Sajju Xu, Jiajia Xu, Rui Zhou, Chengfei Liu, Zhiwu Li, and An Liu. 2020. Tadnn: A transportation-mode aware deep neural model for travel time estimation. In *DASFAA*. 468–484.
- [57] Ling Yang, Shouxu Jiang, and Fusheng Zhang. 2022. Multitask Learning with Graph Neural Network for Travel Time Estimation. *Computational Intelligence and Neuroscience* 2022 (2022).

- [58] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2020. Effective travel time estimation: When historical trajectories over road networks matter. In *SIGMOD*. 2135–2149.
- [59] Junbo Zhang, Yu Zheng, Junkai Sun, and Dekang Qi. 2019. Flow prediction in spatio-temporal networks based on multitask deep learning. *IEEE Trans. on Know. and Data Eng.* 32, 3 (2019), 468–478.
- [60] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI*, Vol. 35. 11106–11115.
- [61] Linqi Zhou, Yilun Du, and Jiajun Wu. 2021. 3d shape generation and completion through point-voxel diffusion. In *IEEE ICCV*. 5826–5835.