

# A Pre-Trained Trajectory Generative Model on Road Network

Anonymous submission

## Abstract

Transportation research heavily relies on traffic simulations for their ability to test hypotheses, crucial for informing transportation policies. However, existing simulations often lack realism, limiting their applicability to the real world. In contrast, data-driven studies leverage actual traffic data for tasks like predictive modeling, but they typically focus only on a specific real-world dataset. Our proposed approach combines the strengths of both simulation and data-driven methods by introducing a generative vehicle trajectory model. Initially trained on simulated data using real roadmap information OpenStreetMap, our model is then aligned with real-world observations. Experiments on city-wide road networks with large-scale trajectory data demonstrate the effectiveness of our proposed method.

## Introduction

Transportation research has extensively relied on traffic simulations because they allow us to test various hypotheses, which is essential for transportation policymakers. However, a well-known drawback of existing traffic simulators is their lack of realism, which limits the applicability of research findings from simulations to the real world. A notable example is the extensive research on reinforcement learning for traffic signal control in recent years (Genders and Razavi 2016; Wei et al. 2018; Chen et al. 2020). Researchers have found that the real bottleneck is the simulator’s deviation from real-world conditions, making it difficult to apply the learned decision models effectively in real-world scenarios (Wei et al. 2021).

On the other hand, there is substantial research in the AI field that focuses on real traffic data. In particular, studies have explored generating trajectories by learning from actual vehicle movements in urban settings, e.g., the use of generative adversarial networks (GANs) (Jiang et al. 2023) and diffusion methods (Zhu et al. 2024a) to generate realistic trajectories. However, a key limitation of these approaches is that we can only learn from the observed data. In reality, it is impossible to capture all the trajectories within a city’s road network. Instead, we can only observe a very small subset of all possible trajectories. Learning exclusively from observed trajectories can lead to out-of-distribution (OOD) problems when attempting to apply these models to an entire city’s road network.

This paper adopts a novel approach to address the challenges inherent in the two research directions mentioned above. We propose to develop a pre-trained generative model using the vast amounts of data generated by the simulator and then further align this model with real-world observations. Compared to the simulator, our generative model offers greater realism. Additionally, unlike models that rely solely on real-world data, our approach can generalize more effectively to the entire city-wide road network.

We would like to emphasize that this pre-training step can be universally applied to any city, as the only requirement is the access to a simulator that runs on the city’s road network. In our experiment, we use SUMO (Lopez et al. 2018), a widely-adopted microscopic traffic simulator that is freely available for public use. SUMO seamlessly integrates with OpenStreetMap (OSM), the most comprehensive road network data source globally. This means we can pre-train a trajectory generative model for any city, which can then serve as a foundational model to be further aligned with real-world observations.

Our proposed model is an auto-regressive generative model built on the decoder-only Transformer architecture (Vaswani et al. 2017), which is particularly well-suited for trajectory data. A trajectory on the road network is essentially a sequence of road network edges that a vehicle traverses. To optimize the configuration of our generative model, we utilize a recently released real city-scale vehicle trajectory dataset obtained from traffic camera videos (Yu et al. 2023). This dataset, derived from two cities in China - Jinan and Shenzhen - encompasses road networks containing more than 20,000 road edges.

The experimental results highlight the effectiveness of combining simulated and real-world data. Although simulators can be calibrated using real data, the back-propagation learning mechanism in our model enables more efficient and effective alignment with real-world data compared to the brute-force search methods typically used in calibration (Yu and Fan 2017). Additionally, we demonstrate that even when observations are masked in certain regions of the city, generative models based purely on real data tend to produce trajectories that deviate significantly from the actual distribution, whereas our model can still generate realistic trajectories for these masked regions.

## Related Work

### Simulation and Calibration

Effectively generating traffic in city road network has been investigated for decades in transportation field. Researchers mainly follow two categories of methods to generate traffic. One way is based on a pipeline of classic traffic models, including traffic demands models (Flötteröd, Bierlaire, and Nagel 2011) to estimate origin and destination distribution, routing models (Ben Ticha et al. 2018), and dynamic traffic assignment models (Peeta and Ziliaskopoulos 2001; Wang et al. 2018) to infer how vehicles choose routes and traffic macroscopic fundamental diagrams (Daganzo and Geroliminis 2008) to infer vehicle speed and volume in regions and roads. Based on these traffic models, recent studies have started to build comprehensive traffic simulation tools like SUMO (Lopez et al. 2018) and AIMSIM (Barceló and Casas 2005). These simulations either describe how each vehicle proceeds and interacts with the environment or describe how the traffic status (e.g., volume, speed) of each road or region evolves when traffic enters the system. In order to make the generated results closer to the real world, researchers use real data to calibrate the traffic model or simulation via methods like Tabu search (Osorio and Punzo 2019).

However, these physical models and simulations may still deviate far from the real world traffic since their ideal assumptions may not hold or their parameters are not correct in the real world. Hence, in this paper, we propose a traffic generation model pre-trained with simulation data and further improved with real data.

### Traffic Generative Models

In recent years, studies have started using data-driven methods to generate traffic for cities when given traffic demands data (i.e., origin-destination data). From an objective perspective, studies mainly focus on learning vehicle trajectories (usually represented as GPS points) (Zhu et al. 2024b) and learning road indicators (e.g., volume, speed) (Zhang et al. 2020, 2019). In terms of methodologies, researchers have used various methods to achieve traffic generation. Studies use generative adversarial networks (Zhang et al. 2020), imitation learning or reinforcement learning methods (Zheng et al. 2021) as the learning paradigm to generate trajectories or road indicators. Various neural network structures, e.g., RNN (Yu et al. 2017), GNN and Transformer (Haydari et al. 2024; Zhu et al. 2024b,a; Wang et al. 2024), are used to capture the temporal and spatial relation among trajectories.

Current studies majorly suffer from the following three issues. First, most studies primarily address traffic generation at a regional level or through point-wise trajectories, ignoring road network constraints. As a result, vehicles can generate unrealistic trajectories out of the road. Second, current GAN-based methods are generating the “routes” rather than “trajectories with timestamps”. Hence, they fail to model how long it takes for a vehicle to pass a road and finish the trip. Third, current diffusion models based on GPS locations may mistakenly generate nonconsecutive trajectories.

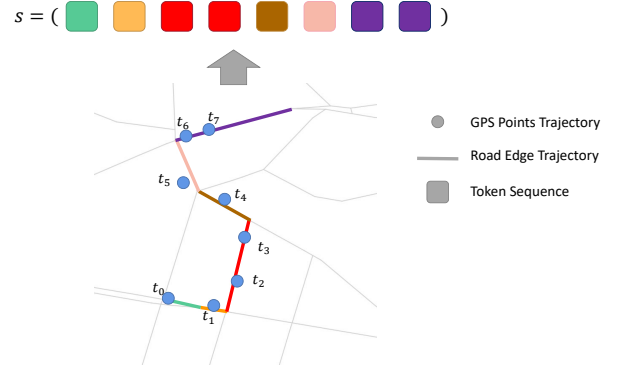


Figure 1: Discretizing an example continuous vehicle trajectory into a sequence of edges.

Adding noise to GPS trajectories may move the vehicle from the northbound side of a road to the southbound.

### Urban Foundation Model

Urban foundation models have become popular in recent two years since the coming of GPT (Achiam et al. 2023). Studies have emerged to work on very different problems. Some representative ones are described here, including but not limited to (1) representing urban regions or points using OpenStreetMap or satellite images (Balsebre et al. 2023; Yan et al. 2024); (2) making various spatial temporal predictions using one single model (Li et al. 2024); (3) utilizing the capability of LLM to assist traffic simulation process (Zhang et al. 2024). Though named urban foundation models, these studies are fairly distinct from our study since we are working on a different problem of city-scale traffic generation, which can benefit various downstream city-scale traffic tasks, e.g., prediction and policy control tasks.

### Problem Definition

Suppose the road network is represented as  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ , where  $\mathbf{V}$  is the set of vertices (intersections) and  $\mathbf{E}$  is the set of edges (road segments).

The movement of vehicles in cities can be described by their trajectories, i.e., a sequence of GPS points. Due to the fact that vehicles must run on road edges, the GPS trajectory of a vehicle can be mapped into the road network graph and converted to a road edge trajectory on the graph. Then, a trajectory is represented as  $s = (e_0, e_1, \dots, e_i, \dots, e_T)$ , where  $e_i \in \mathbf{E}$  denotes the road edge that the vehicle is running on at the  $i$ -th timestep, and  $T$  is the total number of time steps of this trajectory. Further,  $o = e_0$  and  $d = e_T$  can represent the origin and destination of this trajectory, respectively. Thus, a set of trajectories is represented as  $\mathbf{S} = \{s^{(1)}, s^{(2)}, \dots, s^{(j)}, \dots, s^{(N)}\}$  and their corresponding origin destination pairs  $\mathbf{OD} = \{(o^{(1)}, d^{(1)}), (o^{(2)}, d^{(2)}), \dots, (o^{(j)}, d^{(j)}), \dots, (o^{(N)}, d^{(N)})\}$ . Figure 1 provides an example of the trajectory of a vehicle.

**Road Edge Trajectory Learning Problem** Given a road network graph  $\mathbf{G}(\mathbf{V}, \mathbf{E})$ , and a set of origin destination pairs

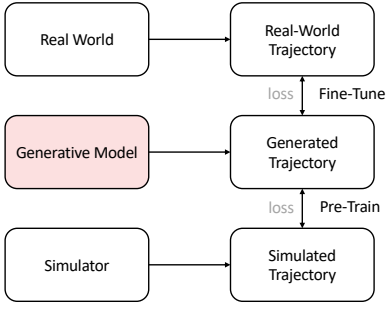


Figure 2: Illustration of our proposed method.

**OD**, the objective is to learn a mapping function  $q_\theta(s|o, d)$  that generates a corresponding set of road edge trajectories  $\mathbf{S}$ , so that the following probability of generating the real vehicle trajectories are maximized:

$$\prod_{j=1}^N p(s^{(j)}|o^{(j)}, d^{(j)}). \quad (1)$$

Here, the probability of generating each vehicle trajectory is represented as

$$p(s^{(j)}|o^{(j)}, d^{(j)}) = \prod_{t=1}^T p(e_t|e_0, \dots, e_{t-1}, d). \quad (2)$$

This problem is significantly different from the previous studies of GPS trajectory learning for the following reasons. (1) The road edge trajectory representation naturally takes the road network connectivity into consideration, so it avoids generating unrealistic trajectories that deviate from road constraints in the real world. (2) This formulation enables the trajectory generator to learn more generalized trajectory patterns according to the real road conditions. It essentially allows the generator to learn how vehicles choose their routes and adjust their speeds on the roads.

## Method

Our method designs a simulation-real combined training pipeline to achieve the following goals. (1) Mitigate the gap between generated data and real data. (2) Make sure the generated data maintain essential physical constraints of real-world trajectories (e.g., continuous in road edges, reasonable proceeding speed).

Thus, we design our method as shown in Figure 2. We propose a pipeline that our model learns the basic physical constraints from simulation data through the pre-training process, and the following fine-tuning process with real data aligns the pre-trained base model with real world driving behaviors. This combined use of real and simulated data helps address the challenges of limited data and out-of-distribution scenarios in real-world datasets. For the neural representation and learning of trajectories, we design an auto-regressive transformer-based model architecture.

### Pre-Training using Simulated Data

**Simulated Data Generation** The aim of pre-training is to explore different OD distributions as many as possible.

Hence, a domain randomization mechanism is used to first enumerate possible OD pairs from the real data, and then employ a famous microscopic traffic simulator SUMO to generate their corresponding trajectories. It is expected that simulated data will teach the pre-trained base model to grasp basic physical properties when generating trajectories, e.g., the model can select consecutive road edges in the road network and choose a relatively short path.

**Pre-training stage** We use the previously generated trajectory data to pre-train an auto-regressive transformer model represented by  $q_\theta$ , the detailed architecture of which will be introduced in later sections. For the sake of training of this auto-regressive model, each trajectory is broken down into blocks of size  $B$  with sliding stride at 1. Thus, a piece of trajectory with length  $T$  will be converted to  $T - B + 1$  blocks. To avoid notation cluttering, here we define an abbreviation  $\mathbf{e}_{t,i}^h = (e_{t-B}, \dots, e_{i-1})$ . The objective is to minimize the following cross entropy loss as the pre-train loss  $L^{\text{PT}}$ :

$$L^{\text{PT}} = \mathbb{E}_{s \sim \hat{P}} \sum_{t=B}^T \frac{1}{B} \sum_{i=t-B+1}^t -\log q_\theta(e_i|\mathbf{e}_{t,i}^h, d), \quad (3)$$

where  $\hat{P}$  represents the trajectory distribution of simulation data,  $t$  denotes the tail position of a block.

### Fine-Tuning with Real Data

In the fine-tuning stage, real data are further used to train the transformer model. In order to keep a balance between the pre-trained knowledge and the fine-tuned knowledge, a KL regularizer  $\lambda_{\text{KL}}$  is introduced to the loss function to prevent significant deviation from the pre-trained model. Thus, we have the fine-tune loss  $L^{\text{FT}}$  represented as follows:

$$L^{\text{FT}} = L^{\text{PT}} + \lambda_{\text{KL}} L^{\text{KL}}, \quad (4)$$

$$L^{\text{KL}} = \mathbb{E}_{s \sim P} \sum_{t=B}^T \frac{1}{B} \sum_{i=t-B+1}^t L_{t,i}^{\text{KL}}, \quad (5)$$

$$L_{t,i}^{\text{KL}} = D_{\text{KL}}(q_{\theta_{\text{pt}}}(\cdot|\mathbf{e}_{t,i}^h, d) \| q_{\theta_{\text{ft}}}(\cdot|\mathbf{e}_{t,i}^h, d)), \quad (6)$$

where  $P$  represents the trajectory distribution of real data,  $\theta_{\text{pt}}$  denotes the fixed pre-trained model parameter, and  $\theta_{\text{ft}}$  denotes the model parameter being fine-tuned. The dot “.” indicates a probability distribution rather than a single probability value.

### Auto-Regressive Inference

The fine-tuned model is able to generate a road edge trajectory in an auto-regressive manner. Given the origin  $o$  and destination  $d$ , we set  $e_0 = o$  as the first token of the sequence and  $d$  as a condition to the transformer-based model. We then iteratively sample the next token from the predicted distribution, appending it to the end of the sequence until the predefined end token is reached or the maximum length is exceeded.

## Model Architecture

To effectively represent the trajectory, each road edge is encoded as a token. To mimic the short time-dependency observed in real trajectories, the model is designed to consider at most previous  $B$  steps. The model takes a block of token sequence  $\mathbf{e}_t^h = (e_{t-B}, \dots, e_{t-1})$  and the destination token  $d$  as input and outputs the predicted next-token probability  $q_\theta(\cdot | \mathbf{e}_t^h, d)$ .

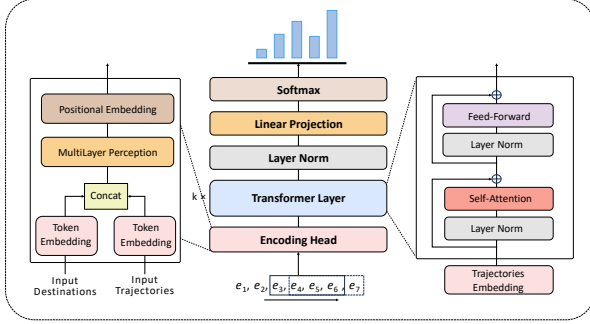


Figure 3: Overall architecture.

The whole model takes a decoder-only Transformer architecture, as Figure 3 illustrates. The Encoding Head, Transformer Backbone and Decoder Head are designed as below.

**Encoding Head** The Encoding Head processes each token in the sequence independently, combining the embedding information from the historical tokens and the destination tokens. This process can be expressed as

$$\begin{aligned} h_d &= \text{NormEmbedding}(d), \\ h &= \text{NormEmbedding}(s), \\ h_0 &= \text{MLP}(\text{Concat}([h, h_d])) + E_{pos}, \end{aligned} \quad (7)$$

where  $h_d \in \mathbb{R}^{1 \times C}$  is destination embedding,  $h \in \mathbb{R}^{B \times C}$  is historical token embedding, and  $C$  is the hidden layer size. Here,  $\text{Concat}$  concatenates the broadcasted  $h_d$  and  $h$  in the last dimension, and  $E_{pos}$  is the absolute positional embedding. The final output of the encoding head  $h_0 \in \mathbb{R}^{B \times C}$ . Specially, the normalized embedding is calculated as follows, based on empirical evidence that it stabilizes the training process.

$$\text{NormEmbedding}(\cdot) \triangleq \frac{\text{LookupTable}(\cdot)}{\|\text{LookupTable}(\cdot)\|}. \quad (8)$$

**Transformer Backbone** The Transformer backbone consists of  $L$  transformer layers, each containing a self-attention module and a feed-forward network, both wrapped with residual connections and LayerNorm. The causal mask is applied to attention weights to prevent positions from attending to subsequent positions. For the activation function, we use SiLU (Ramachandran, Zoph, and Le 2017).

For  $k = 1, 2, \dots, L$ , the process can be formulated as:

$$\begin{aligned} h_k^{\text{SA}} &= \text{SelfAttention}(\text{LN}(h_{k-1})) + h_{k-1}, \\ h_k &= \text{FFN}(\text{LN}(h_k^{\text{SA}})) + h_k^{\text{SA}}, \end{aligned} \quad (9)$$

where  $\text{LN}$  denotes LayerNorm.

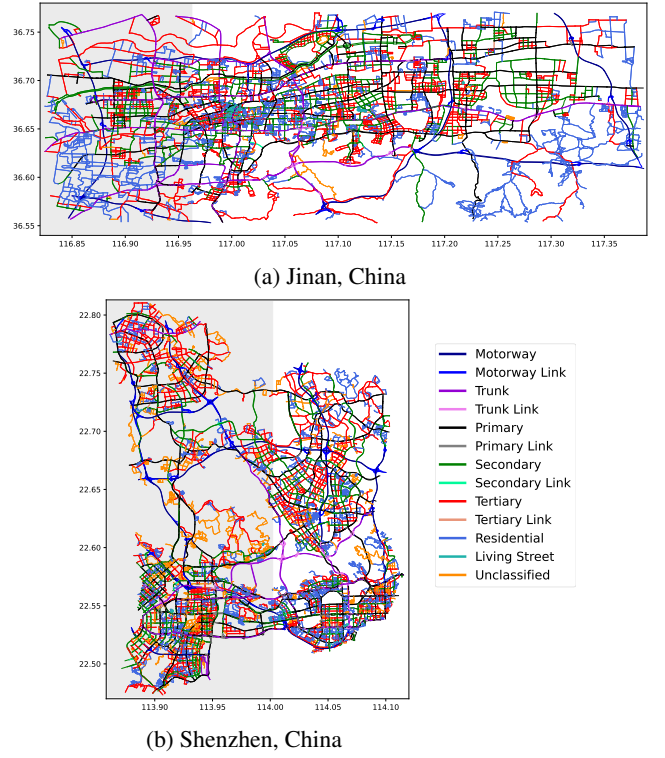


Figure 4: Visualization of road networks in Jinan and Shenzhen. The various colors represent different road tags based on the classification system in OpenStreetMap. The area indicated by the grey background is the masked region for out-of-distribution experiment setting.

**Decoder Head** After the final transformer block, we decode the sequence of hidden vectors into the prediction of the next token via a standard linear decoder. The result at each position represents the logits of the probability distribution of the token at the next position.

$$\begin{aligned} y &= \text{Linear}(\text{LN}(h_L)), \\ q_\theta(\cdot | \mathbf{e}_{t,i}^h, d) &= \text{Softmax}(y^i), \end{aligned} \quad (10)$$

where  $i = 1, 2, \dots, B$ ,  $Y \in \mathbb{R}^{B \times |E|}$  is the final output, and  $Y^i \in \mathbb{R}^{1 \times |E|}$  is the  $i$ -th row of  $Y$ . Finally, the probability of generating a trajectory is represented as the joint probability of generating each token  $q_\theta(s|o, d) = \prod_{t=1}^T p(e_t | e_{\max(t-B, 0)}, \dots, e_{t-1}, d)$ .

## Experiments

### Experiment Settings

**Data Description** The experiment is conducted on two real-world cities, Jinan and Shenzhen. Both the road network data and the city-scale vehicle trajectory data are obtained and processed from the paper (Yu et al. 2023). As described in the paper, the trajectory data is recovered from city-scale traffic camera video data through a trajectory recovery framework.

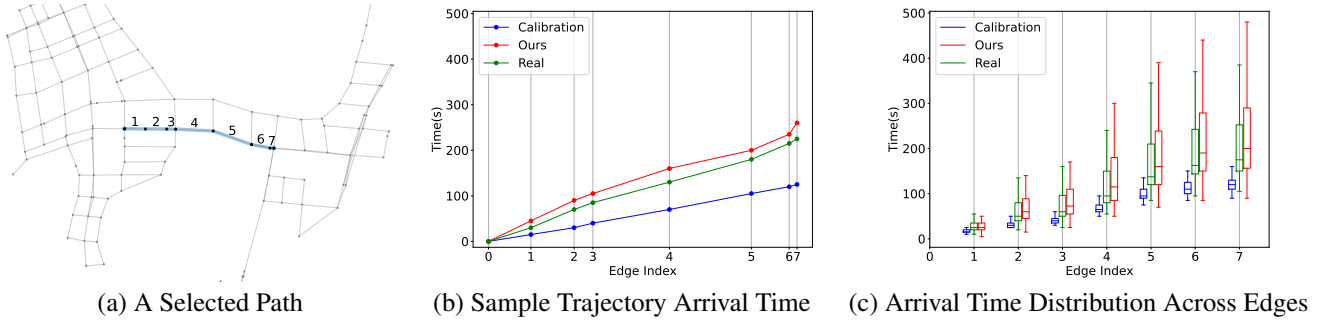


Figure 5: Comparison of trajectories generated by the calibrated simulation and our model. Figure (a) displays a selected path in Jinan. Figure (b) shows three trajectories traversing through this selected path. Y-axis is the first timestamp arriving on that road edge. We can see that the trajectory generated by calibration (in blue) moves too fast, whereas our generated trajectory (in red) presents a similar travel time as the real trajectory (in green). Figure (c) shows the statistics of all trajectories traversing through this path. In general, the travel time of our generated trajectories on this path matches better with that of the real-world observations compared with the calibration.

**Road network data.** The road network data is described by intersections and road edges, with 8,908 intersections and 23,312 road edges in Jinan, and 11,933 intersections and 27,410 road edges in Shenzhen. The road networks in these two cities are visualized in Figure 4.

**City-scale vehicle trajectory data.** The original trajectory data contains three days of vehicle trajectories in Shenzhen and one day of vehicle trajectories in Jinan. Each trajectory is represented by a sequence of intersection ID in the city road network and the corresponding timestamp. To ensure the soundness of these trajectories, the original data undergoes a pipeline process including filtering, mapping, and re-sampling, to remove the impractically long and apparently not consistent trajectories. For simplicity, we convert the timestamped trajectory into an evenly spaced sequence by setting the time interval to 5 seconds. As a result, a processed trajectory is represented as a series of edges. The output trajectory data includes 497,978 trajectories in Jinan and 774,654 trajectories in Shenzhen.

**Evaluation Metrics** The generated trajectories are compared with the groundtruth trajectories using the following metrics.

**Destination Correctness Rate (DCR):** percentage of generated trajectories that reach the intended destination.

**Jensen Shannon Divergence (JSD) of Edge Distribution:** the distance between the generated and groundtruth spatial distribution of edge visits in the road network. We calculate the frequency of each edge appearing in the generated and groundtruth trajectories, respectively, treating these frequencies as multinomial distributions, and then compute the JSD between them.

**Wasserstein Distance (WD) of Sequence Length:** distance between the sequence lengths distributions of the generated and groundtruth trajectories, reflecting differences in the overall travel time. It can be empirically approximated given two sets of samples from the compared distributions (Ramdas, Garcia, and Cuturi 2015).

**Parameters and Computational Setting** The experiment is conducted on a server with eight NVIDIA A100-SXM4-40GB GPUs. Pre-training utilizes two GPUs, while fine-tuning is performed on one GPU. The models are trained using the PyTorch framework (Ansel et al. 2024) with data parallelism to distribute the computational load across the GPUs. We use Adam as the optimizer with  $\beta_1$  as 0.9 and  $\beta_2$  as 0.999. The total training time is approximately 9 hours for pre-training and 2 hours for fine-tuning both in Jinan and Shenzhen data.

Table 1 shows our hyperparameters used in the experiments. We limit the max generated length to 500 for Jinan and to 1,500 for Shenzhen. The regularization parameter  $\lambda^{KL}$  on fine-tune term is set as 0.01 by default, while for the out-of-distribution experiment setting it is set as 100.

| Neural Net Parameters   | Value                |
|-------------------------|----------------------|
| # Layers                | 12                   |
| # Heads                 | 8                    |
| Hidden layer size       | 384                  |
| FFN expansion factor    | 2                    |
| Block size              | 16                   |
| Encoding head MLP shape | [768, 384, 384]      |
| Training Parameters     | Value                |
| Batch size              | 1,024 (pt), 512 (ft) |
| Learning rate           | 1e-4 (pt), 1e-5 (ft) |
| # Iterations            | 3e5 (pt), 5e4 (ft)   |

Table 1: Default hyperparameters of our experiments, with pt & ft represents pre-training and fine-tuning.

## Overall Results

**Comparison with Calibration** To demonstrate that the proposed method surpasses calibrating physics-based simulators in approximating real trajectories, we compare three methods: ours, SUMO simulation (without calibration),

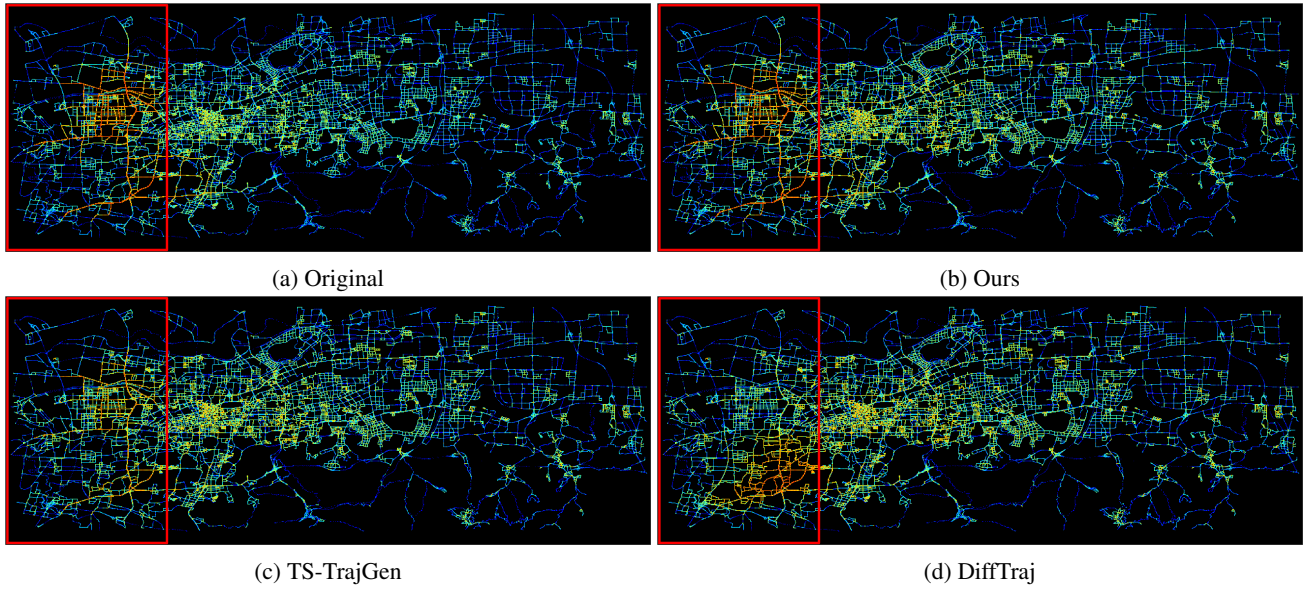


Figure 6: Density maps of trajectory data in Jinan. The red box indicates the boundary of the masked region. The varying colors represent different density levels. Our proposed model (top right) aligns more closely with the original data (top left) compared to the TS-TrajGen model (bottom left) and the DiffTraj model (bottom right).

| Method      | Jinan        |               | Shenzhen     |               |
|-------------|--------------|---------------|--------------|---------------|
|             | JSD↓         | WD↓           | JSD↓         | WD↓           |
| Simulation  | 0.052        | 24.717        | 0.185        | 182.832       |
| Calibration | 0.042        | 17.564        | 0.104        | 127.929       |
| Ours        | <b>0.023</b> | <b>11.076</b> | <b>0.020</b> | <b>86.033</b> |
| Improvement | 45.2%        | 36.9%         | 80.8%        | 32.7%         |

Table 2: Comparison with simulation methods. Simulation refers to simulator generated trajectories without calibration. Calibration refers to generated trajectories from calibrated simulators. Improvement means the relative change of our method over calibration.

SUMO simulation (with calibration), using real data as the reference. The calibration process follows these steps. (1) Speed limits for each road edge are set to the median of speed distribution from trajectory data in the training set. (2) Tabu search (Yu and Fan 2017) is used to adjust vehicle parameters (acceleration, deceleration, maxSpeed) to optimize the Wasserstein Distance (WD).

We randomly split the real trajectory data into three sets, approximately 90% for training, 5% for testing and 5% for validation. In Shenzhen, 699,200 trajectories is used for training, 37,473 for testing, 37,981 for validation. In Jinan, 448,362 trajectories is used for training, 24,800 for testing, 24,816 for validation. Simulated data from 454,518 and 732,150 trajectories is used for pre-training.

Table 2 shows the results of comparison with simulation methods. Calibration shows improvement over default simulation, which is expected because calibration uses the real data. Our method further improves over calibration, showing

| Method      | Jinan        |               | Shenzhen     |                |
|-------------|--------------|---------------|--------------|----------------|
|             | JSD↓         | WD↓           | JSD↓         | WD↓            |
| DiffTraj    | 0.670        | 55.452        | 0.664        | 265.150        |
| TS-TrajGen  | 0.116        | 56.195        | 0.229        | 258.364        |
| Ours        | <b>0.047</b> | <b>38.482</b> | <b>0.081</b> | <b>216.291</b> |
| Improvement | 59.5%        | 31.5%         | 64.6%        | 16.3%          |

Table 3: Comparison with ML methods. Improvement is the relative value of our method over TS-TrajGen.

the strength of using a neural network, which is capable of using backpropagation to fit with real data. We further use a case study in Figure 5 to intuitively illustrate why our model is better than the calibration.

**Comparison with ML Models** There is no existing work that can be directly applied in our problem setting. As discussed in the related work section, GAN-based methods generate *routes* instead of timestamped trajectories, meaning they do not care about the time that a vehicle stays on a road edge. Existing diffusion-based methods require the trajectory to be a sequence of *numeric* GPS points instead of road edges.

We compare our model with two state-of-the-art methods, representing GAN model TS-TrajGen (Jiang et al. 2023) and diffusion model DiffTraj (Zhu et al. 2024a) respectively. Significant changes have been made for DiffTraj in order to make it comparable. We convert road edges to corresponding latitude and longitude coordinates. The generated result on the coordinates needs to map back to the nearest edge. This conversion may introduce errors because a slight dif-

| Data Proportion<br>(Real:Simu) | Combine simulation data and real data |       |        |                           |       |        |                              |       |        | Real Only |       |        |
|--------------------------------|---------------------------------------|-------|--------|---------------------------|-------|--------|------------------------------|-------|--------|-----------|-------|--------|
|                                | $\lambda^{\text{KL}} = 100$           |       |        | $\lambda^{\text{KL}} = 1$ |       |        | $\lambda^{\text{KL}} = 0.01$ |       |        | DCR↑      | JSD↓  | WD↓    |
|                                | DCR↑                                  | JSD↓  | WD↓    | DCR↑                      | JSD↓  | WD↓    | DCR↑                         | JSD↓  | WD↓    |           |       |        |
| 1:100                          | 0.956                                 | 0.045 | 40.437 | 0.872                     | 0.053 | 40.159 | 0.251                        | 0.393 | 13.537 | 0.144     | 0.516 | 42.750 |
| 3:100                          | 0.957                                 | 0.045 | 40.465 | 0.895                     | 0.053 | 39.466 | 0.362                        | 0.350 | 23.648 | 0.239     | 0.425 | 59.234 |
| 10:100                         | 0.958                                 | 0.045 | 40.318 | 0.915                     | 0.053 | 38.032 | 0.555                        | 0.271 | 16.185 | 0.388     | 0.338 | 40.383 |
| 30:100                         | 0.959                                 | 0.045 | 40.190 | 0.933                     | 0.052 | 36.553 | 0.738                        | 0.176 | 5.363  | 0.600     | 0.233 | 26.190 |
| 50:100                         | 0.960                                 | 0.044 | 40.244 | 0.953                     | 0.043 | 34.663 | 0.922                        | 0.054 | 9.941  | 0.860     | 0.077 | 15.453 |
| 70:100                         | 0.961                                 | 0.044 | 40.239 | 0.957                     | 0.040 | 34.107 | 0.957                        | 0.024 | 11.854 | 0.955     | 0.028 | 14.204 |
| 100:100                        | 0.960                                 | 0.044 | 40.316 | 0.956                     | 0.041 | 33.852 | 0.959                        | 0.023 | 11.613 | 0.963     | 0.026 | 14.088 |

Table 4: Results of models fine-tuned with different real data proportions and KL regularization weights  $\lambda^{\text{KL}}$ , evaluated on real data. Regularization term  $\lambda^{\text{KL}}$  is on the pre-train model with higher indicating more weights on the pre-trained model.

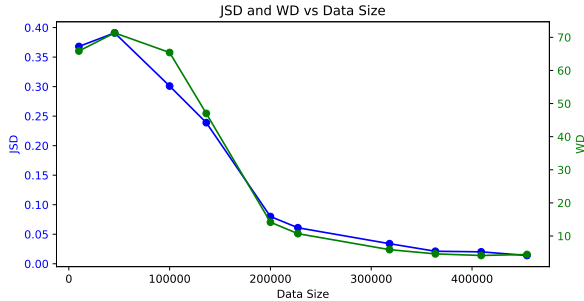


Figure 7: Ablation study on simulated data size. Performance improves with more data, leveling off after 227k samples and significantly drops below 136k samples.

ference in GPS coordinates may move the vehicle from the northbound side of a road to the southbound.

Data-driven ML models are known to suffer from over-fitting problems. To test the generalization, we mask some regions, as shown in Figure 4, and trajectories with origin or destination points within this region are all masked to mimic the real-world data missing scenario. Thus, approximately 20 percent of real data is masked in both Jinan and Shenzhen. Then, the unmasked data is used for training, and the masked data is used for testing.

Table 3 presents the performance of our proposed model alongside the two compared models. Notably, our model achieves the best JSD and WD scores in both Jinan and Shenzhen, indicating our model’s superior efficacy in generating trajectories within unknown regions compared to the TS-TrajGen and DiffTraj. Figure 6 further illustrates the differences on the map.

## Ablation Studies

**How much simulated data is necessary?** Experiment is conducted to test the model performance w.r.t. amount of simulated data in pre-training stage. The results are in Figure 7. It is observed that the model reaches better performance with more examples, while the growth slows when data size exceeds a limit (roughly 227k). However, when the dataset size is less than 136k samples (30% of the total

generated), the model’s performance significantly degrades, underscoring the need for sufficient data to ensure effective learning.

**How to balance the weight of real data and simulated data?** We adjust the KL regularizer, denoted as  $\lambda^{\text{KL}}$ , and vary the proportion of real data versus simulated data. From the results in Table 4, we observe the following patterns:

- When the proportion of real data is small, it is advantageous to set  $\lambda^{\text{KL}}$  to a higher value, thereby placing greater trust in the pre-trained data. Conversely, when sufficient real data is available, it is preferable to rely less on the pre-trained model.
- In comparison to using only real data, incorporating the pre-trained model consistently yields better results (as seen in the comparison between the “real only” column and the  $\lambda^{\text{KL}} = 0.01$  column). This further confirms the necessity of leveraging a pre-trained model.

## Conclusion

In this paper, we investigate solving the road edge trajectory learning problem via pre-training a model with simulation data and fine-tuning the model with real data. The proposed method has exhibited superior performance compared to simulation methods and existing data-driven trajectory learning methods. Ablation studies have also shown the benefit of adding simulation data as a pre-train step. This indicates that this learning model can serve as a foundation model for further trajectory learning tasks and other downstream tasks.

## References

- Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Ansel, J.; Yang, E.; He, H.; and et al. 2024. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, 929–947. New York, NY, USA: Association for Computing Machinery. ISBN 9798400703850.
- Balsebre, P.; Huang, W.; Cong, G.; and Li, Y. 2023. City Foundation Models for Learning General Purpose Representations from OpenStreetMap. *arXiv e-prints*, arXiv–2310.
- Barceló, J.; and Casas, J. 2005. Dynamic network simulation with AIMSUN. In *Simulation approaches in transportation analysis: Recent advances and challenges*, 57–98. Springer.
- Ben Ticha, H.; Absi, N.; Feillet, D.; and Quilliot, A. 2018. Vehicle routing problems with road-network information: State of the art. *Networks*, 72(3): 393–406.
- Chen, C.; Wei, H.; Xu, N.; Zheng, G.; Yang, M.; Xiong, Y.; Xu, K.; and Li, Z. 2020. Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, 3414–3421.
- Daganzo, C. F.; and Geroliminis, N. 2008. An analytical approximation for the macroscopic fundamental diagram of urban traffic. *Transportation Research Part B: Methodological*, 42(9): 771–781.
- Flötteröd, G.; Bierlaire, M.; and Nagel, K. 2011. Bayesian demand calibration for dynamic traffic simulations. *Transportation Science*, 45(4): 541–561.
- Genders, W.; and Razavi, S. 2016. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*.
- Haydari, A.; Chen, D.; Lai, Z.; and Chuah, C.-N. 2024. Mobilitygpt: Enhanced human mobility modeling with a gpt model. *arXiv preprint arXiv:2402.03264*.
- Jiang, W.; Zhao, W. X.; Wang, J.; and Jiang, J. 2023. Continuous trajectory generation based on two-stage GAN. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 4374–4382.
- Li, Z.; Xia, L.; Tang, J.; Xu, Y.; Shi, L.; Xia, L.; Yin, D.; and Huang, C. 2024. UrbanGPT: Spatio-Temporal Large Language Models. *arXiv:2403.00813*.
- Lopez, P. A.; Behrisch, M.; Bieker-Walz, L.; Erdmann, J.; Flötteröd, Y.-P.; Hilbrich, R.; Lücken, L.; Rummel, J.; Wagner, P.; and Wießner, E. 2018. Microscopic traffic simulation using sumo. In *2018 21st international conference on intelligent transportation systems (ITSC)*, 2575–2582. IEEE.
- Osorio, C.; and Punzo, V. 2019. Efficient calibration of microscopic car-following models for large-scale stochastic network simulators. *Transportation Research Part B: Methodological*, 119: 156–173.
- Peeta, S.; and Ziliaskopoulos, A. K. 2001. Foundations of dynamic traffic assignment: The past, the present and the future. *Networks and spatial economics*, 1: 233–265.
- Ramachandran, P.; Zoph, B.; and Le, Q. V. 2017. Searching for Activation Functions. *arXiv:1710.05941*.
- Ramdas, A.; Garcia, N.; and Cuturi, M. 2015. On Wasserstein Two Sample Testing and Related Families of Nonparametric Tests. *arXiv:1509.02237*.
- Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Wang, Y.; Szeto, W. Y.; Han, K.; and Friesz, T. L. 2018. Dynamic traffic assignment: A review of the methodological advances for environmentally sustainable road transportation applications. *Transportation Research Part B: Methodological*, 111: 370–394.
- Wang, Y.; Zheng, T.; Liang, Y.; Liu, S.; and Song, M. 2024. Cola: Cross-city mobility transformer for human trajectory simulation. In *Proceedings of the ACM on Web Conference 2024*, 3509–3520.
- Wei, H.; Zheng, G.; Gayah, V.; and Li, Z. 2021. Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation. *ACM SIGKDD Explorations Newsletter*, 22(2): 12–18.
- Wei, H.; Zheng, G.; Yao, H.; and Li, Z. 2018. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2496–2505.
- Yan, Y.; Wen, H.; Zhong, S.; Chen, W.; Chen, H.; Wen, Q.; Zimmermann, R.; and Liang, Y. 2024. UrbanCLIP: Learning Text-enhanced Urban Region Profiling with Contrastive Language-Image Pretraining from the Web. In *Proceedings of the ACM on Web Conference 2024*, 4006–4017.
- Yu, F.; Yan, H.; Chen, R.; Zhang, G.; Liu, Y.; Chen, M.; and Li, Y. 2023. City-scale vehicle trajectory data from traffic camera videos. *Scientific data*, 10(1): 711.
- Yu, L.; Zhang, W.; Wang, J.; and Yu, Y. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.
- Yu, M.; and Fan, W. 2017. Calibration of Microscopic Traffic Simulation Models Using Metaheuristic Algorithms. *International Journal of Transportation Science and Technology*, 6.
- Zhang, S.; Fu, D.; Liang, W.; Zhang, Z.; Yu, B.; Cai, P.; and Yao, B. 2024. Trafficgpt: Viewing, processing and interacting with traffic foundation models. *Transport Policy*, 150: 95–105.
- Zhang, Y.; Li, Y.; Zhou, X.; Kong, X.; and Luo, J. 2019. TrafficGAN: Off-deployment traffic estimation with traffic generative adversarial networks. In *2019 IEEE international conference on data mining (ICDM)*, 1474–1479. IEEE.

Zhang, Y.; Li, Y.; Zhou, X.; Kong, X.; and Luo, J. 2020. Curb-gan: Conditional urban traffic estimation through spatio-temporal generative adversarial networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 842–852.

Zheng, G.; Liu, H.; Xu, K.; and Li, Z. 2021. Objective-aware Traffic Simulation via Inverse Reinforcement Learning. In *30th International Joint Conference on Artificial Intelligence, IJCAI 2021*, 3771–3777. International Joint Conferences on Artificial Intelligence.

Zhu, Y.; Ye, Y.; Zhang, S.; Zhao, X.; and Yu, J. 2024a. Difftraj: Generating gps trajectory with diffusion probabilistic model. *Advances in Neural Information Processing Systems*, 36.

Zhu, Y.; Yu, J. J.; Zhao, X.; Liu, Q.; Ye, Y.; Chen, W.; Zhang, Z.; Wei, X.; and Liang, Y. 2024b. Controltraj: Controllable trajectory generation with topology-constrained diffusion model. *arXiv preprint arXiv:2404.15380*.

## Reproducibility Checklist

Unless specified otherwise, please answer “yes” to each question if the relevant information is described either in the paper itself or in a technical appendix with an explicit reference from the main paper. If you wish to explain an answer further, please do so in a section titled “Reproducibility Checklist” at the end of the technical appendix.

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced. yes
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results. yes
- Provides well marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper. yes

Does this paper make theoretical contributions? no

If yes, please complete the list below.

- All assumptions and restrictions are stated clearly and formally. (yes/partial/no)
- All novel claims are stated formally (e.g., in theorem statements). (yes/partial/no)
- Proofs of all novel claims are included. (yes/partial/no)
- Proof sketches or intuitions are given for complex and/or novel results. (yes/partial/no)
- Appropriate citations to theoretical tools used are given. (yes/partial/no)
- All theoretical claims are demonstrated empirically to hold. (yes/partial/no/NA)
- All experimental code used to eliminate or disprove claims is included. (yes/no/NA)

Does this paper rely on one or more datasets? yes

If yes, please complete the list below.

- A motivation is given for why the experiments are conducted on the selected datasets. yes

- All novel datasets introduced in this paper are included in a data appendix. no
- All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. yes
- All datasets drawn from the existing literature (potentially including authors’ own previously published work) are accompanied by appropriate citations. yes
- All datasets drawn from the existing literature (potentially including authors’ own previously published work) are publicly available. yes
- All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisfying. yes

Does this paper include computational experiments? yes

If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix. yes
- All source code required for conducting and analyzing the experiments is included in a code appendix. yes
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. yes
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from. yes
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. NA
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. yes
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. yes
- This paper states the number of algorithm runs used to compute each reported result. yes
- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. no
- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). no
- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper’s experiments. yes
- This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. yes

## Data Processing Details

### Tools for Data Processing

- **MaxCompute:** A highly scalable and efficient cloud-based data processing platform developed by Alibaba Cloud, used for batch processing of trajectory data in the database.
- **SUMO:** SUMO (Simulation of Urban MObility) is an open-source, highly portable traffic simulation software designed to model and analyze large-scale traffic systems. It supports a variety of simulation scenarios, including multi-modal traffic, and provides extensive tools for traffic network design, simulation, and analysis.

### Real Data Pre-processing

We design a pipeline process to batch process the original road network data and real trajectory data, which is obtained from the paper (Yu et al. 2023). The output trajectory data is then used as the input for SUMO simulation, SUMO calibration, model fine-tuning and compared methods.

- **Input Trajectory Format** Vehicle\_ID, Trip\_ID, Date, (Node\_ID, Timestamp) pairs connected in sequence. Vehicle\_ID is the vehicle identifier, Trip\_ID is the index of the trip of the vehicle, Date is the date of this trip.
- **Output Trajectory Format** Trajectory\_ID, (New\_Time, Edge\_ID) pairs connected in sequence.
- **Processing Steps**
  1. **Filter:** We use MaxCompute to filter out unrealistic trips with obvious long detour. First, the original trajectory data is loaded into MaxCompute and each trajectory is assigned a unique Trajectory\_ID. Subsequently, we eliminate trajectories that exhibit discrepancies exceeding a threshold compared with theoretical shortest path given origin and destination points of the trajectories.
  2. **Decompose:** Each trajectory is then decomposed to a sequence of Node\_ID and Timestamp pairs and transformed into multiple rows of (Trajectory\_ID, Node\_ID, Timestamp) via the intrinsic EXPLODE function.
  3. **Map:** Every Node\_ID in each trajectory is mapped to an Edge\_ID in the road network dataset. By further taking into account the timestamps, the exploded trajectory data can be converted into multiple rows of (Trajectory\_ID, Edge\_ID, Start\_Time, Time\_Interval), where Start\_Time represents the timestamp that specific vehicle entering the specific edge, and Time\_Interval denotes the total time spent on that edge.
  4. **Resample:** The previously-processed timestamped trajectory is resampled at a 5-second interval. The resampled trajectory data follows rows of (Trajectory\_ID, Edge\_ID, New\_Time), where New\_Time represents the new timestamp with 5-second interval.

5. **Connect:** The resampled trajectory data is concatenated according to Trajectory\_ID. Then, the entire trajectory dataset is divided into train, validation, and test sets in the ratio of approximate 90%, 5%, and 5%, respectively.
6. **Mask:** The mask process is employed to simulate real-world data missing scenarios. A rectangular area is selected and trajectories with origin or destination points within this region are all masked. Approximately 20 percent of total real data is masked in both Jinan and Shenzhen. Then, the unmasked data is used for training, and the masked data is used for testing.

### Simulation Data Generation

- **Roadnet Format** Edge\_ID, Start\_Node\_ID, End\_Node\_ID. The road network is represented as a graph in adjacent list, with each row as an edge.
- **Input OD Format** Trajectory\_ID, Depart\_Time, From\_edge, To\_edge
- **Output Trajectory Format** Trajectory\_ID, (Timestamp, Edge\_ID, ratio of location)
- **Processing Steps**
  1. **Generate OD and routes:** To mimic the real data distribution as much as possible, we extract the origin (first edge) and destination (last edge) from the processed real data and use SUMO to generate the route for each vehicle.
  2. **Simulation:** The road network file and the routes file are fed to SUMO to start the simulation. The simulation output is formatted in the same way as the real data, (Trajectory\_ID (Timestamp, Edge\_ID, Ratio\_of\_Location) ...), except the Ratio\_of\_Location, which is further saved to assist the training.

### Baseline Setting Description

- **TS-TrajGen:** The model is revised to enable the generation of duplicate edges, since vehicles may stay on a specific edge for multiple time steps. The hyper-parameter configuration follows the setting of the original paper, except that the training epoch is set to 10 in each training stage.
- **DiffTraj:** We make several modifications to make it fit the data. (1) Since the model takes GPS point sequences as input, each road edge of the input data is mapped to its geometric center using geometric descriptions from the dataset (Yu et al. 2023). Linear interpolation is then applied to align all sequences to a length of 40. (2) During generation, the model is conditioned on the start and end grid indices, along with the time bucket index for the departure time, while other input variables are sampled from normal distributions, consistent with their normalization in the training data. (3) After generation, the generated point sequences are mapped back to edge sequences by selecting the nearest edge to each point.

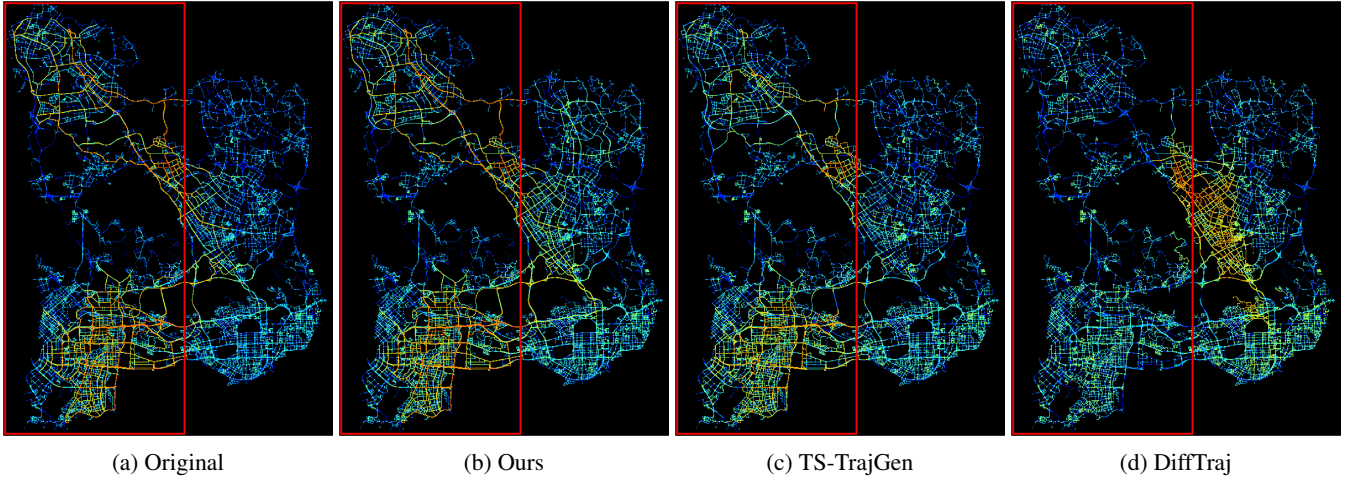


Figure 8: Density maps of trajectory data in Shenzhen. The red boundary delineates the limits of the masked region, and the varying colors denote different levels of densities. Our proposed model (b) aligns more closely with the original data (a) compared to the TS-TrajGen model (c) and the DiffTraj model (d).

### Calibration Setting Description

The goal of the calibration is to adjust the parameters of the simulation (`accel`, `decel`, `maxSpeed`), so that the generated trajectories are similar to the real ones. The train data is used to search the best parameter, and the validation set is used for choosing the parameter set.

- **Input Trip Format:** Trajectory\_ID, (New\_Time, Edge\_ID) pairs connected in sequence.
- **Processing Steps:**
  1. *Adjust the speed limit of road network:* To mimic the situation that different edges exhibit different intensity of traffic jam, we calculate the average speed for each trajectory passing through each edge and take the median as the speed limit for the specific edge.
  2. *Generate routes:* Given origin and destination (OD), use SUMO to calculate the routes for each pair of OD.
  3. *Parameters search:* This involves three steps. (1) Adjust the simulation parameters, `maxSpeed`, `acceleration`, and `deceleration` and generate 10 parameter sets near the original parameter. (2) Run SUMO simulation according to the road network and routes files and generate vehicle trajectories. (3) Evaluate the generated trajectories against the real trajectories in terms of JSD and WD scores. (4) Repeat (1) (2) (3). The parameter set with best JSD and WD results are reported.

### Extended Experiment Results

Figure 8 illustrates density maps for Shenzhen, showcasing the original masked data and the data generated by our proposed model, the TS-TrajGen model and the DiffTraj model. Similar to the density maps in Jinan, the result of our model exhibits the highest similarity to that of the original data, which further demonstrates the superiority of our model in terms of adaptability to novel scenarios.