A City-level High-performance Spatio-temporal Mobility Simulation System

Jun Zhang
zhangjun990222@gmail.com
Department of Electronic Engineering, Tsinghua University
TsingRoc, Beijing, China

Wenxuan Ao
johnao73thu@gmail.com
Department of Electronic Engineering, Tsinghua University
Beijing, China

Depeng Jin
jindp@tsinghua.edu.cn
Department of Electronic Engineering, Tsinghua University
Beijing, China

Li Liu
lliu@birentech.com
BirenTech Research
Shanghai, China

Yong Li*
liyong07@tsinghua.edu.cn
Department of Electronic Engineering, Tsinghua University
Beijing, China

ABSTRACT

Urban mobility simulation refers to simulating human fine-grained spatio-temporal mobility and activity behaviors in cities, which facilitates the measurement of traffic operations, assesses the impacts of transportation on other areas of the city like environment, and supports the designation of simulation-driven mobility-related sustainable policies. It is becoming one of the important tools in the development of sustainable cities. However, the main challenge currently restricting the mobility simulation and its applications is poor performance when dealing with city-scale million or even ten million people simulation. In mobility simulation, agents like people and vehicles need to take actions based on the previous step's states of other agents nearby, which reflects the spatio-temporal dependencies among agents. Facing city-scale scenarios, the spatio-temporal dependencies become the main barrier to achieving efficient parallel computation acceleration. To alleviate the impact of spatio-temporal dependencies on parallel acceleration, we propose a city-level high-performance spatio-temporal mobility simulation system designed with a two-stage parallel process based on read/write separation and a parallel-friendly indexing subsystem. The two-stage parallel process optimizes cross-step state read/write processes among agents by reorganizing all states into three categories (public read-only, public write-only, and private) and introducing a two-stage control flow design to divide the entire data flow into two easily parallelized groups. The indexing subsystem optimizes spatial belonging relationship maintenance and relative location queries through parallel-friendly data structure selection with addition, deletion, and query process design. We implement the whole system on both CPU and GPU to adapt to different hardware environments. We also conduct extensive experiments and build use cases to demonstrate that the system achieves the expected results in terms of performance and can support innovative applications about sustainable mobility research. The experiments show that our proposed system achieves a computational speedup of 278.77 times the wall clock time, i.e. 3.59 milliseconds per step, in a city-level simulation with nearly 1 million people simultaneously.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**; • **Computer systems organization → Parallel architectures**; **Multi-core architectures**.

KEYWORDS

Mobility simulation, parallel system, heterogeneous acceleration system

ACM Reference Format:
Jun Zhang, Wenxuan Ao, Depeng Jin, Li Liu, and Yong Li. 2023. A City-level High-performance Spatio-temporal Mobility Simulation System. In *1st ACM SIGSPATIAL International Workshop on Sustainable Mobility (SuMob '23), November 13, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3615899.3627936

1 INTRODUCTION

Urban mobility simulation, as computational tools to simulate fine-grained spatio-temporal patterns and dynamics of human mobility and activity behaviors within urban space, can help researchers and urban decision-makers to measure traffic operations [1, 13], evaluate the impacts on other areas like environment [4], public health [12], planning [9], etc. It can also support sustainable mobility policy making [17].

From a technical point of view, mobility simulation is generally implemented as the process of calculating the fine-grained change of human's spatio-temporal positions through step-by-step iteration based on the kinematic physical laws and the mobility simulation models like driving and walking models. Fine-grained refers to the time granularity of less than or equal to 1 second.

Nowadays, the main challenge of large-scale city-level mobility simulations is poor computational performance. SUMO [2], as the most popular open-source traffic simulation system, has mature

*Yong Li is the Corresponding Author.

Permission to make digital or hard copies of part or all 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 third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SuMob '23, November 13, 2023, Hamburg, Germany
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0361-4/23/11.
https://doi.org/10.1145/3615899.3627936

map processing[1] and traffic demand generation[2]. However, it can only calculate one iteration of about 80,000 people per second according to its documentation[3]. This means that if faced with a city-level scenario where 800,000 people need to move at the same time, the simulation would take 10 times longer than the wall clock time. The calculation speed is unacceptable. In addition to the real-time simulation application mentioned above, the performance issues severely restrict the applications of mobility simulation in decision-making tasks. Because these tasks usually require multiple simulations to find the optimal solution, researchers have to waste a lot of time waiting for the calculation. Cityflow [25] implements multi-thread acceleration and achieves significant performance improvements compared to SUMO, but it only completes the driving mode, which cannot fully simulate urban mobility. Therefore, we need to achieve parallel acceleration on the main mobility modes so that the tool can help researchers find city-scale solutions for sustainable mobility.

To achieve parallel computing for mobility simulation, we have to properly handle the **the spatio-temporal dependencies** among agents like people and vehicles in system implementation. The spatio-temporal dependencies include the time dependency of the agent reading the state of the previous step in each iteration and the spatial dependencies of obtaining other agents' states which are spatially close. For example, when a person is driving, he/she needs to care about his/her speed and the speed and distance of the vehicle ahead to take the next driving action. Due to the existence of these dependencies, we cannot simply divide computing tasks into unrelated batches, and then use mature parallel computing frameworks such as MapReduce [8] and Spark [23] to accelerate it. Instead, we need to design data flow and control flow for efficient parallelization to handle the time dependency and provide parallel-friendly ways of maintaining and querying these spatial relationships to handle spatial dependencies.

Therefore, we propose a high-performance spatio-temporal mobility simulation system. Due to the differences in patterns of human mobility behavior in different kinds of spaces, the system first divides the whole city space into two parts, *indoor* and *outdoor. Indoor* is the collection of area-of-interest (AOI), representing relatively enclosed areas, such as schools, shopping malls, parks, neighborhoods, etc. *Outdoor* is the collection of the urban road network, including driveways, sidewalks, crosswalks, etc. Human mobility in outdoor spaces is usually along the direction of the road, while in indoor spaces it is mainly limited by the boundaries of the area. According to the division of city space, mobility models are also divided into two categories: indoor (e.g. walking in AOI) and outdoor (e.g. driving, biking, walking). With the above city space division, the system provides universal mechanisms to alleviate the computational parallelization restrictions from the spatio-temporal dependencies. These universal mechanisms include a two-stage parallel process based on read/write separation for solving the time dependency and an indexing subsystem for efficiently handling the spatial dependencies. To verify the performance of the above design, we implement the system with multiple mobility models on both CPU and GPU and conduct corresponding experiments.

[1]https://sumo.dlr.de/docs/#network_building
[2]https://sumo.dlr.de/docs/#demand_modelling
[3]https://sumo.dlr.de/docs/FAQ.html#how_fast_can_sumo_run

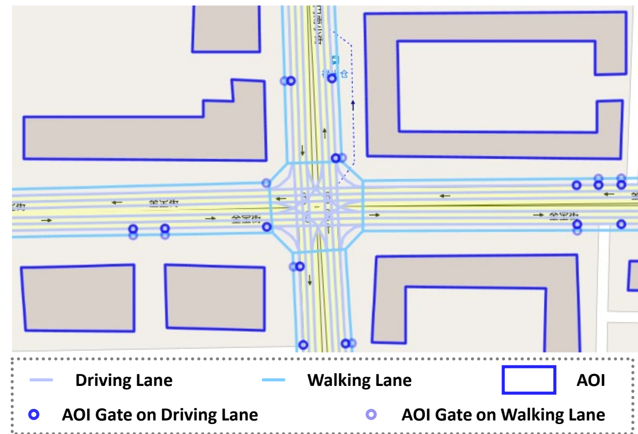

Figure 1: An example of the city space division in our proposed system.

To summarize, the main contributions of the work are as follows:

- We design two universal mechanisms, a two-stage parallel process based on read/write separation and an indexing subsystem, to solve the spatio-temporal dependencies challenge which restrains the parallel acceleration of large-scale city-level mobility simulation.
- We implement the city-level high-performance spatio-temporal mobility simulation system on both CPU and GPU, conduct experiments to verify the system's performance and build application cases to show its usability. The experiments show that in a city that simulates 900,000 running agents simultaneously in a large city, the system achieves a computational speed of 3.59 milliseconds per step on GPU, which is 278.77 times the wall clock time.

The remainder of the paper is organized as follows. We first introduce the preliminaries in Section 2. In Section 3 we elaborate on the spatio-temporal dependencies in mobility simulation, and the technical challenges they pose. The design and implementation of the proposed system are detailed in Section 4. Afterward, we conduct experiments and show several application cases in Section 5. We review related work in Section 6 and conclude the paper in Section 7.

2 PRELIMINARIES

2.1 City Space Division

Due to the differences in the inherent structure of urban space, people have different mobility behavior patterns in different types of spaces. For example, when people walk on a sidewalk, they usually move in the direction of the road, while when they move around a square, they are limited only by the square's boundaries. Therefore, we divide the whole city space into *indoor* and *outdoor* parts in the system to match these two common human mobility behavior patterns. *Indoor* space mainly corresponds to human mobility behavior inside buildings or enclosed areas. While the *outdoor* space mainly corresponds to the human mobility behavior in the urban road network.

As shown in Figure 1, the *indoor* space is modeled as a collection of AOIs. AOI represents relatively enclosed areas like schools, shopping malls, neighborhoods, etc. In the system, the AOI is recorded as a $(id, polygon)$ tuple, where *polygon* is a list of coordinate points (x, y) to represent the boundary of the AOI. Modeling *outdoor* spaces is more complex. To provide a common abstraction, modeling of the outdoors needs to include at least driveways and sidewalks, and express the topological relationships between driveways and driveways, and between sidewalks and sidewalks. Also, crosswalks should be considered. For the sake of uniformity, we refer to the driveways, sidewalks, and crosswalks collectively as lanes. All lanes are divided into two categories, driving lanes and walking lanes, which are also shown in Figure 1. Obviously, driving can only happen on the driving lane, while walking and biking can only happen on the walking lane. The lane in the system is recorded as a $(id, type, polyline, predecessors, successors, left, right)$ tuple, where *type* is driving or walking, *polyline* denotes the center line of the lane and is recorded as a list of coordinate points (x, y), *predecessors* is a list of IDs for the current lane's preceding lane, *successors* is the lane ID list following the current lane, $left$ and *right* are the IDs of the adjacent lanes to the left and right of the current lane respectively. Such a data model reflects the spatial location and topological relationships of the lanes.

To link the *indoor* space with the *outdoor* space, the AOI has a series of "gates" that indicate where to enter and exit the AOI. The gates are recorded as a list of $(lane_id, s, type)$ tuple in the AOI's data structure, where $lane_id$, s and $type$ denote the lane ID where the gate is located, the distance of the gate location relative to the beginning of the lane at the lane's polyline and the type of the lane, respectively.

To construct the above city space data, we implement a tool chain for processing open-source map data from OpenStreetMap[4].

2.2 Input for Mobility Simulation

The input for mobility simulation is uniformly modeled as a list of trips for each person over time. A trip is defined as a $(P_s, P_e, t_s, mode)$ tuple, where P_s and P_e are respectively the starting and ending positions of the trip, t_s is the starting time of the trip, and *mode* is the trip's mobility mode like walking, driving and biking. The position P can be either an AOI or a position at a lane. Through the input form, our system is compatible with mobility simulation demand inputs of different spatio-temporal scales. For example, we can sample trips from the results of origin-destination (OD) predictions [7, 21] to restore the fine-grained mobility of the city's overall population. We can also use deep learning methods to directly generate individual position-based activity trajectories [6, 12, 22], which may be beneficial for city managers to focus on urban dynamics under the influence of specific events. In addition, rule-based generation [3] is a more classic way to provide individual position-based activity trajectories.

Driven by such input data, the task of mobility simulation is to output fine-grained mobility trajectories in a step-by-step calculation.

[4]https://www.openstreetmap.org/

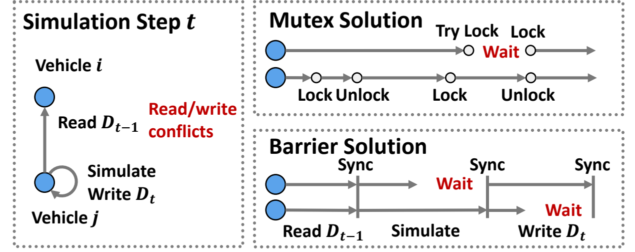

Figure 2: An example of the time dependency and the corresponding mutex and barrier solution.

2.3 Basic Process of Mobility Simulation

The basic process of mobility simulation is a discrete-time iterative calculation process. There is a global clock that increments by a fixed amount of time Δ_t (the default is 1 second) at each simulation step. At each step, all agents are updated based on the mobility models corresponding to their current mobility behavior. The update process can be divided into two parts: one is kinematic physical laws, and the other is acceleration decision-making. The kinematic physical laws are as follows:

$$P_t = P_{t-1} + v_{t-1}\Delta_t + \frac{a_t \Delta_t^2}{2}, v_t = v_{t-1} + a_t \Delta_t, \quad (1)$$

where P_t, v_t, a_t are the position, the velocity, and the acceleration at step t. P, v and a are represented as 1D scalars in an outdoor space and as 2D vectors in an indoor space. The acceleration decision-making, i.e., how to determine the value of a at each step, depends entirely on the choice of the mobility model. Generally speaking, the determination of a depends on the self state and the states of other agents around, such as the position and velocity. Thus, we can formulate it as follows:

$$a_t = f\left((P_{t-1,self}, v_{t-1,self}), \left\{(P_{t-1,i}, v_{t-1,i}) | i \in \mathcal{N}_{self}\right\}\right), \quad (2)$$

where $\mathcal{N}_{self}$ is the set of agents that are close to the agent $self$. The function f is the model of the agent's mobility behavior, which can be a simple collision-free model with fixed speed for walking, a complex model to consider relative distance and velocity for driving like [16, 19], etc.

3 SPATIO-TEMPORAL DEPENDENCIES IN MOBILITY SIMULATION

The section gives examples to explain the spatio-temporal dependencies among the states of agents at different steps in mobility simulation shown in Equation 2 and analyze the key issues that need to be addressed in the design and implementation of parallel systems due to the spatio-temporal dependencies.

3.1 Time Dependency

Equation 1 and 2 show that the calculation of an agent depends on its state and other agents' states at the previous step, which leads to a time dependency.

The example, shown in Figure 2, are now two vehicles i and j in the system, and the state of vehicle j at step t is denoted as D_t. From the perspective of vehicle j, it performs the simulation and modifies

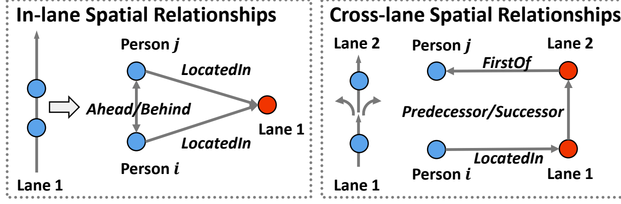

Figure 3: Examples of the spatial relationships in the *outdoor* space.

its state to the new value D_t, while vehicle i needs to obtain its data D_{t-1} for the simulation process. This leads to read/write conflicts for the state data of vehicle j. The most intuitive solution to the problem of read/write conflicts in parallel systems is the mutual exclusion lock (a.k.a mutex). The mutex ensures that a piece of data can only be manipulated by one task by providing two primitives: `lock` and `unlock`. If a task locks the mutex, other tasks trying to lock it must wait until the mutex is unlocked by the task. In the mutex solution, since multiple tasks may wish to process the same piece of data at the same time, waiting occurs, which leads to a waste of computational resources for this period and introduces an implicit sequential process. Moreover, the solution does not guarantee that vehicle i always obtains the state D_{t-1} of vehicle j at the previous step, which is unavailable.

Another way that can guarantee correctness is a barrier-based solution, which is implemented in [25]. Barrier synchronization requires all tasks to wait at the barrier until all the tasks reach the barrier. In this way, all vehicles can perform data reads simultaneously first, then simulations and data write simultaneously in sequence. However, an unbalanced amount of computation can result in more waiting time for tasks that end earlier. Also, as a multitasking synchronization mechanism, using the barrier itself introduces some implicit sequential processes.

According to Amdahl's law, the speedup ratio of task parallel processing can be expressed as follows:

$$S = \frac{1}{1 - a + \frac{a}{n}}, \tag{3}$$

where S denotes the speedup ratio, a denotes the ratio of parallelizable parts, n is the number of parallelism. The law shows that even if infinite computational resources are invested to parallelize the simulation tasks, i.e., n tends to infinity, the overall speedup ratio will not exceed $1/(1 - a)$. Therefore, mutex and barrier not only cause a waste of computational resources but also constrain the upper limit of the system speedup.

The key to solving the issue lies in how to reasonably design the data flow and control flow of the system according to the characteristics of mobility simulation, to minimize mutex, barrier, and other sequential processes, thus increasing the upper limit of the system acceleration ratio.

3.2 Spatial Dependencies

The spatial dependencies in the mobility simulation are reflected as spatial belonging relationship maintenance and relative location queries.

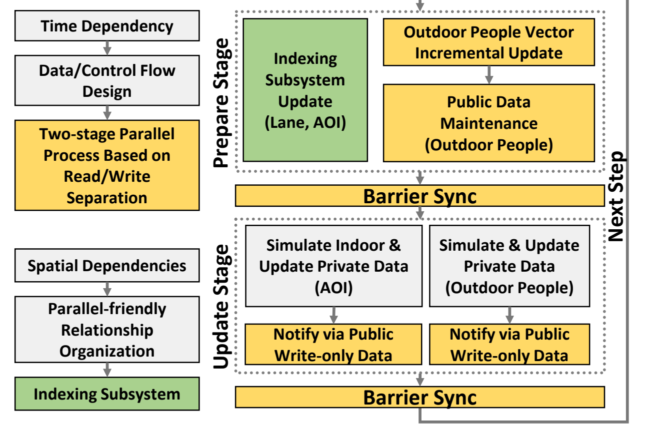

Figure 4: The system framework with *prepare* stage and *update* stage parallel process based on read/write separation and indexing subsystem to solve the key issues from spatio-temporal dependencies in the mobility simulation.

Taking the *outdoor* space as an example, Figure 3 shows two simple spatial dependency phenomena with corresponding spatial relationships. In both cases, we assume that person i needs to identify the person in front of him/her, which is a common spatial query process used by driving mobility models. If both people are in the same lane, the relative position relationship of the two people (*ahead/behind*) and the spatial belonging relationship (*located in*) are included here. If two people are not in the same lane, person i first needs to confirm the lane he/she is *located in*, then get the next lane based on the lane *predecessor/successor* relationship and finally find the *first* person *of* lane 2.

As for *indoor* spaces, the spatial belonging relationship is also necessary. Moreover, people within the AOI have different ways of using the space, such as walking and waiting to leave. Walking people treat AOI as a 2-dimensional flat space. The waiting person is standing still at the gate waiting for the departure condition to be met. As the simulation progresses, people in the AOI will change the way they use their space.

Overall, the most critical issue in the face of spatial dependencies is to find parallel-friendly data structure and their operation primitives to maintain these relationships and provide efficient queries.

4 SYSTEM DESIGN AND IMPLEMENTATION

4.1 Overview

We design and implement the system, whose framework is shown in Figure 4. Specifically, first, we introduce the data flow and control flow design that minimizes the use of mutex, barrier, and other sequential processes to alleviate the impact of the time dependency. The design is called two-stage parallel process based on read/write separation. The read/write separation data flow design organizes data into three types: *private data*, *public read-only data*, and *public write-only data*. By read/write separation, a large number of potentially mutually exclusive accesses and mutex-related system calls are avoided in the system. Read/write separation also requires

that each simulation step be divided into the *prepare* stage and the *update* stage to maintain and update different types of data separately. In the system, we design an indexing subsystem with parallel-friendly data structure to organize spatial relationships and provide spatial queries with very low time complexity. The indexing subsystem includes lane location relationship linked lists and AOI multi-state index. The indexing subsystem also follows the two-stage parallel process based on read/write separation for data maintenance, receiving notifications in the *update* stage through *public write-only data* interface and maintaining *public read-only data* in the *prepare* stage by incremental update to ensure its efficiency. The details of the above design and implementation are shown in Figure 5.

4.2 Two-stage Parallel Process Based on Read/Write Separation

Due to the time dependency in the mobility simulation, there are complex time-related data operations with read/write conflicts in the system, including reading the state of other agents at the previous step and writing the state of itself at the current step, etc. Faced with the situation, the common solution of mutex and barrier in parallel systems introduces waiting times and implicit sequential processes that degrade system performance. To minimize the use of mutex and barrier and provide the correct functionality, we apply a read/write separation strategy.

In each simulation step, the agent's data is divided into three parts: *public read-only data*, *public write-only data*, and *private data*. *Public read-only data* stores some information about the agent that is accessible to others, such as the person's position, speed, etc. *Public write-only data* is used to store modifications and notifications from other agents, such as people informing the lane that they have left or entered, etc. *Private data* contains everything that supports the agent's simulation calculation and is only maintained and updated by itself. Obviously, reading *public read-only data* and using *private data* do not require mutually exclusive protection, while *public write-only data* does. Therefore, it is worth noting that the system needs to be designed in such a way that the *public write-only data* always receives only a small number of writes in one simulation step, to avoid the waiting process of mutex as much as possible.

To solve the problem of maintaining different types of data, each step of the simulation process is divided into two stages: *prepare* stage and *update* stage. Both stages are finished with a barrier to ensure synchronization so that the number of barriers in the whole system is limited to two. In the *prepare* stage, each agent handles the copying and processing of data from *public write-only data* to *private data*, completes the resetting of *public write-only data*, and copies the corresponding data from *private data* to *public read-only data*, which is shown in Figure 5(d). Also, as shown in Figure 5(a) and Figure 5(b), the indexing subsystem mentioned below updates the indexes based on the change notifications in the *public write-only data* in the *prepare* stage, thus ensuring that they are ready during the *update* phase. In the *update* stage, each agent obtains the required information from the *public read-only data* of other agents by the indexing subsystem, computes according to the corresponding logic like mobility models or state change rules, updates the *private data* and performs notification using the *public write-only data* interface of the relevant agent, which is shown in Figure 5(e) and Figure 5(f).

Throughout the two-stage parallel process, there is an unavoidable explicit sequential process, which is the maintenance of the outdoor people vector. It exists because we use AOI as the parallel unit in *outdoor* space and use people as the parallel unit in *outdoor* space considering the differences and characteristics of the simulation models. Specifically, indoor people mostly do not consume calculations due to the resting state, which is mentioned in the following AOI multi-state index part, while outdoor people mostly have relatively complex mobility models. Since people will switch between the *indoor* space and the *outdoor* space during the simulation, the outdoor people vector should be updated. To alleviate the impact of this explicit serial process on overall performance, an incremental update mechanism is introduced. As shown in Figure 5(c), the incremental update mechanism consists of a join vector and a departure vector, both of which are protected by the corresponding mutex. In the *update* stage, if a person enters the *outdoor* space from the *indoor* space or leaves the *outdoor* space to enter the *indoor* space, the person needs to add its pointer to the corresponding vector. During the *prepare* stage, incremental updates based on the two vectors will be executed. We first pair the additions and deletions in the join and departure vectors and complete in-place replacement by the offsets recorded in the private data of people who will be deleted. Secondly, we append the remaining additions directly to the end of the vector and remove the remaining deletion requirements by swapping them with the last element of the vector and modifying the offset of each data accordingly. The average time complexity of the above operation is $O(m+n)$, where m and n are the average lengths of the join and departure vectors, respectively. Since the time granularity of the system is less than or equal to 1 second, the number of people added and deleted at each step is relatively small, so the impact of this sequential process on the overall performance is also small.

In summary, as shown in Figure 5, the workflow in the *prepare* stage includes:

(1) Incremental update of the *outdoor* people vector.
(2) Parallel incremental update lane location relationship linked lists with new joins and departures.
(3) Parallel update AOI multi-state index with new joins.
(4) Parallel update outdoor people's *public read-only data* and maintain their *public write-only data* if needed.

The workflow in the *update* stage is as follows:

(1) Parallel processing of simulations within each AOI, including walks within the AOI, state changes of people with the necessary calculations.
(2) Parallel execution of the simulation process for outdoor people, including data fetching by index subsystem, simulation calculation, private data updating, notification of the belonging space changes to the corresponding lane, AOI, and outdoor people vector mechanism if necessary.

4.3 Indexing Subsystem

According to the division of city space, the indexing subsystem contains lane location relationship linked lists and AOI multi-state index.

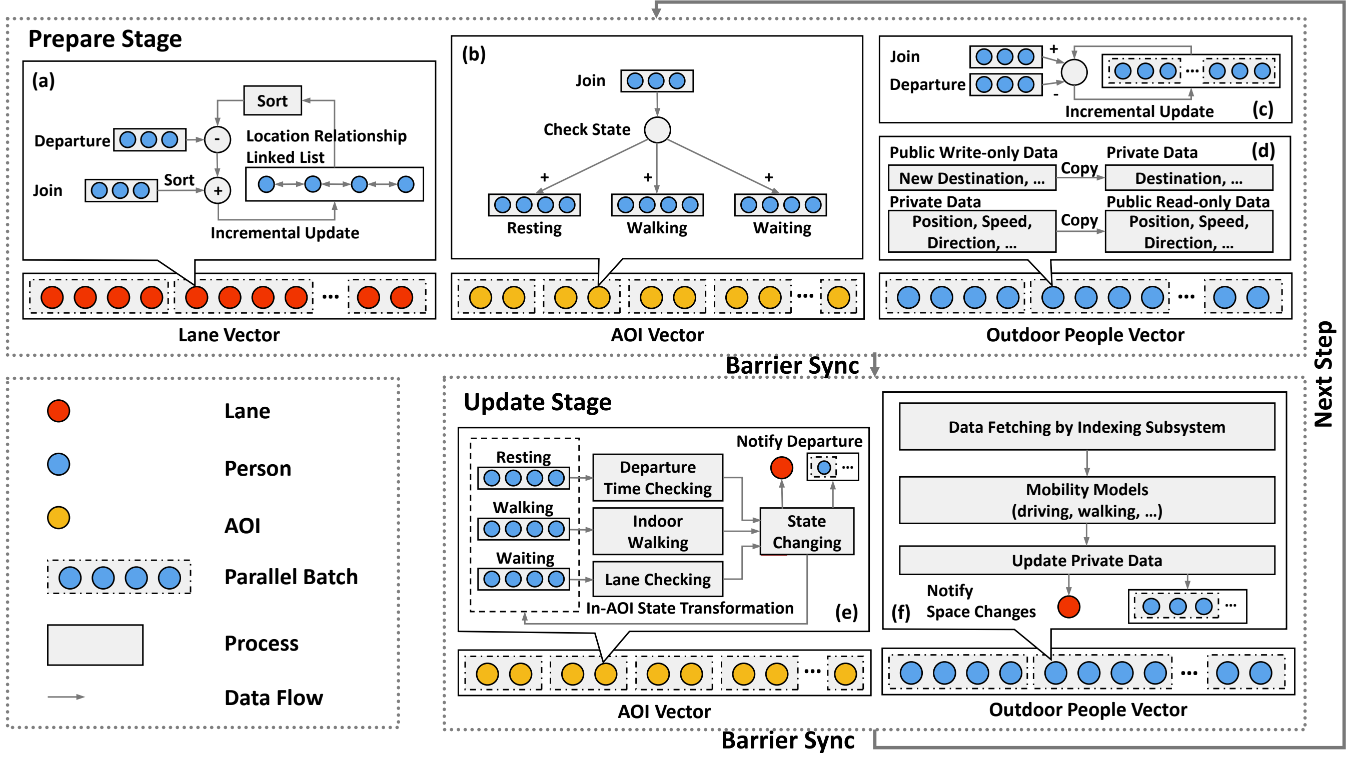

Figure 5: The system design and implementation details.

4.3.1 Lane Location Relationship Linked Lists. Lane location relationship linked lists maintain two spatial relationships, including the belonging relationship between people and lanes, and the relative position relationship between people and people on the same lane.

As shown in Figure 5(a), lane location relationship linked lists are ordered bi-directional linked lists. The nodes of the linked lists are people. The linked lists are arranged in ascending order of the distance from the person following the lane's polyline to the beginning of the lane. Based on such data structure, a person can easily find the people before and after him/her through the next pointers of the ordered bi-directional linked lists. Also, the first and last person in the lane can be obtained from the head and tail pointers of the linked lists. The time complexity of all the above operations is $O(1)$.

Similar to the outdoor people vector, the linked lists also use incremental updates, containing a join vector and a departure vector as each lane's *public write-only data*. The difference is that due to the characteristics of the data structure of the linked list, in the incremental update, we first perform all the deletion operations, secondly reorder the original linked list based on the new location of the vehicles, and finally merge the two linked lists by merge sort after constructing the joined vehicles into an ordered linked list by location. Since the outdoor mobility simulation process does not involve the rear people overtaking the front people in the same lane at most times, the reordering operation is usually only a linear time complexity check. Therefore the average time complexity of an incremental update process is rough $O(l + mlog(m) + n)$, where l, m, and n are the average lengths of the linked list, the join and departure vectors, respectively. And the number of added and deleted people is limited, i.e., m and n are small, so the computational overhead of index maintenance is acceptable.

4.3.2 AOI Multi-state Index. AOI multi-state index records the people who are located in the AOI and the data structures are designed for different people states.

People in AOI have multiple states, which usually include *resting*, *walking*, and *waiting*. *Resting* state means that the person has no mobility needs due to not reaching the departure time. *Walking* state means that the person is walking indoor. *Waiting* state means that the person is unable to leave the AOI due to congestion on the lane corresponding to the AOI gate so he/she stands still and waits. According to the meaning of each state, it can be seen that the only agents to be calculated in each step of the simulation include only the people who arrive at the departure time in *resting* state, the people whose corresponding lanes are free in *waiting* state, and all the people in *walking* state. Based on the above analysis, all people in *resting* state are organized into a priority queue in ascending order of departure time. And those in *waiting* state are stored in a first-in-first-out (FIFO) queue. Those in *walking* state are organized into a common vector. All of the above data structures are implemented through arrays due to the efficient processing power of modern CPUs and GPUs for small arrays.

The deletion operations for people in AOI are done in the *update* stage itself. The join operations are stored in a *public write-only* join vector by notifications from people in the *update* stage. All the joined people will be assigned to the data structure of the

Table 1: Empirically optimal parallel batch size for agents in *prepare* stage and *update* stage.

Agent type	*Prepare* stage	*Update* stage
AOI	512	512
People	2048	1024
Lane	2048	-

corresponding state in the *prepare* stage as shown in Figure 5(b). With AOI multi-state index, it is no longer necessary to waste computing power in AOI to traverse every person, but only to deal with the agents that have actual updates.

In conclusion, through the agent state division and two-stage grouping of calculation, we compress the proportion of the sequential process in the entire simulation, improve the system parallelism, and reduce the impact of time dependence on parallel acceleration. The index subsystem provides efficient spatial relationship queries to address the spatial dependencies and improves simulation efficiency without adding too many sequential processes.

4.4 Mobility Modes

For the driving mode, we use the classical Krauss [16] model following SUMO to implement car-following. The model controls the vehicle to drive at a safe maximum speed by calculating the relative distance and the relative speed between the vehicle and the preceding vehicle, and the braking acceleration of both vehicles, which can be obtained from the indexing subsystem at constant time complexity. Further to this, we also refined the impact of lane speed limits, signals, distance to the end of the line, and other factors on vehicle control. The model for outdoor walking and biking is relatively simple, and we use a one-dimensional collision-free model with fixed speed. Different fixed speed values are used for walking and biking, and the speed parameters for different people are randomly disturbed to make them look more reasonable. The routes of all outdoor mobility behaviors are precomputed by the A* algorithm with time as the weight.

For the indoor walking model, indoor walking is modeled as a two-dimensional movement within an AOI polygon considering pedestrian avoidance. We use the method proposed in [24], which is one of the latest work to improve the classic social force model [14]. The method first uses physics-infused neural networks (PINN) to learn crowd mobility patterns in 2-dimensional space from a dataset, and then extracts the formulas and parameters from the neural network model, thus enabling the model to be simply implemented into the system.

4.5 Implementation Details

Due to the strong universality of the above design, we implemented both the CPU version and the GPU version of the system to meet different hardware cost considerations. The CPU version is implemented by Golang 1.19[5], while the GPU version is implemented by C++/CUDA.

On the CPU version, all parallel processes are implemented via Goroutine in the Golang programming language. Goroutine is a lightweight user-state thread managed by the Go runtime.[6] The scheduler in the Go runtime manages all Goroutines and preemptively schedules them to run on the OS threads created by the Go runtime. The approach avoids the huge overhead associated with the frequent creation and deletion of OS threads. The preemptive scheduling also allows for a more balanced amount of computation to be carried on different CPU cores and allows Goroutines blocked by synchronization mechanisms such as mutex and barrier to be hung to avoid wasting hardware computational resources. It is obvious that Goroutine is also created with some overhead. To avoid creating too many Goroutines and increasing the processing overhead of the Goroutine scheduler, each Goroutine is responsible for processing a batch of the agents as shown in Figure 5. For the two stages, the empirically optimal batch size take values for each type of agent are given in Table 1.

On the GPU version, all parallel processes are implemented as CUDA kernel functions. And we always use one GPU thread to process one agent. The block size and grid size of each kernel function are computed by the helper function provided in the CUDA library. Fair scheduling of kernel functions is done by NVIDIA GPU drivers and hardware.

In terms of hardware, we use Aliyun[7] ecs.g7.16xlarge instance with Ubuntu 20.04 following Mirage [26] for the CPU version of the system. The instance provides 64 virtual cores with 256GB of memory, whose virtual cores are Intel Xeon(Ice Lake) Platinum 8369B. We use Aliyun ecs.gn7e-c16g1.4xlarge instance with Ubuntu 20.04 for the GPU version of the system. The instance provides one NVIDIA A100 80G GPU, and 16 virtual cores with 125GB of memory, whose virtual cores are also Intel Xeon(Ice Lake) Platinum 8369B.

Table 2: Statistics of the city space datasets.

Dataset	Grid-40	Grid-80	Beijing
#Driving lane	130056	670656	528742
#Walking lane	0	0	228720
#AOI	0	0	136619

5 EXPERIMENTS

5.1 Experimental Setup

In this section, we carry out experiments to answer several research questions:

- **RQ1:** What is the overall performance of the system compared to other mobility simulations such as SUMO and Cityflow?
- **RQ2:** What is the relationship between the computational performance of the system and the number of agents?
- **RQ3:** How much computing resources are consumed by each functional module in the system, such as the indexing subsystem and the people data preparation?

5.1.1 City Space Datasets. To facilitate comparison of the performance of mobility simulations such as SUMO [2] and Cityflow [25], we use SUMO's netgenerate tool to generate grid-like road networks as the city space datasets, and randomly generate trips on it.

[5]https://go.dev/

[6]https://go.dev/tour/concurrency/1

[7]https://cn.aliyun.com/

The lane number of each road is set to 3. We generated two road networks with grid numbers of 40×40 and 80×80, named Grid-40 and Grid-80 respectively.

We also build a Beijing city space dataset from OpenStreetMap through our toolchain. However, since we have not implemented the conversion tool from our data format to the SUMO format, the dataset will not be used for performance testing between different mobility simulations.

The three datasets' statistics are shown in Table 2.

5.1.2 Baseline Methods. We compare the CPU version and the GPU version of our system to the two existing popular mobility simulations as baselines, SUMO [2] and Cityflow [25]. To easily distinguish the hardware resources used by different methods, we add the hardware resource identifier used after the system name. Specifically, SUMO-1 represents the single-threaded version of SUMO, and SUMO-64 represents the SUMO program executed with 64 threads. Cityflow-64 represents the Cityflow program executed with 64 threads. Besides, Ours-CPU-64 represents the CPU version of our system executed with 64 threads, and Ours-GPU represents the GPU version of our system.

5.1.3 Experiment Settings. To answer the first research question, we chose two scenarios to compare the time cost of various methods. The first scenario is when the number of running agents reaches 100,000 on the Grid-40 dataset, which is named as Grid-40-100k. The second scenario is when the number of running agents reaches 200,000 on the Grid-80 dataset, which is named as Grid-80-200k. Since Cityflow does not support walking models, the agents in the two scenarios only include vehicles on the road. The time cost is calculated as the average simulation time of the 10 steps in which the current number of running agents is closest to the target value.

For the second research question, we test the CPU version and the GPU version of our system for 100,000, 300,000, 500,000, 700,000, and 900,000 running agents simultaneously on the Beijing dataset and calculate the time costs. The starting position and end position of these agents' trips are randomly generated between all AOIs of the Beijing dataset. If the distance between the starting position and end position is less than 5 kilometers, they use walking mode, otherwise, they use driving mode.

We adopt pprof[8], which is a profiling tool for Golang programs, to profile the CPU version of the system on the 900,000 agent scenario to answer the third research question. Based on the pprof report, we organize the percentage of each functional module in the overall CPU computing resource consumption.

Moreover, we build a preliminary application case to show the potential of our system to support sustainable mobility research. In this case, we first simulate the whole-day mobility of Beijing and adopt the carbon emission model [10] to analyze the distribution of carbon emissions from Beijing roads.

5.2 Results and Analysis

5.2.1 Overall Performance. The result of the comparison regarding overall performance among SUMO, Cityflow, CPU version and GPU version of our system are shown in Table 3. In summary, the overall performance of our system is significantly higher than the baseline methods, especially the GPU version. The GPU version of our system achieves 1.36 and 1.78 milliseconds per step in the Grid-40-100k and Grid-80-200k scenarios respectively. The time costs of the CPU version of our system in the two scenarios are 11.17 and 28.15 milliseconds per step respectively. Cityflow shows similar performance to the CPU version of our system, taking 3.31 and 2.55 as long on the two scenarios respectively. SUMO is almost unusable in such relatively large scenarios. Regardless of whether multi-threading is used, its single-step running time exceeds 500 milliseconds.

Table 3: Overall performance comparison.

	Time cost (ms/step)	
Scenario	Grid-40-100k	Grid-80-200k
SUMO-1	1049	2217
SUMO-64	625	1445
Cityflow-64	36.92	71.66
Ours-CPU-64	11.17	28.15
Ours-GPU	1.36	1.78

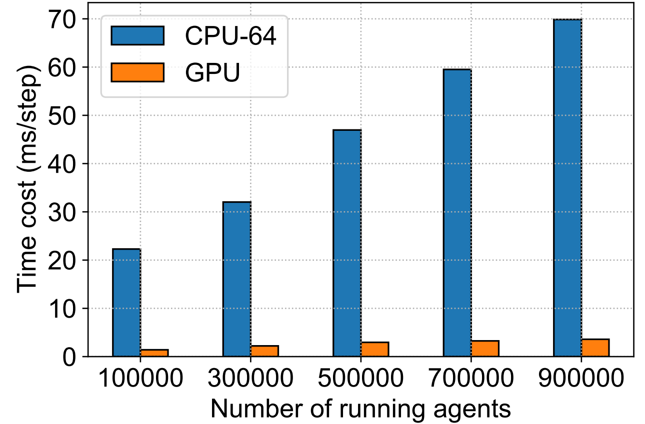

Figure 6: The relationship between the time cost per step of our system and the number of running agents.

5.2.2 The Relationship Between the Computational Performance of Our System and the Number of Agents. The results of the comparison regarding the performance and the number of running agents of our system is shown in Figure 6. The time cost of the CPU version of the system per step is (22.29,32.03,46.96,59.51,69.87) milliseconds at (100,000, 300,000, 500,000, 700,000, 900,000) agent scenarios, respectively. The GPU version's time cost is (1.41,2.22,2.95,3.26,3.59) milliseconds. We can observe that as the number of agents increases, our system shows a more linear growth trend, indicating that our system has fewer sequential processes and greater potential for larger-scale fine-grained human mobility simulation tasks. Meanwhile, facing the simulation task of simulating 900,000 running agents simultaneously, our system can complete the simulation task at 278.77 times the wall clock time by using GPU, which fully demonstrates that the design of the system achieves the goal of high performance.

[8] https://github.com/google/pprof

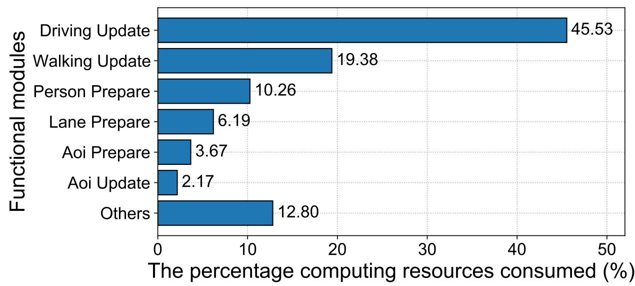

Figure 7: The percentage of computing resources consumed by different functional modules in the CPU version of the system organized from the pprof's report.

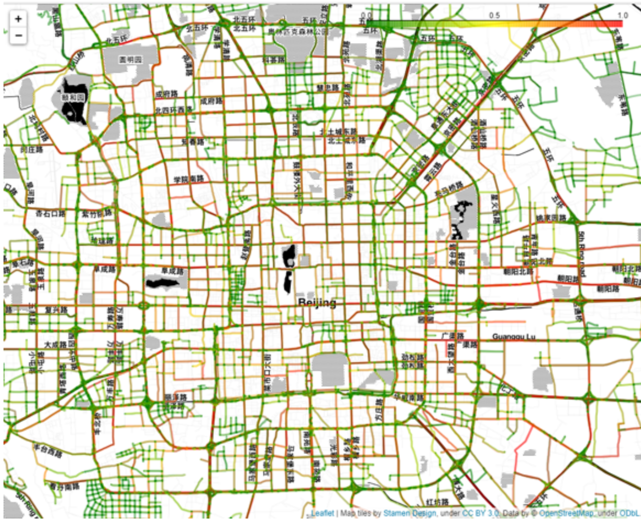

Figure 8: Carbon emission intensity distribution per lane length from Beijing roads based on a full-day mobility simulation.

5.2.3 Functional Module Computation Percentage Analysis. Figure 7 shows the computational resource percentage of different functional modules in the whole simulation process on the CPU version of the system. As can be seen, most of the system's calculations are used for the simulation of the *update* stage, including driving (45.53%) and walking (19.38%). The people data preparation introduced by the two-stage parallel process based on read/write separation design accounts for 10.26%. The maintenance of the indexing subsystem totals about 10%, including lane location relationship linked lists (6.19%) and AOI multi-state index (3.67%). This indicates that the functional modules introduced in the system design are efficient and do not impose significant additional overhead.

5.2.4 Application Case. Based on the system, we obtained all-day fine-grained spatio-temporal human mobility data for central Beijing, which is one of the largest cities in China. We use such all-day fine-grained spatio-temporal human mobility data to estimate the distribution of carbon emission intensity from Beijing roads, whose result is illustrated in Figure 8. Specifically, the velocity, acceleration, and distance of each vehicle at each moment obtained through mobility simulation are input into the carbon emission model to obtain the carbon emissions of the vehicle at the current time. The carbon emission amounts are then aggregated by the road to obtain the result. Based on the preliminary estimation experiment, researchers can further investigate interesting topics such as how to regulate travel demand to reduce emissions and so on.

6 RELATED WORKS

6.1 Mobility Simulation

Mobility simulation is a feasible way to convert the input coarse-grained human mobility data into fine-grained human mobility data. Currently, the most popular open-source mobility simulation system is SUMO [2]. SUMO supports a rich set of mobility models and can simulate cars, buses, trains, bicycles, pedestrians, public transport, and more. But the biggest problem with SUMO is that its single-threaded architecture makes it slower than wall clock time when simulating more than 80,000 cars at the same time. Therefore, the poor computational performance limits the use of SUMO for city-level spatio-temporal mobility simulation. Cityflow [25] and QarSUMO [5] are also parallel mobility simulation solutions, which focus on the simulation and acceleration of vehicle driving. Cityflow [25] uses a barrier-based architecture to achieve proper multi-threaded acceleration by dividing the simulation step into multiple processes such as planning the route, getting action, updating location, updating action, and updating leader and gap. QarSUMO [5] splits the map into multiple partitions, each of which is simulated by a SUMO simulator, and the different partitions exchange incoming and outgoing vehicles through communication, thus achieving the parallel acceleration of the SUMO simulator. Mirage [26] is an efficient and extensible city simulation framework that simulates multiple elements in a city and the interactions and dependencies between them. Its mobility module, which is the predecessor of the work, adopts parallel safety design principle and Intel Threading Building Block[9] (TBB) to implement multi-threaded mobility simulation, and it can calculate several times faster than the wall clock time in the face of urban-level mobility simulation tasks.

6.2 Mobility Prediction and Generation

Prediction and generation are also commonly used to obtain human mobility data. As prediction methods, DeepTransport [18] proposes a long short-term memory (LSTM) based module to understand and predict human mobility. DeepMove [11] adopts an attentional recurrent network for mobility prediction based on lengthy and sparse trajectories. [6] proposes a context-aware model named DeepJMT to jointly do mobility and timestamp prediction. As generation methods, [15] combines variational autoencoder (VAE) and sequence-to-sequence (seq2seq) model to do human mobility reconstruction tasks. MoveSim [12] proposes a generative adversarial framework and utilizes the prior knowledge of human mobility regularity to improve mobility generation quality. [20] proposes a two-stage generative adversarial network (GAN) to generate spatio-temporal human mobility data based on 15-second interval taxi

[9] https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html

trajectories. [22] presents a generative adversarial imitation learning framework to generate artificial activity trajectories. Due to the spatio-temporal granularity of the data, most of the prediction and generation methods focus on coarse-grained human mobility, which is the data basis for the mobility simulation.

7 CONCLUSION

By analyzing the spatio-temporal dependencies in mobility simulation and the corresponding technical issues, we design the two-stage parallel process based on read/write separation and the indexing subsystem to achieve parallel acceleration of mobility simulation. We implement the whole system on both CPU and GPU and show through experiments and cases that the system we design achieves the expected results in terms of performance and can support innovative applications on sustainable mobility. Powered by the high-performance system, researchers have more room to introduce methods such as reinforcement learning to propose intelligent solutions to sustainable policy-making problems in cities. For example, researchers can try to modify mobility infrastructure settings such as traffic lights, intersection road markings, and road speed limits on the existing road network to improve mobility costs across the city, including but not limited to time costs, environmental costs, etc. Researchers can also enforce certain transportation mode selection rules to observe the impact of policies on urban transportation, such as requiring that only walking or cycling is required for trips less than a certain distance. Moreover, the game between accessibility and residents' income and expenditure is also an interesting research topic that can be based on simulation.

In future work, we will focus on the combination of mobility and carbon emission estimation, try to track the environmental impact of the mobility process from the perspective of the agent, and study methods to reduce carbon emissions caused by the mobility process.

ACKNOWLEDGMENTS

This work is sponsored by Shanghai Biren Technology Co., Ltd. through the Collaboration on GPGPU Innovation Research between Tsinghua University and Shanghai Biren Technology Co., Ltd.

REFERENCES

[1] Lukas Ambühl, Monica Menendez, and Marta C González. 2023. Understanding congestion propagation by combining percolation theory with the macroscopic fundamental diagram. *Communications Physics* 6, 1 (2023), 26.
[2] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. 2011. SUMO–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind.
[3] John L Bowman and Moshe E Ben-Akiva. 2001. Activity-based disaggregate travel demand model system with activity schedules. *Transportation research part a: policy and practice* 35, 1 (2001), 1–28.
[4] Matteo Böhm, Mirco Nanni, and Luca Pappalardo. 2022. Gross polluters and vehicle emissions reduction. *Nature Sustainability* 5, 8 (June 2022), 699–707. https://doi.org/10.1038/s41893-022-00903-x
[5] Hao Chen, Ke Yang, Stefano Giovanni Rizzo, Giovanna Vantini, Phillip Taylor, Xiaosong Ma, and Sanjay Chawla. 2020. QarSUMO: A Parallel, Congestion-optimized Traffic Simulator. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*. 578–588.
[6] Yile Chen, Cheng Long, Gao Cong, and Chenliang Li. 2020. Context-aware Deep Model for Joint Mobility and Time Prediction. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. ACM, Houston TX USA, 106–114. https://doi.org/10.1145/3336191.3371837
[7] Kai-Fung Chu, Albert YS Lam, and Victor OK Li. 2019. Deep multi-scale convolutional LSTM network for travel demand and origin-destination predictions. *IEEE Transactions on Intelligent Transportation Systems* 21, 8 (2019), 3219–3232.
[8] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.
[9] Sven Eggimann. 2022. The potential of implementing superblocks for multifunctional street use in cities. *Nature sustainability* 5, 5 (2022), 406–414.
[10] Mehrsa Ehsani, Abbas Ahmadi, and Dawud Fadai. 2016. Modeling of vehicle fuel consumption and carbon dioxide emission in road transport. *Renewable and sustainable energy reviews* 53 (2016), 1638–1648.
[11] 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 *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*. ACM Press, Lyon, France, 1459–1468. https://doi.org/10.1145/3178876.3186058
[12] Jie Feng, Zeyu Yang, Fengli Xu, Haisu Yu, Mudan Wang, and Yong Li. 2020. Learning to Simulate Human Mobility. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, Virtual Event CA USA, 3426–3433. https://doi.org/10.1145/3394486.3412862
[13] Alexander A Ganin, Maksim Kitsak, Dayton Marchese, Jeffrey M Keisler, Thomas Seager, and Igor Linkov. 2017. Resilience and efficiency in transportation networks. *Science advances* 3, 12 (2017), e1701079.
[14] Dirk Helbing and Peter Molnar. 1995. Social force model for pedestrian dynamics. *Physical review E* 51, 5 (1995), 4282.
[15] Dou Huang, Xuan Song, Zipei Fan, Renhe Jiang, Ryosuke Shibasaki, Yu Zhang, Haizhong Wang, and Yugo Kato. 2019. A Variational Autoencoder Based Generative Model of Urban Human Mobility. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE, San Jose, CA, USA, 425–430. https://doi.org/10.1109/MIPR.2019.00086
[16] Stefan Krauß, Peter Wagner, and Christian Gawron. 1997. Metastable states in a microscopic model of traffic flow. *Physical Review E* 55, 5 (1997), 5597.
[17] Yanchi Liu, Chuanren Liu, Nicholas Jing Yuan, Lian Duan, Yanjie Fu, Hui Xiong, Songhua Xu, and Junjie Wu. 2017. Intelligent bus routing with heterogeneous human mobility patterns. *Knowledge and Information Systems* 50, 2 (2017), 383–415.
[18] Xuan Song, Hiroshi Kanasugi, and Ryosuke Shibasaki. 2016. Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. 2618–2624.
[19] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. 2000. Congested traffic states in empirical observations and microscopic simulations. *Physical review E* 62, 2 (2000), 1805.
[20] Xingrui Wang, Xinyu Liu, Ziteng Lu, and Hanfang Yang. 2021. Large Scale GPS Trajectory Generation Using Map Based on Two Stage GAN. *Journal of Data Science* (2021), 126–141. https://doi.org/10.6339/21-JDS1004
[21] Yuandong Wang, Hongzhi Yin, Hongxu Chen, Tianyu Wo, Jie Xu, and Kai Zheng. 2019. Origin-destination matrix prediction via graph convolution: a new perspective of passenger demand modeling. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 1227–1235.
[22] Yuan Yuan, Jingtao Ding, Huandong Wang, Depeng Jin, and Yong Li. 2022. Activity Trajectory Generation via Modeling Spatiotemporal Dynamics. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4752–4762.
[23] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.
[24] Guozhen Zhang, Zihan Yu, Depeng Jin, and Yong Li. 2022. Physics-infused Machine Learning for Crowd Simulation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2439–2449.
[25] Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Li. 2019. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The world wide web conference*. 3620–3624.
[26] Jun Zhang, Depeng Jin, and Yong Li. 2022. Mirage: an efficient and extensible city simulation framework (systems paper). In *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*. 1–4.