CBLab: Supporting the Training of Large-scale Traffic Control Policies with Scalable Traffic Simulation

Chumeng Liang
caradryan2022@gmail.com
Shanghai Jiao Tong University
China

Zherui Huang
huangzherui@sjtu.edu.cn
Shanghai Jiao Tong University
China

Yicheng Liu
liuyicheng1515@sjtu.edu.cn
Shanghai Jiao Tong University
China

Zhanyu Liu
zhyliu00@sjtu.edu.cn
Shanghai Jiao Tong University
China

Guanjie Zheng*
gjzheng@sjtu.edu.cn
Shanghai Jiao Tong University
China

Hanyuan Shi
shihanyuan@sjtu.edu.cn
Independent Researchers
China

Kan Wu
kanwu@zhejianglab.com
Research Center for Intelligent Transportation, Zhejiang Lab
China

Yuhao Du
apiadu17a6@gmail.com
Independent Researchers
China

Fuliang Li
tjfulianglee@gmail.com
Baidu
China

Zhenhui Li
jessielzh@gmail.com
Yunqi Academy of Engineering
China

ABSTRACT

Traffic simulation provides interactive data for the optimization of traffic control policies. However, existing traffic simulators are limited by their lack of scalability and shortage in input data, which prevents them from generating interactive data from traffic simulation in the scenarios of real large-scale city road networks.

In this paper, we present **C**ity **B**rain **Lab**, a toolkit for scalable traffic simulation. CBLab consists of three components: CBEngine, CBData, and CBScenario. CBEngine is a highly efficient simulator supporting large-scale traffic simulation. CBData includes a traffic dataset with road network data of 100 cities all around the world. We also develop a pipeline to conduct a one-click transformation from raw road networks to input data of our traffic simulation. Combining CBEngine and CBData allows researchers to run scalable traffic simulations in the road network of real large-scale cities. Based on that, CBScenario implements an interactive environment and a benchmark for two scenarios of traffic control policies respectively, with which traffic control policies adaptable for large-scale urban traffic can be trained and tuned. To the best of our knowledge, CBLab is the first infrastructure supporting traffic control policy optimization in large-scale urban scenarios. CBLab has supported the City Brain Challenge @ KDD CUP 2021. The project is available on GitHub: https://github.com/CityBrainLab/CityBrainLab.git.

CCS CONCEPTS

• **Information systems** → **Spatial-temporal systems**.

KEYWORDS

Traffic Simulation, Traffic Control Policy, Traffic Signal Control, Large-scale Data

ACM Reference Format:
Chumeng Liang, Zherui Huang, Yicheng Liu, Zhanyu Liu, Guanjie Zheng, Hanyuan Shi, Kan Wu, Yuhao Du, Fuliang Li, and Zhenhui Li. 2023. CBLab: Supporting the Training of Large-scale Traffic Control Policies with Scalable Traffic Simulation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599789

1 INTRODUCTION

Well-crafted traffic control policies, such as traffic signal control and congestion pricing, are expected to improve the efficiency of urban transportation. In recent years, many studies have been conducted to optimize the traffic control policies according to real-time traffic data [2, 6, 11, 13, 15, 18, 21, 22, 26, 28–30]. These policies depend on data generated by interaction with the traffic environment where they explore to make good decisions under different consequences.

However, real-world urban traffic cannot provide enough interactive data to train these policies, because the exploration of the policy may have a toxic impact on the urban traffic e.g. provoke severe congestion. Traffic simulators are therefore born as alternatives to provide traffic environments for traffic control policies to

*corresponding author

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599789

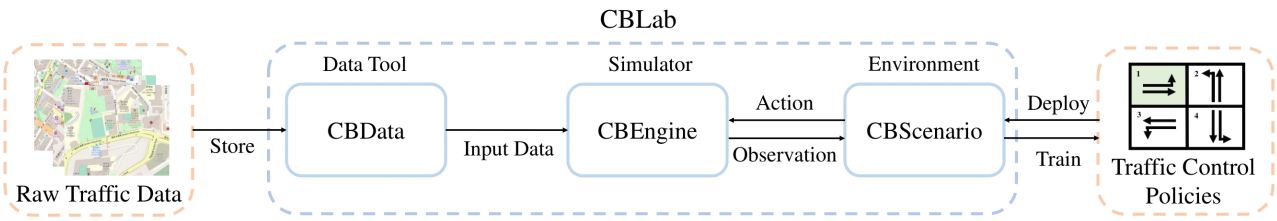

Figure 1: An overview of CBLab.

interact with. These simulators [3, 10, 27] simulate the microscopic evolution of the urban traffic. For each time step, they describe the traffic state, obtain a traffic action from the decision made by traffic control policies and make it happen in the simulation. Traffic control policies can then learn from how the traffic evolves under certain actions and improve decision-making.

While existing traffic simulators help hatch various traffic control policies successfully, they still come with drawbacks. Current simulators, as they were designed primitively, support simulation in road networks smaller than one hundred intersections and cannot scale to city-level traffic, which involves thousands of intersections. Due to limits in efficiency and scalability, these simulators are either not able to conduct a city-level simulation in a feasible time or set to prevent masses of vehicles from coming in the traffic.

Another concern lies in the shortage of input data for large-scale traffic simulation. Although the map data of the main cities in the world is completed, there is an absence of infrastructure for access to the map data and a pipeline to transform it into simulation inputs. Therefore, inputs for traffic simulation only come from manual work and are limited to a small set of road networks [19, 20, 23, 30] whose scales are often dozens of intersections (e.g. 4x3 or 4x4) - much smaller than real urban road networks.

To overcome two aforementioned drawbacks, we propose **C**ity **B**rain **Lab**, a novel toolkit for scalable traffic simulation. CBLab consists of three components: a microscopic traffic simulator CBEngine, a data tool CBData, and a traffic control policy environment CBScenario (See Figure 1). Benefiting from well-designed parallelization, CBEngine is of high efficiency and scalability and is capable of running the traffic simulation on the scale of 10,000 intersections and 100,000 vehicles with a real-simulation time ratio of 1:4 with ordinary computing hardware. CBData includes an accessible dataset that contains raw road networks of 100 main cities all around the world. A pipeline is prepared to automatically transform the raw data into input data for traffic simulation. Combining CBEngine and CBData, users can easily start up traffic simulation on real city-level road networks. Based on the scalable traffic simulation, we implement CBScenario as an environment for two traffic control policies: traffic signal control and congestion pricing. Users can design, develop, and train traffic control policies in the framework of CBScenario. To the best of our knowledge, we are the first to provide infrastructure for large-scale traffic control policy optimization.

Our contribution can be summarized as follows.

- We develop a scalable traffic simulator CBEngine that supports city-level microscopic traffic simulation for the first time.
- We develop a data tool CBData to provide input data for large-scale traffic simulation.
- We implement an interactive environment CBScenario for training traffic control policies under a large-scale setting.

The original version of CBLab has successfully supported the City Brain Challenge @ KDD CUP 2021 with 1,156 participating teams. See Appendix C for details.

2 CBENGINE: CITY-SCALE TRAFFIC SIMULATION ENGINE

In this section, we introduce CBEngine, the traffic simulator. We demonstrate its traffic modeling and conduct extensive experiments to show the efficiency, scalability, and plausibility of CBEngine.

2.1 Overview of Simulation Modeling

Traffic simulators take the road network and the traffic flow (vehicles in the traffic) as inputs and aim to simulate their interaction. Road networks describe the topology of roads and intersections. Traffic flows describe the origins, destinations, and routes of vehicles. As shown in Figure 2, the road network interacts with vehicles through traffic signal lights, which control the passing of vehicles at intersections. When the simulation starts, vehicles in the traffic flow set out from their origins, travel down the routes, and finally arrive at their destinations. Roads, intersections, traffic signal lights, and vehicles can be considered as *traffic elements*.

We can formulate the simulation in the form of states and actions. The simulator holds states for traffic elements at the time step t. For the vehicle, the state is its location and speed. For the traffic signal light, the state is the current signal. Actions of elements then iterate the current state to that in the next time step. For example, the acceleration of a vehicle rises up the vehicle's speed at the next time step. Finally, the simulation moves on to the next time step, where a new state-action iteration will begin. Let s_t, a_t, and s_{t+1} denote the current state, action, and the next state, the traffic modeling in the simulation can be concluded by Eq 1.

$$s_t + a_t \rightarrow s_{t+1} \tag{1}$$

2.2 Road Network

In CBEngine, **Road Network** is the topological network where vehicles drive. It consists of two components: roads and intersections. **Road** models the road segment in the real-world road network. A road may include multiple lanes. Each **lane** holds one or more vehicles. **Intersection** is the nexus of different roads. Through **lane links** in the intersection, lanes of different roads connect to each

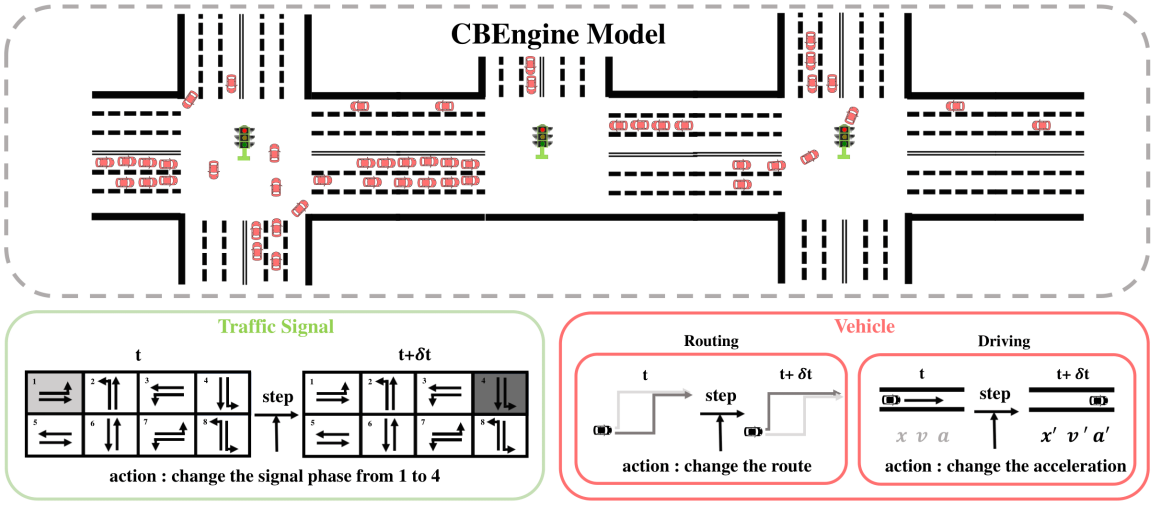

Figure 2: The Traffic Model of CBEngine

other. **Traffic signal light** is another key element in the intersection, which assigns a true-or-false signal for each lane link. Vehicles can only move from one road to another through available lane links with a true signal.

2.3 Driving Model

Behaviors of vehicles are controlled by the driving model in the traffic simulation. Driving models determine how vehicles move on the road. Towards simulating the behaviors of vehicles, researchers have proposed various models [8, 24, 25].

The default driving model of CBEngine is a modified version of the driving model used in Cityflow [27], originating from the driving model proposed by Stefan Krauß [8]. The key idea is that: the vehicle will drive as fast as possible subject to safety regularization. Specifically, the maximum speed of the vehicle is subject to several static or dynamic speed constraints. Vehicles will accelerate or decelerate to the speed with the maximum acceleration or deceleration. In the implementation, the maximum acceleration (deceleration) is a hyperparameter and is open to users to tune with an API. The considered speed constraints are listed and discussed respectively as follows:

- Road speed limit
- Collision-free following & leading speed
- Cutting-in collision-free speed
- Traffic-signal safe speed

Road speed limit. Each road has its own speed limit. This is a static speed constraint.

Collision-free following & leading speed. To avoid collisions, vehicles need to adapt their speed to the speed of their following and leading vehicles. We use the collision-free following speed to model the max speed constrained by the leading vehicle. Following the driving model in Cityflow, we compute these two constraints with Eq 2. It takes v current speed of the vehicle, v_L current speed of the leading vehicle, d maximum deceleration of the vehicle, d_L maximum deceleration of the leading vehicle, D the current distance between two vehicles, *interval* the length of each time step as parameters to compute the collision-free following speed s_{cfs}.

$$
\begin{aligned}
a &= \frac{1}{2 \cdot d}, b = \frac{interval}{2} \\
c &= \frac{v \cdot interval}{2} - \frac{v_L^2}{2 \cdot d_L} - D \\
s_{cfs} &= \frac{-b + \sqrt{b^2 - 4ac}}{2 \cdot a}
\end{aligned}
\tag{2}
$$

To compute the collision-free leading speed s_{cls}, we use a mirror equation of Eq 2. We use the d_F the maximum deceleration of the following vehicle and v_F the speed of the following vehicle to replace d_L and v_L in the Eq 2. The collision-free following and leading speed constraints can be summarized as Eq 3.

$$
s_{cls} \leq v \leq s_{cfs} \tag{3}
$$

Cutting-in collision-free speed. CBEngine supports self-adaptive lane changing within the road (Cutting-in). This asks for cutting-in collision-free following and leading speed, avoiding collisions with the leading and following vehicles in the target lane. We compute this constraint using Eq 2 with v_L, d_L, and D given by the leading vehicle in the target lane and v_F, d_F, and D given by the following vehicle in the target lane.

An exception happens when the vehicle needs to conduct an emergent cutting-in. This takes place when the vehicle is very close to the intersection but still in the wrong lane (e.g. The vehicle is in the go-straight lane but needs to turn left). On this occasion, the vehicle tries to stop to wait until it is able to change the lane. Therefore, the maximum speed constraint equals zero and the vehicle will decelerate to zero with its maximum deceleration.

Traffic-signal safe speed. Vehicles heading for an intersection are subject to two constraints determined by the traffic signal. First, the current speed can be decelerated to zero within the remained passing time of the traffic signal. Second, the driving distance cannot exceed the distance to the intersection, under the decelerating process defined in the first constraint.

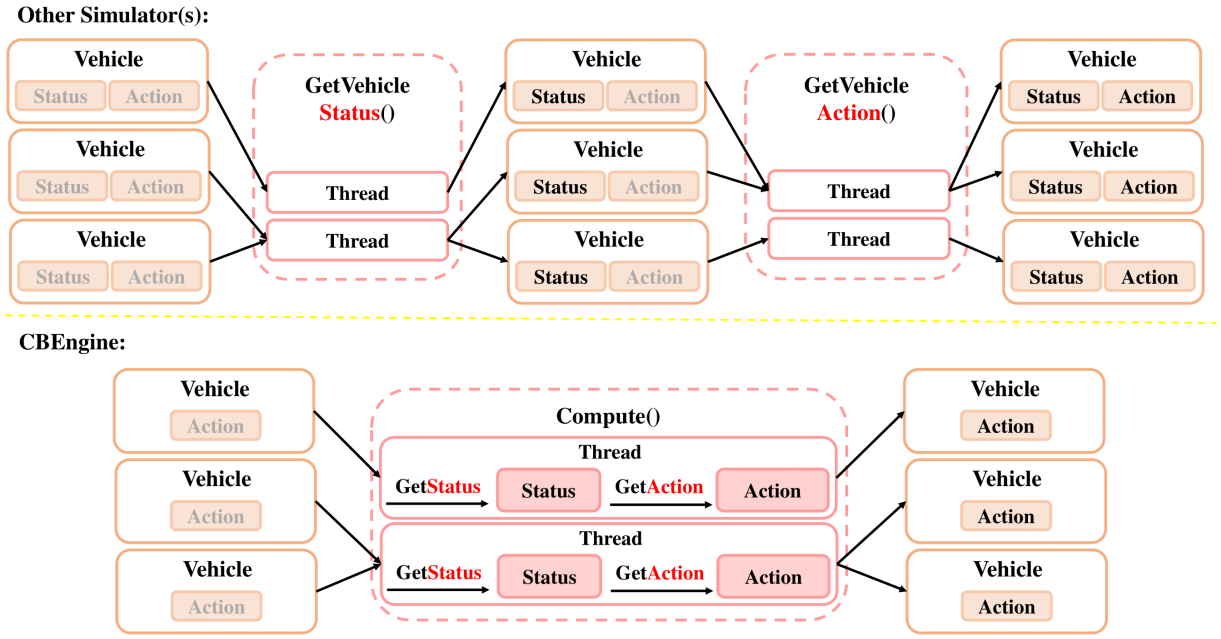

Figure 3: Comparison between CBEngine and other simulators on parallelization design of computing actions for vehicles. The gray data member indicates it is not updated, while the black one is updated.

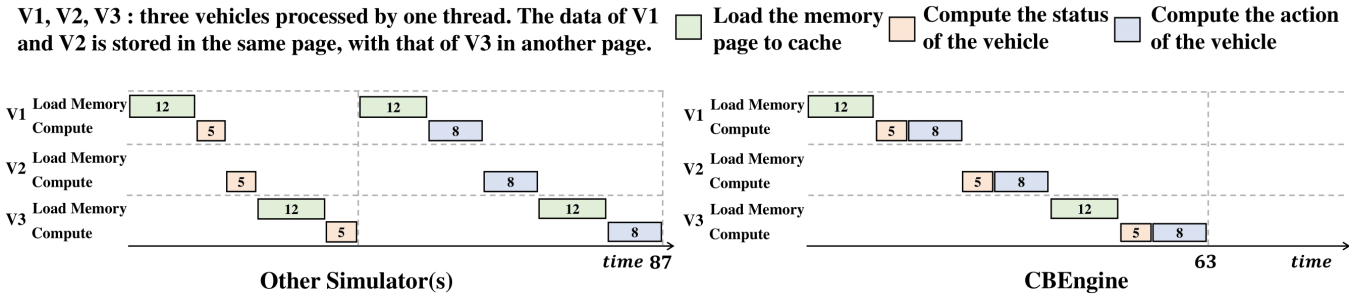

Figure 4: A schedule case of computing vehicle action. The length of the bar indicates the time the corresponding task consumes.

2.4 Model Customization

An important function implemented in CBEngine is vehicle model customization. In the simulator, both the driving model and the routing model are black boxes. We modularize two models as an independent C++ class in our implementation, respectively. Users can customize two models by modifying the class without modification to other parts of CBEngine. Our documentation in Appendix A.3 provides detailed instructions. To the best of our knowledge, CBEngine is the first traffic simulator to support easy-to-use customization for driving and routing models.

2.5 Efficiency and Scalability

The major bottleneck of supporting city-wide traffic control policy training is the limit in efficiency and scalability of existing simulators, of which extensive efforts have been made in CBEngine for improvement. The progress in efficiency and scalability can be validated by the time cost of running the simulation on the same traffic and the maximum capacity of intersections and vehicles. In this subsection, we first introduce several systematic designs of CBEngine that greatly promote its efficiency and scalability. Then, we conduct extensive experiments to validate the efficiency and scalability of CBEngine. We mainly compare CBEngine with two widely-used traffic simulators, SUMO [10] and Cityflow [27].

2.5.1 System Implementation. To overcome the disadvantages in efficiency and scalability, we introduce three designs for the system implementation of CBEngine, among which *Parallel Design in Computing the Vehicle Behavior* mainly contributes to the efficiency while *Lane Changing in Driving Model* and *Intersection Links* contribute to the scalability.

Parallel Design in Computing the Vehicle Behavior. The future state in the simulation is determined by the present state and the current action. This is the major computational job in traffic simulation, which costs over 90% of the step time. In CBEngine, we carefully design the parallelization process of computing the action for vehicles. The comparison of our design and that of other simulators is shown in Figure 3.

Dataset	Nanchang	Changchun	JiNan	Shenzhen	Hangzhou	Shanghai
Intersection Num	1506	2228	2314	3427	3434	8474
Vehicle Num	~25000	~50000	~50000	~70000	~70000	~120000
	Time Cost					
SUMO	1239.93 (±3.58)	2091.60 (±7.84)	2151.01 (±70.64)	3103.58 (±110.08)	3199.14 (±70.87)	6173.51 (±75.27)
Cityflow	164.08 (±6.30)	243.22 (±11.85)	242.47 (±21.34)	289.31 (±**1.74**)	310.98 (±18.22)	664.18 (±21.01)
CBEngine	**104.39** (±**0.54**)	**169.36** (±**1.60**)	**174.29** (±**1.93**)	**243.26** (±1.75)	**245.01** (±**2.16**)	**472.07** (±**2.90**)

Table 1: Comparison results of efficiency experiments in six real-world cities.

The process to compute actions for vehicles can be divided into two stages: getting the Status and getting the Action. In the first stage, the simulator collects information for computing the action of a vehicle. The set of information is denoted as the *Status*. In the second stage, the simulator computes the action according to Status. Existing simulators parallelize these two stages respectively. Vehicle objects in the simulator need to keep two data members: the Status and the Action. Two members are updated sequentially by two parallelized methods: GetVehicleStatus() and GetVehicleAction().

In CBEngine, we assemble two stages in one parallelized method: Compute(). We implement this by adjusting data dependencies and reconstructing the parallel architecture. This design benefits efficiency from two perspectives. First, the space cost is lower. We bind the Status data on the Thread object rather than on the Vehicle objects. This is because the number of threads is quite smaller than that of vehicles. Second, by merging two stages, the CPU is more likely to access existing pages in the memory when getting the action, because those pages are recently loaded to the memory when getting the status. Since this design reduces the number of times that the CPU loads pages from the memory, it decreases the number of cache misses and the time cost.

Figure 4 raises a case comparing the scheduling of one thread processing vehicles in other simulators to that in CBEngine. For each vehicle, the thread needs to compute its status and then its action accordingly. The page where the vehicle is stored is required to be loaded in memory for access to the vehicle. Assume that the cache can only store one page. For other simulators, the processing is divided into two stages, while each stage loads two pages from memory. This is because the thread needs to access all three vehicles in each stage. By contrast, processing in CBEngine combines two stages and does not access new vehicles until the job on the vehicle is finished. This design helps reduce the operation of loading pages and saves processing time.

Lane Changing in Driving Model. Lane changing is a driving action. Drivers may change the lane if the current lane is too congested. However, lane changing is hard to simulate. Cityflow [27], one of our baselines, omits all lane changing except those happening at the intersection. The lane of the vehicle is determined by the direction it will turn. This simplifies the implementation but leads to poor plausibility because vehicles on the road cannot make full use of all lanes as they do in real urban traffic. They have to stay at the current lane and may wait a long time, although their neighbor lane is clear. Furthermore, to avoid collision on this occasion, Cityflow does not allow new vehicles to come in until the lane is relatively unblocked. This limitation severely impacts the scalability of Cityflow and explains the fact in our experiment of scalability that Cityflow cannot hold 1,000,000 vehicles.

In CBEngine, we implement a driving model allowing lane changing. Vehicles are put in a random lane when they get into the road. They will try lane changing according to the direction they are to turn to. But if that lane is in congestion, they will keep going in the current lane which is relatively clear. This design achieves higher plausibility and provides stronger scalability for CBEngine.

Intersection Links. Another key mechanism of the traffic simulator is the intersection link. SUMO and Cityflow track the behavior of vehicles inside the intersection. However, due to the limit in the implementation, this design may lead to deadlocks very frequently in the practice, especially when running large-scale traffic simulations. This is because the track of some vehicles may block that of others. Therefore, we conduct simplification here to avoid such deadlocks. When a vehicle passes the intersection, the intersection will hold it for a while and then send it to the target road. We believe that the effect of the intersection link can be simulated by the holding time. To the best of our knowledge, our design avoids all such deadlocks in practice.

2.5.2 *Experiments.*

Experimental Setup. To evaluate the efficiency and scalability of CBEngine, we compare it with two widely-used open-source microscopic traffic simulators, SUMO [10] and Cityflow [27]. We compare these simulators in three aspects, running time, road network scalability, and traffic flow scalability. All the experiments are conducted on a Ubuntu20.04 system with a 40-core CPU and 128GB RAM. The unit of time cost is second for all three experiments. More details of the experimental setup are given in Appendix B.

Experiment 1: Efficiency. We run a one-hour traffic simulation on urban traffic cases of six cities with distinct scales and compare the time cost of baselines and our simulator. Road networks of these cases are obtained and cleaned from OpenStreetMap [7]. Traffic

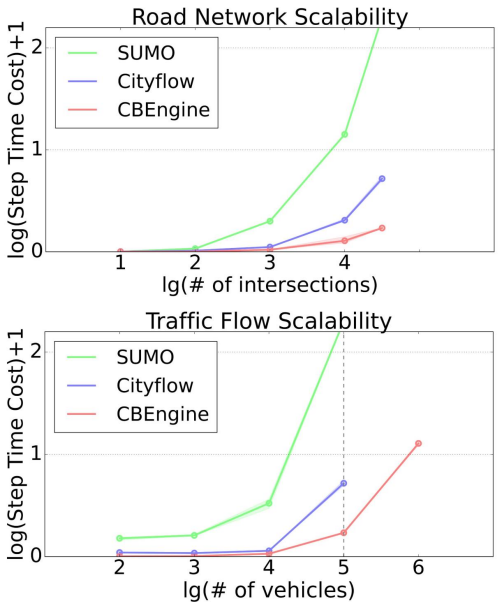

Figure 5: Comparison results of experiments for scalability.

flows are generated according to the scale of the road network. Results are displayed in Table 1.

We can observe that CBEngine achieves significant improvement in simulation efficiency (usually 30%-40% compared with Cityflow and more than 90% compared with SUMO). The stability of CBEngine is distinctly better than that of baselines. Furthermore, the gap between baselines and ours grows with the scale of traffic cases, indicating that CBEngine can adapt well to large-scale cases.

Experiment 2: Scalability on Road Networks. Simulators with high scalability on road networks can run the simulation efficiently on large-scale road networks. To explore the scalability of baselines and CBEngine on road networks, we run a traffic simulation on road networks of different scales. We select five regions from real road networks with $\{10, 10^2, 10^3, 10^4, 10^{4.5}\}$ intersections. The upper bound is set as $10^{4.5}$ because this is the largest road network for a single city in OpenStreetMap. For each setting, we repeat the experiment three times. We report the time cost of single-step simulation for baselines and CBEngine as well as the range.

The results are visualized in Figure 5 (left). CBEngine outperforms two baselines in time cost on road networks with all scales. Take the experiment setup under the road network with the largest scale as a quantitative example. The average single step time cost of CBEngine is 0.2670 seconds, while that of SUMO [10] and Cityflow [27] are 9.1832 seconds and 1.0343 seconds, respectively.

Experiment 3: Scalability on Traffic Flows. With high scalability on traffic flows, the simulator keeps efficient under heavy traffic. Similar to Experiment 2, we conduct an experiment to evaluate the scalability of traffic flows of baselines and CBEngine. We generate five traffic flows with $\{10^2, 10^3, 10^4, 10^5, 10^6\}$ on-way vehicles. For each setting, we repeat the experiment three times. We report the time cost of a single step as well as the range.

The results are visualized in Figure 5 (right). CBEngine outperforms two baselines under different scales of traffic flows. To give a quantitative example, the average single step time cost of CBEngine under the traffic flow of 10^5 vehicles is 0.2610 seconds, while that of SUMO [10] and Cityflow [27] are 8.9111 seconds and 1.1058 seconds, respectively. Specifically, two baselines are not able to run the case with 1,000,000 vehicles. For Cityflow, the reason has been discussed in Section 2.5.1.

2.6 Plausibility

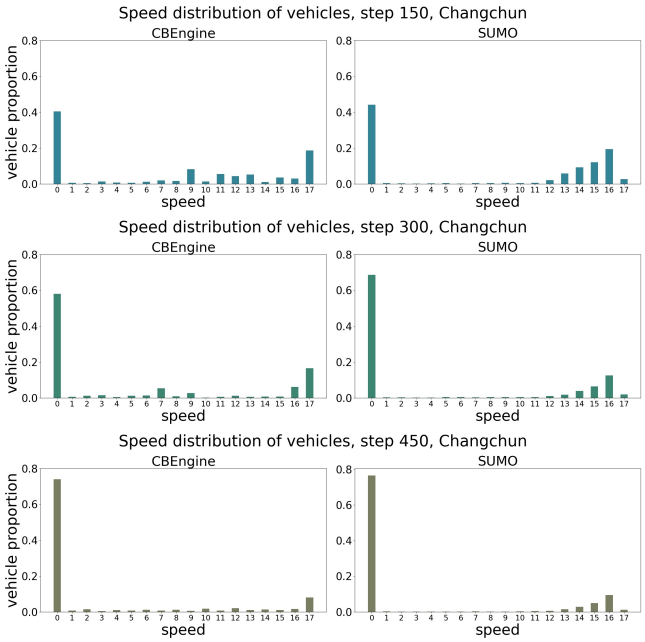

Figure 6: Comparison of speed distributions of vehicles, CBEngine and SUMO

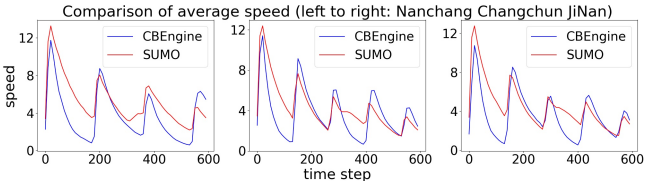

Figure 7: Comparison of the average speed of vehicles, CBEngine and SUMO

Although CBEngine provides driving model customization, which decreases the impact of the default driving model, we conduct two experiments to evaluate its plausibility by comparing the vehicle behavior of CBEngine and that of SUMO [10]. First, we compare the average speed of vehicles in CBEngine and SUMO under the same simulation setups over 600 seconds. The experiments are finished on three experimental setups used in our efficiency and scalability experiments: Nanchang, Changchun, and JiNan. The result is visualized in Figure 7. Overall tendencies of two average speeds fit each other approximately, while that in CBEngine is more volatile. This can be explained by the difference in the driving model and

will not have visible impacts on traffic flow statistics, which is the main factor in learning traffic control policies.

The second experiment focuses on the speed distributions of vehicles in CBEngine and SUMO. We compare the speed distributions in 150 seconds, 300 seconds, and 450 seconds under the experimental setup of Changchun used in our efficiency experiments. The comparison is visualized in Figure 6. Normalized Wasserstein distance between two distributions is shown in Table 2. The two distributions are similar to each other roughly but differ in details, which may not influence the overall performance of the simulated traffic in the level of traffic statistics. Also, the driving model customization supported by CBEngine makes it possible for users to use driving models according to their needs, which greatly improves the plausibility of CBEngine, compared to SUMO and Cityflow.

Time Step	150	300	450
Normalized Wasserstein Distance	0.0141	0.0124	0.0089

Table 2: Normalized Wasserstein distance between speed distribution of CBEngine and SUMO

3 CBDATA: TRAFFIC DATA NETWORK CONNECTED TO CBENGINE

In this section, we introduce our data tool CBData. CBData serves to provide enriched input data supporting large-scale traffic simulation. The support is achieved by a dataset with raw road networks of 100 main cities all around the world and the transformation pipeline shown in Figure 8. Moreover, CBData includes two other pipelines that help the simulator learn from other traffic data.

3.1 Pipeline: Simulation Input Data Supporting

Despite existing traffic simulators being capable of simulating the evolution of urban traffic, the application of traffic simulation is vastly limited by the shortage of input data. Specifically, to start up a simulation, the simulator takes road networks and traffic flows as input data. At present, there is no convenient access to these two kinds of data, although road networks of almost all main cities in the world can be extracted from open-source map data [7].

To disentangle this problem, we implement a pipeline to bridge open-source map data and input data for simulation. This pipeline provides a one-click service to offer enriched input data for large-scale traffic simulation, solving the shortage of input data.

As shown in Figure 8, the pipeline consists of a dataset, a preprocessor, and a flow generator.

Dataset: We obtain the map data of the whole world from OpenStreetMap [7]. We extract the road network data of 100 main cities and store the data in our dataset. The dataset is now available on Google Drive (See Appendix A.4). Users can directly download the data and pick up their interested road networks. Details of the dataset are given in Appendix D.

Preprocessor: The preprocessor first constructs the road network as a raw graph by matching and connecting edges and nodes. The raw graph is then cleared to remove the redundant nodes and graphs. This is necessary because redundancy is common in open-source map data. After removing the redundancy, the reduced graph is transformed into the road network in the standard format.

Flow Generator: The flow generator generates the traffic flow for a road network. Given the total number of vehicles, the generator assigns origins and destinations for these vehicles, respectively, which distribute as average as possible. The default route for each pair of the origin and the destination is the shortest path and can be changed by the routing model when running the simulation.

3.2 Pipeline: Learning to simulate from data

In addition to map data, there is a lot of other traffic data with the potential to enhance the plausibility of traffic simulation. In CBData, we propose two paradigmatic pipelines to illustrate how to learn to simulate from traffic data. Note that we are not to propose effective methods but to provide a paradigm for learning to simulate.

3.2.1 Learning to Simulate Driving. The driving model determines how drivers accelerate and decelerate according to their observation of the circumstance. The driving behavior in different traffic or different cities can be distinctly different. Therefore, learning the driving parameters of the traffic simulator from the traffic data is sound for traffic simulation. It helps the simulator to behave more plausibly like the local drivers.

The goal of learning to drive is to find a set of driving model parameters that can minimize the gap between the traffic data and the simulator, *e.g.* that between the observed speed in the real data and the observed speed in the simulator, with the same traffic flow. As mentioned in Section 2.1, the driving model of CBEngine is easy to modify. Here, we adopt the default model and select three parameters from the model as the parameters to be optimized: acceleration maximum, deceleration maximum, and speed limit.

We use a black-box optimization toolkit OpenBox [9] as the optimization tool. OpenBox searches for parameters to fill the gap between simulation observation and the ground truth. The 1-hour GPS trajectory data of vehicles on two roads in Shenzhen are used to search the parameters. The observation interval is 1 minute. For every 20 minutes, the traffic changes. We expect that acceleration parameters can be continuously learned when the traffic changes.

The loss curves of the learning process of these two avenues are displayed in Figure 10. The gap between the observed speed average and the ground truth is decreasing as time changes. After the traffic changes at the 20th and the 40th minute, the driving model can not fit the new traffic data. Hence, we observe a high loss immediately. After a few minutes, the gap continuously decreases. Note that, the difference still maintains a certain positive value. This implies the potential to change the driving module to pursue a better performance of driving module correction.

3.2.2 Learning to Simulate Routing. The routing model determines how vehicles route themselves, given the origin and destination. A data-dependent routing model can be formulated as a route generator, which generates routes for certain origins and destinations based on real trajectories. We exploit the Recurrent Neural Networks (RNNs) as our route generator [4].

We use part of a vehicle trajectory dataset in Shenzhen, China. This dataset contains 22 different routes in total. We train the RNN [4] to learn the distribution of routes and conduct routing for vehicles in the simulator. Figure 11 shows the result of loss curves and two trajectory generating metrics, BLEU [14] and METEOR [1].

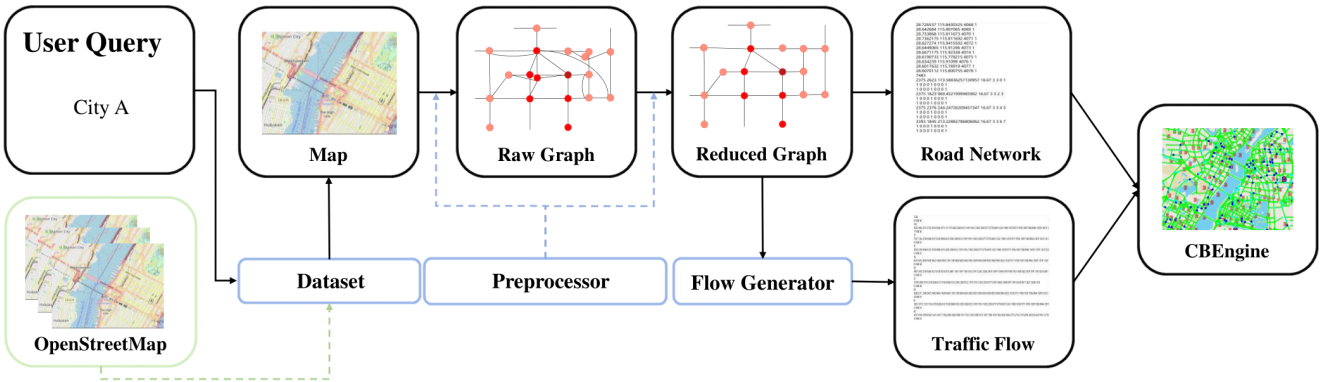

Figure 8: The simulation input data pipeline of CBData.

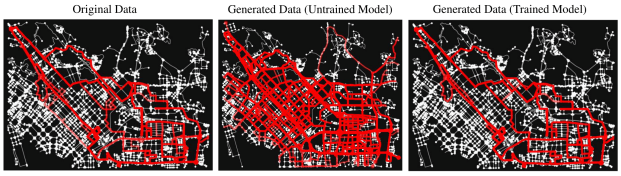

Figure 9: Visualization of the routing result. The deeper red is, the more frequent the route is picked.

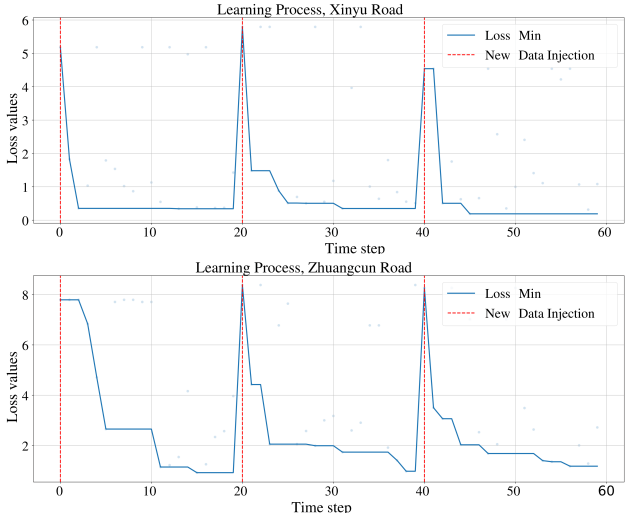

Figure 10: Loss curves during learning the driving module on two roads.

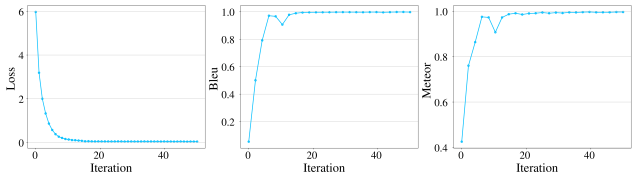

Figure 11: Curves of loss, BLEU, and METEOR during learning the routing module.

The loss converges to 0 at the first twenty iterations, while BLEU and METEOR get close to 1. This indicates that at this stage, routes generated overlap at least one route in the real trajectory data.

In addition, the routing result is visualized in Figure 9. Compared with an untrained generator, the trained generator recognizes the main distribution pattern of real trajectories. Meanwhile, it tends to ignore some infrequently-appearing routes.

4 CBSCENARIO: ENVIRONMENT FOR LARGE-SCALE TRAFFIC CONTROL POLICIES

In this section, we introduce CBScenario, an interactive environment for large-scale traffic control policies. CBScenario benefits from the large-scale traffic simulation supported by CBEngine and CBData and is capable of training traffic control policies for city-level traffic. Concretely, CBScenario includes benchmarks for two traffic control policies: "Traffic Signal Control" and "Congestion Pricing". We conduct experiments on our environment to show the plausibility of the traffic simulation.

4.1 Policy 1: Traffic Signal Control

The traffic signal control problem [20] tries to improve the performance of urban traffic by carefully choosing the phase of traffic signals at intersections. An ideal traffic signal control policy can capture the global and local traffic dynamics and allocate more passing time to the phase with higher traffic pressure. Figure 12 (left) shows the problem setting of traffic signal control.

In consideration of the Markov nature of traffic signal control, the traffic signal control can be formulated as a Markov Decision Process (MDP):

- **State:** Intersection-level and road-level observation and statistics of observation, *e.g.* the number of waiting vehicles on the road, historical average vehicle throughput of different phases at the intersection.
- **Action:** Decide which directions can pass.
- **Reward:** Metrics measuring the performance of urban traffic. We provide two widely-used metrics: the total number of waiting vehicles at the intersections and the average waiting time during an action interval.

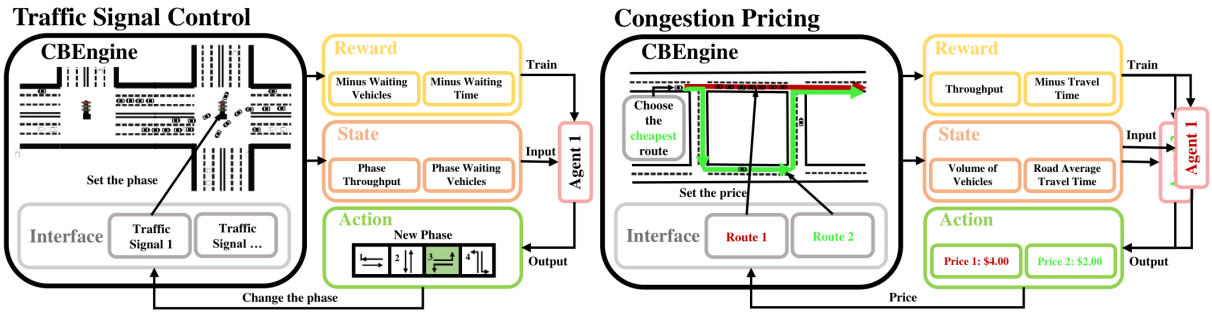

Figure 12: Illustration of two scenarios: traffic signal control and congestion pricing.

Dataset	Hangzhou		Manhattan	
Metrics	Throughput	Travel Time(s)	Throughput	Travel Time(s)
FixedTime	2184	1478.01	2894	1309.88
MaxPressure	3336	700.66	3364	805.14
SOTL	1122	305.79	137	488.62
DQN	3573	309.10	3926	375.51

Dataset	Hangzhou		Manhattan	
Metrics	Throughput	Travel Time(s)	Throughput	Travel Time(s)
No-Change	2176	1455	2911	1328
Random	3008	644.00	3459	1120.69
Deltatoll	3186	604.00	3670	960.17
EBGtoll	2803	310.18	3494	1019.85

Table 3: Performance of baseline algorithms on traffic signal control (Up) and congestion pricing (Down).

We implement several baseline algorithms to justify the plausibility of our simulation. Two metrics are used to evaluate their performances: arriving vehicle throughput and average travel time of vehicles. The results are shown in Table 3. Two transportation methods, MaxPressure [17] and Self Organized Traffic Light (SOTL) [5], show a degree of advantages in increasing throughput and reducing travel time. As a learning-based traffic control policy, DQN [12, 21] performs even better. See Appendix B for more details.

4.2 Policy 2: Congestion Pricing

Congestion pricing reroutes vehicles by dynamically assigning prices to different routes and guiding vehicles to drive on the route of the lowest price. Good pricing methods tend to allocate heavy traffic on different routes with a trade-off of the distance and the capacity, therefore enhancing traffic efficiency. Figure 12 (right) shows the setting of congestion pricing.

We then define congestion pricing as an MDP:

- **State:** Road-level observation and statistics of observation. For the observation commonly used, we have the vehicle number and the average speed of vehicles on the road.
- **Action:** Price roads in the road network.
- **Reward:** Metrics measuring the performance of urban traffic. A common reward is the average travel distance of vehicles during an interval.

Congestion pricing has been researched in the transportation field for a while yet few studies use data-driven methods. We implement two transportation-based methods, Random and Deltatoll [16], and an RL algorithm, EBGtoll [15]. More details of the experiment are given in Appendix B. Results are shown in Table 3. Random outperforms No-change which keeps original routes for vehicles. This is plausible because random rerouting averagely allocates traffic on all available routes. Deltatoll and EBGtoll behave similarly and outperform Random in both evaluation metrics.

5 CONCLUSION

In this paper, we present CBLab, a toolkit for scalable traffic simulation. CBLab provides the first simulator to support real-time simulation on large-scale cities with more than 10,000 intersections and 1,000,000 vehicles. A data tool is implemented to supply large-scale simulation data. We build an interactive environment for two traffic control policies. CBLab is the first infrastructure supporting the training of large-scale traffic control policies.

ACKNOWLEDGEMENT

This work was sponsored by National Key Research and Development Program of China under Grant No.2022YFB3904204, National Natural Science Foundation of China under Grant No. 62102246, 62272301, and Provincial Key Research and Development Program of Zhejiang under Grant No. 2021C01034.

REFERENCES

[1] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In *IEEvaluation@ACL*. 65–72. https://aclanthology.info/papers/W05-0909/w05-0909

[2] Chacha Chen, Hua Wei, Nan Xu, Guanjie Zheng, Ming Yang, Yuanhao Xiong, Kai Xu, and Zhenhui Li. 2020. Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3414–3421.

[3] 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.

[4] Seongjin Choi, Jiwon Kim, and Hwasoo Yeo. 2021. TrajGAIL: Generating urban vehicle trajectories using generative adversarial imitation learning. *Transportation Research Part C: Emerging Technologies* 128 (2021), 103091.

[5] Seung-Bae Cools, Carlos Gershenson, and Bart D'Hooghe. 2013. Self-organizing traffic lights: A realistic simulation. In *Advances in applied self-organizing systems*. Springer, 45–55.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.

[7] Mordechai Haklay and Patrick Weber. 2008. Openstreetmap: User-generated street maps. *IEEE Pervasive computing* 7, 4 (2008), 12–18.

[8] Stefan Krauß. 1998. Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics. (1998).

[9] Yang Li, Yu Shen, Wentao Zhang, Yuanwei Chen, Huaijun Jiang, Mingchao Liu, Jiawei Jiang, Jinyang Gao, Wentao Wu, Zhi Yang, et al. 2021. Openbox: A generalized black-box optimization service. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3209–3219.

[10] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. 2018. Microscopic Traffic Simulation using SUMO, In The 21st IEEE International Conference on Intelligent Transportation Systems. *IEEE Intelligent Transportation Systems Conference (ITSC)*. https://elib.dlr.de/124092/

[11] Hamid Mirzaei, Guni Sharon, Stephen Boyles, Tony Givargis, and Peter Stone. 2018. Enhanced delta-tolling: Traffic optimization via policy gradient reinforcement learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 47–52.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[13] Afshin Oroojlooy, Mohammadreza Nazari, Davood Hajinezhad, and Jorge Silva. 2020. Attendlight: Universal attention-based reinforcement learning model for traffic signal control. *Advances in Neural Information Processing Systems* 33 (2020), 4079–4090.

[14] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *ACL*. 311–318. http://www.aclweb.org/anthology/P02-1040.pdf

[15] Wei Qiu, Haipeng Chen, and Bo An. 2019. Dynamic Electronic Toll Collection via Multi-Agent Deep Reinforcement Learning with Edge-Based Graph Convolutional Networks.. In *IJCAI*. 4568–4574.

[16] Guni Sharon, Michael W Levin, Josiah P Hanna, Tarun Rambha, Stephen D Boyles, and Peter Stone. 2017. Network-wide adaptive tolling for connected and automated vehicles. *Transportation Research Part C: Emerging Technologies* 84 (2017), 142–157.

[17] Pravin Varaiya. 2013. Max pressure control of a network of signalized intersections. *Transportation Research Part C: Emerging Technologies* 36 (2013), 177–195.

[18] Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. 2019. Presslight: Learning max pressure control to coordinate traffic signals in arterial network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1290–1298.

[19] Hua Wei, Nan Xu, Huichu Zhang, Guanjie Zheng, Xinshi Zang, Chacha Chen, Weinan Zhang, Yanmin Zhu, Kai Xu, and Zhenhui Li. 2019. Colight: Learning network-level cooperation for traffic signal control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1913–1922.

[20] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. 2019. A survey on traffic signal control methods. *arXiv preprint arXiv:1904.08117* (2019).

[21] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. 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.

[22] Libing Wu, Min Wang, Dan Wu, and Jia Wu. 2021. DynSTGAT: Dynamic Spatial-Temporal Graph Attention Network for Traffic Signal Control. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2150–2159.

[23] Bingyu Xu, Yaowei Wang, Zhaozhi Wang, Huizhu Jia, and Zongqing Lu. 2021. Hierarchically and cooperatively learning traffic signal control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 669–677.

[24] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2011. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 316–324.

[25] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*. 99–108.

[26] Xinshi Zang, Huaxiu Yao, Guanjie Zheng, Nan Xu, Kai Xu, and Zhenhui Li. 2020. Metalight: Value-based meta-reinforcement learning for traffic signal control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1153–1160.

[27] 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.

[28] Huichu Zhang, Chang Liu, Weinan Zhang, Guanjie Zheng, and Yong Yu. 2020. Generalight: Improving environment generalization of traffic signal control via meta reinforcement learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1783–1792.

[29] Liang Zhang, Qiang Wu, Jun Shen, Linyuan Lü, Bo Du, and Jianqing Wu. 2022. Expression might be enough: representing pressure and demand for reinforcement learning based traffic signal control. In *International Conference on Machine Learning*. PMLR, 26645–26654.

[30] Guanjie Zheng, Yuanhao Xiong, Xinshi Zang, Jie Feng, Hua Wei, Huichu Zhang, Yong Li, Kai Xu, and Zhenhui Li. 2019. Learning phase competition for traffic signal control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1963–1972.

A KEY INFORMATION OF CBLAB

A.1 Licensing

CBLab uses the MIT license.

A.2 Code

The code of CBLab is available on GitHub.
https://github.com/CityBrainLab/CityBrainLab.git

A.3 Documentation

The documentation of CBLab is available.
https://cblab-documentation.readthedocs.io/en/latest/

A.4 Road Network Dataset

The road network dataset is available on Google Drive:
https://drive.google.com/drive/folders/1q0StZW9ERwMOKQRIT_29ZR-hhGT013J5?usp=sharing

B DETAILS OF EXPERIMENTAL SETUP

B.1 Efficiency and Scalability (Section 2.5)

Baselines Setup. In our experiment, we use the default setting of SUMO (TRACI) and Cityflow. Note that our simulator is a microscopic one. Therefore, we select two open source microscopic traffic simulators as our baselines. The routing model is not activated for all three simulators so it is not relevant if routing parallelization is used. In the experiment presented in our paper, internal links between intersections are used.

Considering that CBEngine simplifies links between intersections, a question may be raised whether the internal link has a decisively negative impact on the performance of the baseline. To answer this question, we conduct new efficiency experiments on SUMO with no internal links and the same setting otherwise. The comparison result is demonstrated in Table 4.

According to the result, removing internal links for SUMO does improve the simulation efficiency. However, SUMO's efficiency is still not considerable compared with that of CBEngine. This is because CBEngine deploys other optimization (e.g. an optimized parallelization architecture) to improve efficiency.

Datasets. The efficiency experiment is conducted on the traffic data from six cities of different scales: Nanchang, Changchun, Jinan, Shenzhen, Hangzhou, and Shanghai. We obtain the road network data from our dataset of CBData. Traffic flows are generated according to the scale of the road network. Two scalability experiments are conducted on real-world road networks from our dataset with scales close to the selected road network size ($\{10, 10^2, 10^3, 10^4, 10^{4.5}\}$ intersections). Traffic flows are generated according to the scale of the road network. All road networks and traffic flows are available in our code provided on Google Drive.
https://drive.google.com/file/d/1NHlIR_0CRlRc87SOACBJxt764Obz6ca/view?usp=sharing
We also provide a reproducing instruction to help reproduce our experimental results.
https://github.com/CityBrainLab/CityBrainLab/blob/main/CBEngine/Reproducing_Instruction.md

Computing Resources. All the experiments are conducted on a Ubuntu20.04 system with a 40-core CPU and 128GB RAM.

Hyperparameters. The number of threads is chosen as 20 to stay consistent with the cores. Note that, using fewer or more threads lead to worse efficiency for both Cityflow and CBEngine.

B.2 Learning from Real Traffic Data (Section 3.2)

In these experiments, we aim to provide demonstrations to show the possibility to use real-world traffic data to optimize the traffic simulator. Hence, there might be further room for improvement if the parameters are tuned carefully.

B.2.1 Learning to Simulate Driving.

Datasets. The road network of Shenzhen is obtained from our dataset. We obtain the traffic flow (as the input) and the observation of speed (as the ground truth) from the GPS trajectory data of cars in Shenzhen, China for one day. The data covers 123,481 trajectories and comes from personal data providers with consensus.

Optimization Details. We use OpenBox as a toolkit to search for parameters. The code can be found at https://github.com/PKU-DAIR/open-box.git under the MIT license. The start-up hyperparameters are as follows. For the maximum acceleration and deceleration, we set the default value (value where the search starts) as $2.0m/s^2$ and $5.0m/s^2$, respectively. For the speed limit, we set the default value as $11.1m/s$. The number of rounds is set as 20. We use the surrogate type *auto* and optimizer type *auto*.

B.2.2 Learning to Simulate Routing.

Datasets. We demonstrate how to learn the routing module on trajectories data of Shenzhen, China for one day. We collect the trajectories with the origin of {Latitude: $22.5405°N$, Longitude: $113.967°E$} and the destination of {Latitude: $22.6164°N$, Longitude: $113.853°E$}. The origin is the bus station of the Window of the World, a famous scenic spot in Shenzhen. The destination is a bus station on the highway from Shenzhen to Guangzhou. This origin-destination pair aggregates the most number of different routes in our dataset, while routes of other origin-destination pairs are quite unified. The total number of different routes is 22 and that of trajectories is 118.

Optimization Details. We use an RNN-based model as the trajectory generator. The code can be found at https://github.com/ben choi93/TrajGAIL.git under MIT license. We follow the setting of hyperparameters in the original paper [4] except for the iteration number since our trajectories are complicated for the generator to learn from. We set the iteration number as 100.

B.3 Traffic Signal Control and Congestion Pricing (Section 4.2 and 4.3)

The goal of these two experiments is to provide possible benchmarks for algorithms. Here, we use several typical algorithms to validate these scenarios. Providing comprehensive baseline methods comparison is out of the scope here. Hence, people are welcome to provide more advanced algorithms for these scenarios.

Dataset	Nanchang	Changchun	JiNan	Shenzhen	Hangzhou	Shanghai
	Time Cost					
SUMO (with internal links)	1239.93 (±3.58)	2091.60 (±7.84)	2151.01 (±70.64)	3103.58 (±110.08)	3199.14 (±70.87)	6173.51 (±75.27)
SUMO (no internal links)	1218.53	1987.12	2046.63	2973.50	3020.16	6083.37

Table 4: Comparison results of efficiency experiments between two setups of SUMO.

Datasets. We use two real-world datasets to validate CBScenario: Hangzhou and Manhattan. Both datasets are transformed from the traffic data at https://traffic-signal-control.github.io/#open-data sets, which serves as a widely used benchmark for traffic signal control. We use part of CBData to transform them to the format suitble for CBEngine. The goal of the experiment is to validate the plausibility of our traffic simulation. Therefore, we refer to the widely used benchmark rather than picking up novel cases not being evaluated yet.

Memory size	Value updating interval	ϵ	γ
5000	1	0.9	0.95
Learning rate	Target updating interval	ϵ_{min}	Decay of ϵ_{min}
0.005	20	0.2	0.995

Table 5: Hyperparameters of DQN in traffic signal control.

Memory size	Value updating interval	Policy learning rate
2000	1	0.001
τ	Target updating interval	Critic learning rate
0.125	10	0.0005

Table 6: Hyperparameters of EBGtoll in congestion pricing.

Hyperparameters of Traffic Signal Control. The traffic in one episode lasts for 1800 seconds. The DQN method is trained for 50 episodes and the batch size is set as 64. Other hyperparameters are listed in Table 5.

Hyperparameters of Congestion Pricing. The traffic in one episode lasts for 10800 seconds. Actions are taken every 540 seconds. We use the fixed time policy as the default traffic signal control policy. For transportation-based methods, we evaluate them in one episode. For the training of EBGtoll, the number of episodes is 200 and the batch size is set as 32. Other hyperparameters are listed in Table 6.

B.4 APIs for Traffic Control

CBEngine provides various APIs for users to develop novel traffic control policies. The functions supported are listed as follows:

- Changing the phase of a traffic signal light
- Modifying the speed limit of a road
- Changing the route of a vehicle

C CITY BRAIN CHALLENGE @ KDD CUP 2021

City Brain Challenge @ KDD CUP 2021 [1] aimed to explore an efficient decision-making solution for traffic signal control in city-level urban traffic scenarios. The challenge provided an interactive city-level traffic environment with 2,048 intersections, whose data originated from the real urban traffic in Nanchang, China. Players of the challenge were in charge of the urban traffic signals and tried to improve the number of served vehicles and decrease the traffic delay. Each team designed and tuned their plan based on a given traffic flow, while their submitted plan would be then evaluated in another traffic flow. See the documentation [2] for the challenge for more information.

City Brain Challenge @ KDD CUP 2021 had 1,156 teams of participants. The team *IntelligentLight* from Shanghai Jiao Tong University finally got first place. *IntelligentLight* innovated intelligent traffic signal control algorithms that can improve the traffic efficiency of a city by at least 31%.

The original version of CBLab supported the interactive city-level traffic environment in the City Brain Challenge @ KDD CUP 2021. Specifically, we built the basic interactive traffic environment on the early version of the simulator CBEngine. After the City Brain Challenge, we introduced more designs to improve the efficiency and scalability of CBEngine, developed the data tool CBData, and expanded the basic interactive environment into CBScenario.

D DETAILS OF ROAD NETWORK DATASET

The road network dataset in CBData includes raw road networks of 100 main cities around the world. The list and the range of these road networks are given on GitHub:
https://github.com/CityBrainLab/CityBrainLab/blob/main/CBData/citylist.csv. We obtain the data from OpenStreetMap, an open-source map database. Note that the bounding boxes of these road networks have to be a rectangle thus not strictly consistent with the boundary of the city. These road networks can be transformed into road network inputs for traffic simulation with our pipeline in CBData.

[1] https://kdd.org/kdd2021/
[2] https://kddcup2021-citybrainchallenge.readthedocs.io/en/latest/city-brain-challenge.html