CBLAB: SCALABLE TRAFFIC SIMULATION WITH ENRICHED DATA SUPPORTING

Anonymous authors
Paper under double-blind review

ABSTRACT

Traffic simulation provides interactive data for the optimization of traffic 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 is consist 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 several baseline methods for two scenarios of traffic policies respectively, with which traffic 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 policy optimization in large-scale urban scenarios. The code is available on GitHub: `https://github.com/CityBrainLab/CityBrainLab.git`.

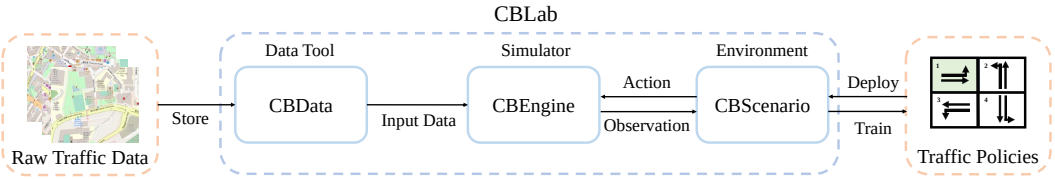

Figure 1: An overview of CBLab.

1 INTRODUCTION

Well-crafted traffic 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 policies according to real-time traffic data Haarnoja et al. (2018); Wei et al. (2018; 2019a); Zheng et al. (2019); Zhang et al. (2020); Chen et al. (2020); Zang et al. (2020); Oroojlooy et al. (2020); Wu et al. (2021); Zhang et al. (2022); Mirzaei et al. (2018); Qiu et al. (2019). 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 policies to interact with. These simulators Lopez et al. (2018); Zhang et al. (2019) simulate the microscopic evolution of the urban traffic. For each time step, they describe the traffic state, obtain a traffic action from the decision of traffic policies and make it happen in the simulation. Traffic policies can then learn from how the traffic evolves under certain actions and improve decision making.

While existing traffic simulators help hatch various traffic 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 mechanisms, 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 main cities in the world is now completed roughly and being refined, there is an absence of infrastructure for convenient 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 Wei et al. (2019c;b); Zheng et al. (2019); Xu et al. (2021) whose scales are often dozens of intersections (e.g. 4x3 or 4x4) - much less 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 policy environment CBScenario. CBEngine is of high efficiency which benefits from well-designed parallelization. With ordinary computing hardware, CBEngine 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. 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 further implement CBScenario as an environment for two common traffic policies: traffic signal control and congestion pricing. Users can design, develop, and train traffic policies in the framework of CBScenario. To the best of our knowledge, we are the first to provide infrastructure for large-scale traffic policy optimization.

Our contribution can be summarized as follows.

- We develop a scalable traffic simulator CBEngine which supports city-level microscopic traffic simulation for the first time.
- We develop a data tool CBData to provide enriched input data for large-scale traffic simulation.
- Based on CBEngine and CBData, we implement an interactive environment CBScenario for two common traffic policies under a large-scale setting.

2 CBENGINE: CITY-SCALE TRAFFIC SIMULATION ENGINE

In this section, we introduce the design of CBEngine. We demonstrate the modeling of urban traffic and conduct extensive experiments to show the superiority of our simulator in efficiency and scalability, compared to existing simulators.

2.1 THE TRAFFIC MODEL OF CBENGINE

The objective of the simulator is to describe the interaction between the road network and the traffic flow (vehicles in the traffic). As shown in Figure 2, the road network involves the interaction through one of its components: the 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.

During this process, traffic signals and vehicles interact as follows. Each traffic signal may change the signal phase (controlling the traffic direction allowed to move) as time changes. Vehicles will move on and accelerate or decelerate according to the current speed and acceleration, respectively. Meanwhile, they may change their routes or acceleration as they move, based on the road or traffic signal constraints, and the movement of other nearby vehicles. For each time step, traffic signals, and vehicles observe from their views and make decisions on their next action. The simulator will then conduct these actions. This forwards the system from state s_t at time t to a new state $s_{t+\delta t}$ at time $t + \delta t$. We can conclude one step with the following formula, where s and a represent the state and

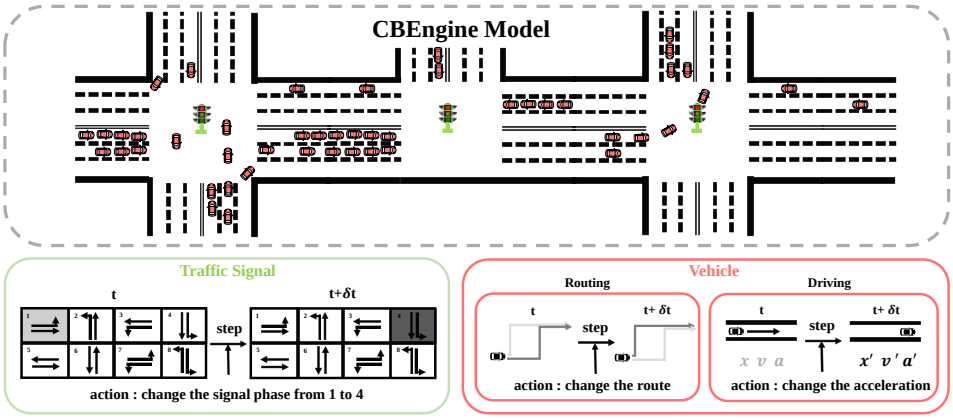

Figure 2: The Traffic Model of CBEngine

action for the whole system (including traffic signals and vehicles), correspondingly.

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

2.2 Customization of Vehicle Behavior Models

Behaviors of vehicles can be different in different traffic. For example, drivers at night tend to conduct sudden stops more frequently Lenné et al. (1997). Towards simulating the behaviors of vehicles, researchers have proposed various models Yuan et al. (2010; 2011). These models can be summarized into two categories: driving models and routing models. Driving models determine the acceleration in the view of drivers, corresponding to when and how they step on the accelerator and the brake. Routing models determine the routing choice for vehicles. Given the origin and the destination, there may be dozens of feasible route candidates. Routing models use information from the driver's view to choose a route from candidates.

CBEngine provides a customization API for users to design and implement their own driving and routing models. As described in Figure 2, both the routing model and the driving model are black boxes with definite inputs and outputs. We modularize two models as an independent C++ class in our implementation, respectively. Users can customize two models by only editing the class without modification to other parts of CBEngine. Our documentation in Appendix A provides detailed instructions. To the best of our knowledge, CBEngine is the first traffic simulator to support easy-to-use customization of vehicle behavior models.

2.3 Efficiency and Scalability

The major bottleneck that prevents existing simulators from supporting city-wide traffic policy training is their limit in efficiency and scalability, of which extensive efforts have been made in CBEngine for improvement. In the following part, we first illustrate a parallel design adapted by CBEngine that greatly promotes the efficiency of CBEngine. Then, we conduct extensive experiments to validate the efficiency and scalability of CBEngine. More implementation details of CBEngine are discussed in Appendix C.

Parallel Design in Computing the Vehicle Behavior The future state in the simulation is determined by the present state and the action. Therefore, computing the action is the main computational job for traffic simulators. 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.

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 necessary 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 the collected information. Existing simulators parallelize these two

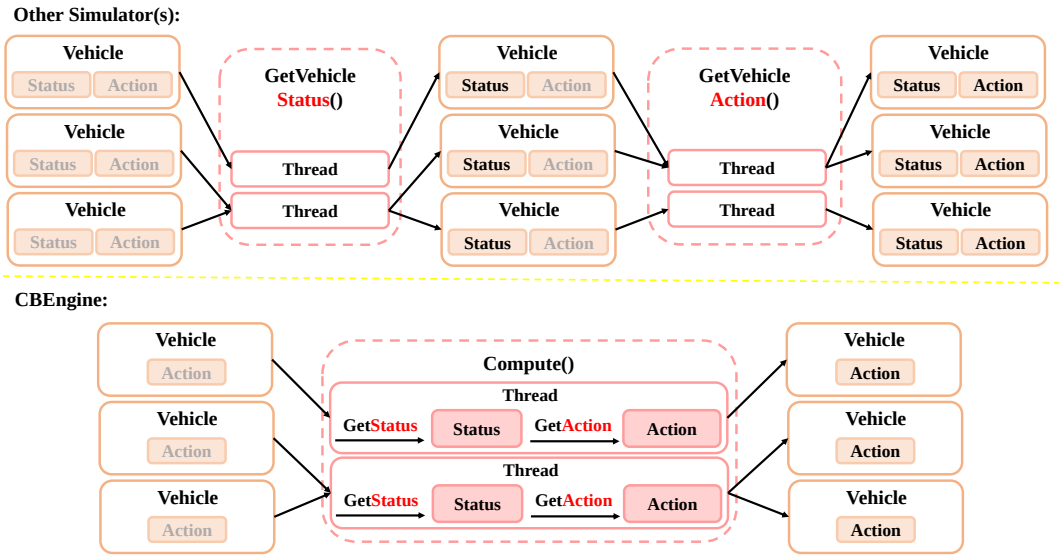

Figure 3: Comparison between CBEngine and other simulators on parallelization design of computing actions for vehicles. The gray data member indicates it is not updated, while the black one is updated.

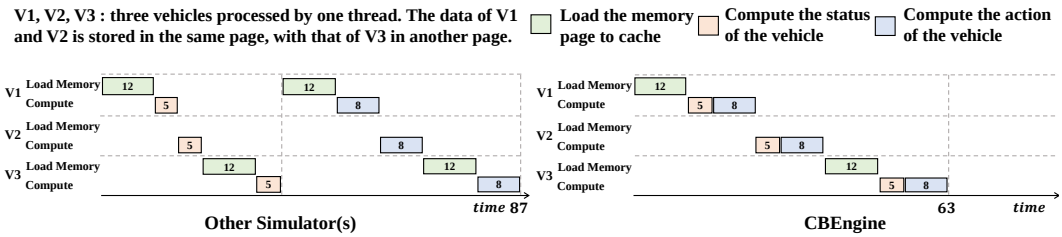

Figure 4: A schedule case of computing vehicle action. The length of the bar indicates the time the corresponding task consumes.

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 load 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 load 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.

Experimental Setup To evaluate the efficiency and scalability of our simulator, we compare it with two widely-used open-source microscopic traffic simulators, SUMO Lopez et al. (2018) and Cityflow Zhang et al. (2019). 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.

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.

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 Haklay & Weber (2008). Traffic 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.

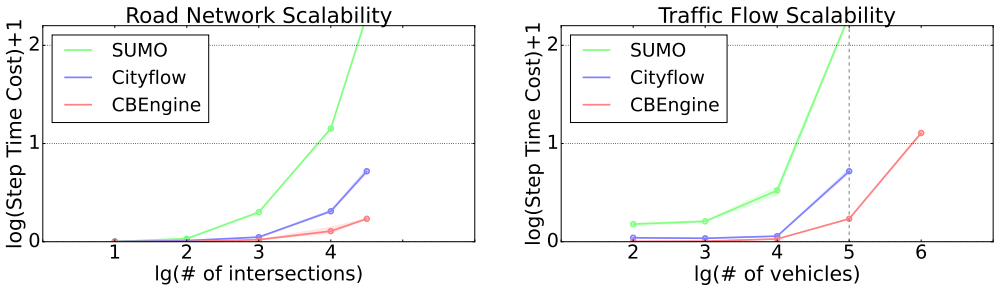

Figure 5: Comparison results of two experiments of scalability.

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 selected. 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 Lopez et al. (2018) and Cityflow Zhang et al. (2019) 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

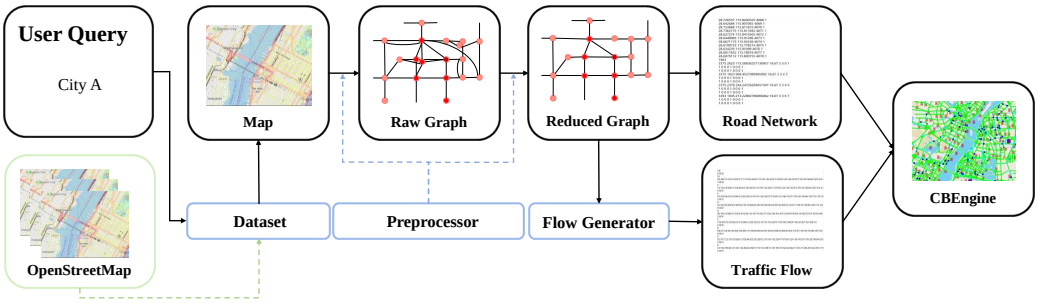

Figure 6: The simulation input data supporting pipeline of CBData.

$\{10^2, 10^3, 10^4, 10^5, 10^6\}$ on-way vehicles. For each setting, we repeat the experiment three times. We report the time cost of a single step for baselines and our simulator as well as the range.

The results are visualized in Figure 5 (right). CBEngine outperforms two baselines in time cost 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 Lopez et al. (2018) and Cityflow Zhang et al. (2019) are 8.9111 seconds and 1.1058 seconds, respectively. Specifically, two baselines are not able to run the case with 1,000,000 vehicles. We discuss the reason in Appendix C.

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 6. 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 Haklay & Weber (2008).

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 6, the pipeline consists of three kits: the dataset, the preprocessor, and the flow generator.

Dataset: We obtain the map data of the whole world from OpenStreetMap Haklay & Weber (2008). 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). 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 averagely as possible. The default route for each pair of the origin and the destination is the shortest path. However, the route can be changed by the routing model of the traffic simulator when running the simulation.

3.2 PIPELINE: LEARNING TO SIMULATE FROM TRAFFIC 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 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, a self-adaptive driving model similar to IDM Yuan et al. (2011). We 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 Li et al. (2021) as the optimization tool. Note that users can use any other optimization toolkit according to their needs. 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, China are used to search the parameters. The observation interval is 1 minute. For every 20 minutes, the traffic distribution will change. We expect that acceleration parameters can be continuously learned when the data changes.

The loss curves of the learning process of these two avenues are displayed in Figure 7. 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.

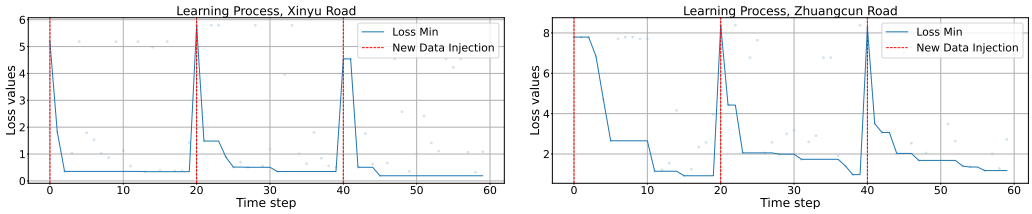

Figure 7: Loss curves during learning the driving module on two roads.

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 a route for certain origin and destination based on real trajectories. However, the studies in this field are limited and we exploit the Recurrent Neural Networks (RNNs) as our route generator Choi et al. (2021).

We use part of a vehicle trajectory dataset in Shenzhen, China. This dataset contains 22 different routes in total. We train the RNN Choi et al. (2021) to learn the distribution of routes and conduct routing for vehicles in the simulator. Figure 8 shows the result of loss curves and two trajectory generating metrics, BLEU Papineni et al. (2002) and METEOR Banerjee & Lavie (2005). 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.

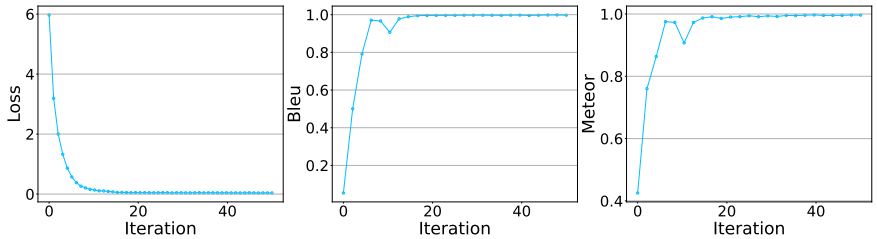

Figure 8: Curves of loss, BLEU, and METEOR during learning the routing module.

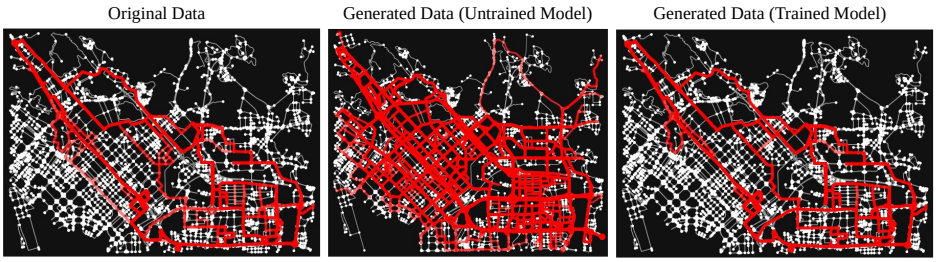

Figure 9: Visualization of the routing result. The deeper red is, the more frequent the route is picked.

4 CBSCENARIO: ENVIRONMENT FOR LARGE-SCALE TRAFFIC POLICIES

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

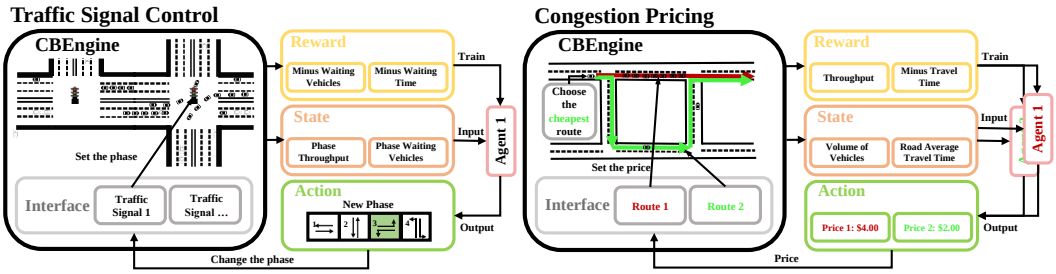

Figure 10: Illustration of two scenarios: traffic signal control and congestion pricing.

4.1 TRAFFIC POLICY 1: TRAFFIC SIGNAL CONTROL

The traffic signal control problem Wei et al. (2019c) 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 10 (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.

We implement several baseline algorithms to justify the plausibility of our simulation. We consider two metrics to evaluate the performance of these algorithms: arriving vehicle throughput and average travel time of vehicles. More details of the experiment are given in Appendix B. The results are shown in Table 2. Two transportation methods, MaxPressure Varaiya (2013) and Self Organized Traffic Light (SOTL) Cools et al. (2013), show a degree of advantages in increasing throughput and reducing travel time. However, as a learning based traffic policy, DQN Mnih et al. (2015); Wei et al. (2018) performs even better.

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

Table 2: Performance of baseline algorithms on traffic signal control.

4.2 Traffic 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 10 (right) shows the setting of congestion pricing.

Similar to traffic signal control, we can define congestion pricing as an MDP:

- **State:** Road-level observation and statistics of observation. For the observation common 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 action 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 Sharon et al. (2017), and an RL algorithm, EBGtoll Qiu et al. (2019). 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 allocates traffic averagely on all available routes. Deltatoll and EBGtoll behave similarly and outperform Random in both evaluation metrics.

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 congestion pricing.

5 Conclusion

In this paper, we present CBLab, a toolkit for scalable traffic simulation. CBLab provides the first simulator to support real-time simulation of large-scale cities with more than 10,000 intersections. A data tool is implemented to supply large-scale input data for the simulation. An interactive environment and benchmarks for two common traffic policies have been built and prove that the simulation is plausible. To the best of our knowledge, CBLab is the first infrastructure supporting the training of large-scale traffic policies.

REFERENCES

Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *IEEvaluation@ACL*, pp. 65–72, 2005. URL https://aclanthology.info/papers/W05-0909/w05-0909.

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

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

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

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

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

Michael G. Lenné, Thomas J. Triggs, and Jennifer R. Redman. Time of day variations in driving performance. *Accident Analysis & Prevention*, 29(4):431–437, 1997.

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

Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL https://elib.dlr.de/124092/.

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

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. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

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

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pp. 311–318, 2002. URL http://www.aclweb.org/anthology/P02-1040.pdf.

Wei Qiu, Haipeng Chen, and Bo An. Dynamic electronic toll collection via multi-agent deep reinforcement learning with edge-based graph convolutional networks. In *IJCAI*, pp. 4568–4574, 2019.

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

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

Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2496–2505, 2018.

Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. 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*, pp. 1290–1298, 2019a.

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

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

Libing Wu, Min Wang, Dan Wu, and Jia Wu. Dynstgat: Dynamic spatial-temporal graph attention network for traffic signal control. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 2150–2159, 2021.

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

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

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

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

Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Li. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In *The world wide web conference*, pp. 3620–3624, 2019.

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

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

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

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/1IyTvWprOA1R_6PVkuh7v9R4xrHcZAmYT?usp=sharing

A.5 LIMITATIONS & FUTURE WORKS

CBLab aims to provide efficient and data-driven traffic simulation for research in traffic policies. Although we provide interfaces for using data to optimize the simulator, we cannot obtain all real traffic data from different cities to conduct complete optimization on our simulator. We will invite future contributors to make more traffic data compatible with these interfaces so that they can be used to optimize the simulator and run the simulation.

Moreover, we plan to implement more scenarios for traffic policies, $e.g.$ traffic restriction. Also, we aim to include more powerful algorithms in these scenarios, $e.g.$ Intellilight and FRAP for traffic signal control. It is our hope that CBLab can serve as an online benchmark for various scenarios with state-of-the-art algorithms in addition to supporting powerful traffic simulation.

A.6 POTENTIAL NEGATIVE SOCIAL IMPACTS

All traffic simulators will suffer from the gap between simulated and simulated observations. This gap may lead to biased observations and traffic policies based on the simulation. CBLab releases the first traffic simulator open for users to optimize so that it tends to perform as the real traffic. However, the traffic patterns from different places are different. It is worth considering how to ensure the reality of our traffic simulation when simulating the traffic from different places. Hence, we are collecting more traffic data and studying the common patterns among the data, with the hope of trying to resolve this problem.

B DETAILS OF EXPERIMENTAL SETUP

B.1 EFFICIENCY AND SCALABILITY (SECTION 2.3)

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.

Dataset	Nanchang	Changchun	JiNan	Shenzhen	Hangzhou	Shanghai
	Time Cost					
SUMO (with internal links)	1239.93 ($\pm$3.58)	2091.60 ($\pm$7.84)	2151.01 ($\pm$70.64)	3103.58 ($\pm$110.08)	3199.14 ($\pm$70.87)	6173.51 ($\pm$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 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/drive/folders/1e8wjEYFnDXluHaOxyAzJOOknNvJPZ4_r?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 number of used cores. Note that, using fewer or more threads will lead to worse efficiency for both Cityflow and CBEngine.

Error Bars Error bars of the experiment are shown in Table 1 and Figure 3.

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. We are working on releasing a processed version by removing sensitive data.

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/benchoi93/TrajGAIL.git` under MIT license. We follow the setting of hyperparameters in the original paper Choi et al. (2021) 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 only 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.

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-datasets`, 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.

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.

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

Table 5: Hyperparameters of DQN in traffic signal control.

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.

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

Table 6: Hyperparameters of EBGtoll in congestion pricing.

C IMPLEMENTATION DETAILS OF CBENGINE

Except for the parallel design mentioned in Section 2.3, two mechanisms of CBEngine also contribute to its efficiency.

Lane Changing in Driving Model Lane changing is a driving action. Drivers may change the lane if they feel the current lane is too congested. However, lane changing is hard to simulate in the traffic simulation. Cityflow, 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 to. 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 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. We also discuss the impact of intersection links on our baselines in Appendix B. To the best of our knowledge, our design avoids all such deadlocks in practice.

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 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. Here, we visualize six road networks used in our experiment for example to give a scratch of our road network data in Figure 11.

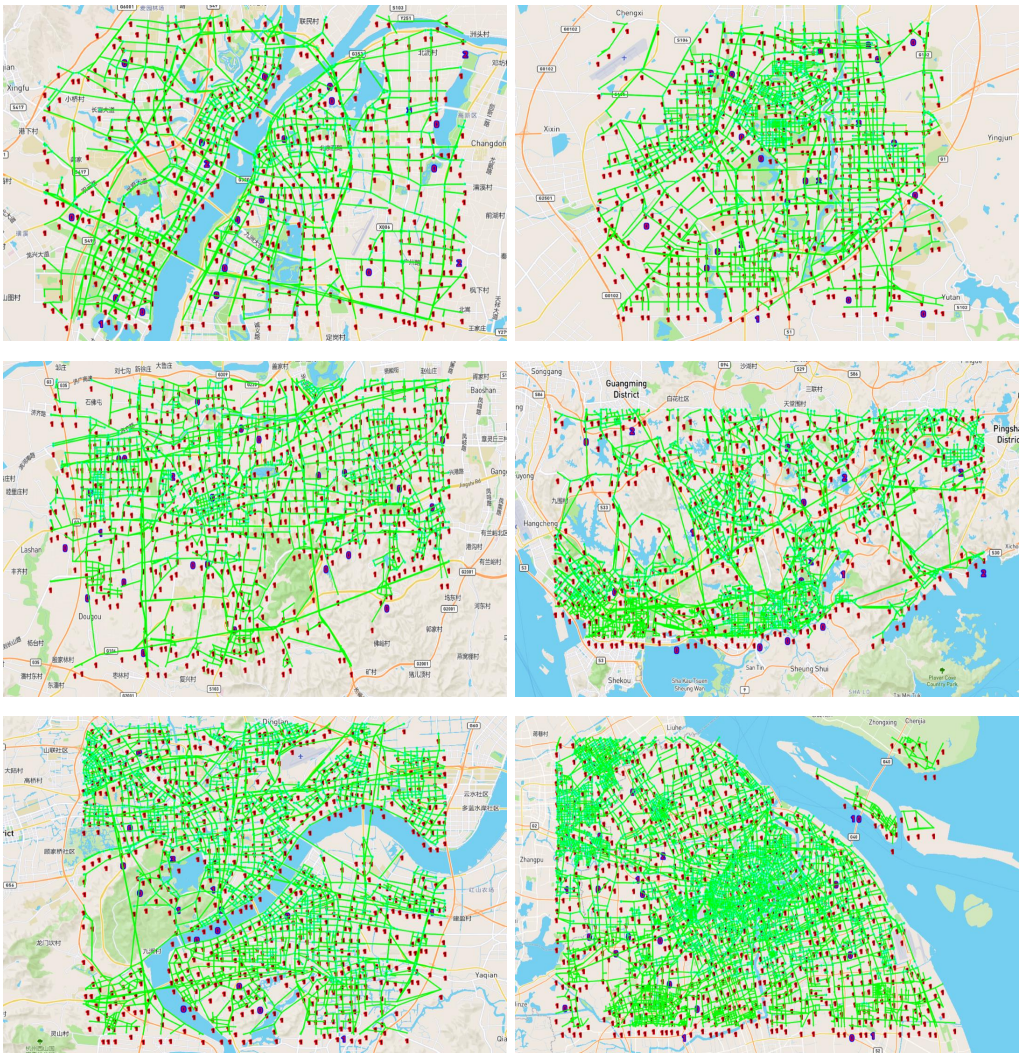

Figure 11: Visualization of road networks of six cities used in our experiment: Nanchang (top left), Changchun (top right), JiNan (middle left), Shenzhen (middle right), Hangzhou (bottom left), Shanghai (bottom right).