Machine Learning
https://doi.org/10.1007/s10994-023-06412-y

Libsignal: an open library for traffic signal control

Hao Mei[1,3] · Xiaoliang Lei[2] · Longchao Da[1,3] · Bin Shi[2] · Hua Wei[1,3]

Received: 9 March 2023 / Revised: 3 August 2023 / Accepted: 3 October 2023
© The Author(s), under exclusive licence to Springer Science+Business Media LLC, part of Springer Nature 2023

Abstract

This paper introduces a library for cross-simulator comparison of reinforcement learning models in traffic signal control tasks. This library is developed to implement recent state-of-the-art reinforcement learning models with extensible interfaces and unified cross-simulator evaluation metrics. It supports commonly-used simulators in traffic signal control tasks, including Simulation of Urban MObility(SUMO) and CityFlow, and multiple benchmark datasets for fair comparisons. We conducted experiments to validate our implementation of the models and to calibrate the simulators so that the experiments from one simulator could be referential to the other. Based on the validated models and calibrated environments, this paper compares and reports the performance of current state-of-the-art RL algorithms across different datasets and simulators. This is the first time that these methods have been compared fairly under the same datasets with different simulators.

Keywords Reinforcement learning · Traffic signal control · Benchmark and dataset · Evaluation

Editors: Emma Brunskill, Minmin Chen, Omer Gottesman, Lihong Li, Yuxi Li, Yao Liu, Zonging Lu, Niranjani Prasad, Zhiwei Qin, Csaba Szepesvari, Matthew Taylor

Hao Mei, Xiaoliang Lei have contributed equally to this work.

✉ Bin Shi
shibin@xjtu.edu.cn

✉ Hua Wei
hua.wei@asu.edu

Hao Mei
hmei7@asu.edu

Xiaoliang Lei
shawlenleo@stu.xjtu.edu.cn

Longchao Da
longchao@asu.edu

[1] Department of Informatics, New Jersey Institute of Technology, Newark, NJ, USA

[2] School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, People's Republic of China

[3] School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, USA

Published online: 28 November 2023

1 Introduction

Traffic signals coordinate the traffic movements at the intersection, and a smart traffic signal control algorithm is the key to transportation efficiency. Traffic signal control remains an active research topic because of the high complexity of the problem. The traffic situations are highly dynamic and thus require traffic signal plans to adjust to different situations. People have recently started investigating reinforcement learning (RL) techniques for traffic signal control. Several studies have shown the superior performance of RL techniques over traditional transportation approaches (Wei et al., 2018, 2019; Xu et al., 2021; Oroojlooy et al., 2020; Ma and Wu, 2020). The most significant advantage of RL is that it directly learns how to take the next actions by observing the feedback from the environment after previous actions.

In literature, a number of traffic signal control methods have been proposed (Yau et al., 2017; Wei et al., 2019), and it has attracted much attention to facilitate the implementation or use of these proposed methods. However, as shown in Table 1, current methods are distributed among different simulators and datasets. As we will show later in this paper, datasets, simulators, and even evaluation metrics vary the performance of the same algorithm. In addition, reinforcement learning is also sensitive to hyperparameters. All these make it difficult for new traffic signal control methods to ensure effective and uniform improvement. Therefore, there is an urgent need for a cross-platform, unified process with an extensible code base that supports multiple models.

This paper presents a unified, flexible, and comprehensive traffic signal control library named *LibSignal*. Our library is implemented based on PyTorch and includes all the necessary steps or components related to traffic signal control into a systematic pipeline. We consider two mainstream simulators, SUMO and CityFlow, and provide various datasets, models, and utilities to support data preparation, environment calibration, model instantiation, and performance evaluation for the two kinds of simulators.

Contributions: To the best of our knowledge, *LibSignal* is the first open-source library that provides benchmarking results for traffic signal control methods across various datasets and simulators. The main features of *LibSignal* can be summarized in three aspects:

- Unified: *LibSignal* builds a systematic pipeline to implement, use and evaluate traffic signal control models in a unified platform. We design cross-simulator data configuration, unified model instantiation interfaces, and standardized evaluation procedures.
- Comprehensive: 10 models covering two traffic simulators have been reproduced to form a comprehensive model warehouse. Meanwhile, *LibSignal* collects 9 commonly used datasets from different sources, makes them compatible with both simulators, and implements a series of widely used evaluation metrics and strategies for performance evaluation.
- Extensible: *LibSignal* enables a modular design of different components, allowing users to insert customized components into the library flexibly. It also has an OpenAI Gym interface (Brockman et al., 2016) which allows easy deployment of standard RL algorithms.

What *LibSignal* isn't: Despite the ability to train and test across different simulators, *LibSignal* does not claim the performances of the same model on different simulators are identical. There are some differences in internal mechanisms between different simulators, for example, vehicle maneuver behaviors, and our emphasis is on the relative performance

Table 1 Summary of the state-of-the-art models (a partial list), sorted by citations from Google Scholar by 2022/06/08

Method	Venue	Cite	Simulator	Dataset (* means open accessed)	Main metrics
IntelliLight (Wei et al., 2018)	KDD 2018	321	SUMO	Jinan 1 x 1	Travel time, speed, queue length, approximated delay
IDQN (Zheng et al., 2019)	arXiv 2019	53	SUMO	LA 1 x 4*, Jinan 1 x 1, Hangzhou 1 x 1	Travel time
MAPG (Chu et al., 2019)	TITS 2019	308	SUMO	Grid 4 x 4, monaco*	Approximated delay, queue length
CoLight (Wei et al., 2019)	CIKM 2019	116	CityFlow	Hangzhou 4 x 4*, Jinan 3 x 4*, Manhattan 3 x 27*	Travel time
MPLight (Chen et al., 2020)	AAAI 2020	98	CityFlow	Grid 4 x 4, manhattan	Travel time, throughput
PressLight (Wei et al., 2019)	KDD 2019	93	CityFlow	Jinan 1 x 3, State College*, Mahattan*	Travel time
FRAP (Zheng et al., 2019)	CIKM 2019	67	CityFlow	Hangzhou 1 x 1*, Atlanta 1 x 5*	Travel time
MetaLight (Zang et al., 2020)	AAAI 2020	43	CityFlow	Hangzhou 1 x 1*, Atlanta 1 x 5*, Hangzhou 4 x 4*, Jinan 3 x 4*	Travel time
DemoLight (Xiong et al., 2019)	CIKM 2019	22	CityFlow	Hangzhou 1 x 1*	Travel time
FMA2C (Ma and Wu, 2020)	AAMAS 2020	16	SUMO	4 x 4 Grid, Monaco*	Queue length, throughput, delay
TPG (Rizzo et al., 2019)	KDD 2019	15	SUMO	Roundabout	Queue length, waiting time, throughput, speed
AttendLight (Oroojlooy et al., 2020)	NeurIPS 2020	12	CityFlow	Hangzhou 4 x 4*, Atlanta 1 x 5*	Travel time
HiLight (Xu et al., 2021)	AAAI 2021	12	CityFlow	Hangzhou 4 x 4*, Jinan 3 x 4*, Manhattan 3 x 27*, Shenzhen*	Travel time, throughput
IG-RL (Devailly et al., 2021)	TITS 2021	11	SUMO	Manhattan	Approximated delay
ExplainPG (Rizzo et al., 2019)	ITSC 2019	11	SUMO	Roundabout	Waiting time, throughput
RACS-R (Wang et al., 2021)	TITS 2021	6	SUMO	Monaco*	Waiting time, queue length
GeneraLight (Zhang et al., 2020)	CIKM 2020	4	CityFlow	Hangzhou 1 x 1*, Atlanta 1 x 5*, Hangzhou 4 x 4*	Travel time
OP-TSC (Yen et al., 2020)	ITSC 2020	4	SUMO	Synthetic	Delay
DFC (Raeis & Leon-Garcia, 2021)	ITSC 2021	2	SUMO	Synthetic	Waiting time
DynSTGAT (Wu et al., 2021)	CIKM 2021	0	CityFlow	Hangzhou 4 x 4*, Jinan 3 x 4*, Grid 4 x 4*	Travel time, throughput
EMV (Cao et al., 2022)	TITS 2022	0	SUMO	Hangzhou 4 x 4*	Waiting time, queue length

The full list can be found in https://darl-libsignal.github.io/.

of compatible policies we provide. This makes *LibSignal* a possible testbed for Sim-to-Real transfer (Zhao et al., 2020; Peng et al., 2018), which is not covered by this paper.

2 Background

2.1 Reinforcement learning for traffic signal control

Problem formulation We now introduce the general setting of the RL-based traffic signal control problem, in which an RL agent or several RL agents control the traffic signals. The environment is the traffic conditions on the roads, and the agents control the traffic signals' phases. At each time step t, a description of the environment (e.g., signal phase, waiting time of cars, the queue length of cars, and positions of cars) will be generated as the state s^t. Then, the agents will predict the next actions a^t to take that maximize the expected return, where the action of a single intersection could be changing to a certain phase. Finally, the actions a^t will be executed in the environment, and a reward r^t will be generated, where the reward could be defined on the traffic conditions of the intersections.

Basic components of RL-based traffic signal control A key question for RL is how to formulate the RL setting, i.e., the reward, state, and action definition. For more discussions on the reward, state, and action, we refer interested readers to Yau et al. (2017); Rasheed et al. (2020); Wei et al. (2021). There are three main components to formulate the problem under the framework of RL:

- Reward design. As RL is learning to maximize a numerical reward, the choice of reward determines the direction of learning. A typical reward definition for traffic signal control is one factor or a weighted linear combination of several components, such as queue length, waiting time, and delay.
- State design. The state captures the situation on the road and converts it to values. Thus the choice of states should sufficiently describe the environment. For example, the state features queue length, the number of cars, waiting time, and the current traffic signal phase. Images of vehicles' positions on the roads can also be considered in the state.
- Selection of action scheme. Different action schemes also have influences on the performance of traffic signal control strategies. For example, if the action of an agent is *acyclic*, *i.e.*, "which phase to change to", the traffic signal will be more flexible than a *cyclic* action, *i.e.*, "keep current phase or change to the next phase in a cycle".

2.2 Difficulties in evaluation

In practice, the evaluation of traffic signal control methods could be largely influenced by simulation settings, including the evaluation metrics and simulation environments.

Evaluation metrics Various measures have been proposed to quantify the efficiency of the intersection from different perspectives, including the average delay of vehicles, the average queue length in the road network, the average travel time of all vehicles, and the throughput of the road network. Signal induced *delay* is another widely used metric, and previous work suggested real-time approximation as the difference between the vehicle's current speed and the maximum speed limit over all vehicles. But as we will show in Sect. 4.2, this approximated delay is not reflecting the actual delay. *Queue length* is another mainly used metric (Wei et al., 2018), while different definitions of a "queuing" state of a vehicle could largely influence the performance of the same method. In comparison, *travel time* and *throughput* are more robust to ad-hoc definitions and approximations. As we will show later, with the same experimental setting, the performance of the same method could be different under different metric, and we aim to provide as comprehensive and flexible metrics as possible in this paper to benchmark methods with a comprehensive view.

Simulation environments Since deploying and testing traffic signal control strategies in the real world involve high cost and intensive labor, simulation is a valuable alternative before actual implementation. Different choices of simulator could lead to different evaluation performances.

Currently, there are two representative open-source microscopic simulators: *Simulation of Urban MObility (SUMO)*[1] Lopez et al. (2018) and *CityFlow*[2] Zhang et al. (2019). SUMO is widely accepted in the transportation community and is a reasonable testbed choice. Compared with SUMO, CityFlow is a simulator optimized for reinforcement learning with faster simulation, while it is not widely used in the transportation field yet.

Because of these different simulation environments, methods adopted by different simulators in their original papers are hard to evaluate. As we will show later, methods perform differently under different well-calibrated simulators, and the efficiency of the training process is also different under different simulators. For the first time, this paper compares the performances of the same model under the same traffic datasets under different simulators.

2.3 Existing libraries and tools

Reinforcement Learning for Traffic Signal Control 2022 is an open-source library that provides a bunch of RL-based traffic signal control methods with traffic datasets *only* on CityFlow (Zhang et al., 2019). Flow (Kheterpal et al., 2018) and RESCO (Ault & Sharon, 2021) are reinforcement learning frameworks that can support the design and experimentation of traffic signal control methods *only* on SUMO (Lopez et al., 2018). TSLib (Tran et al., 2021) is another library that could work under both SUMO and CityFlow, yet it has limited extensibility: (1) it is challenging to deploy standard RL algorithms since it does not have an OpenAI Gym interface; (2) there is no benchmarking dataset that works across both simulators, which makes it challenging to help determine which algorithm results in state-of-the-art performance.

3 *LibSignal* toolkit

We propose *LibSignal* library integrating different influential traffic flow simulators and denote it as a standard RL traffic control testbed. The primary purposes of this standard testbed are:

1. Provide a converter to transform configurations, including road networks and traffic flow files across different simulators, enabling comparisons between different algorithms originally conducted in different simulators.
2. Standardized implementation of state-of-the-art RL-based and traditional traffic control algorithms.

[1] http://sumo.sourceforge.net..

[2] https://cityflow-project.github.io/.

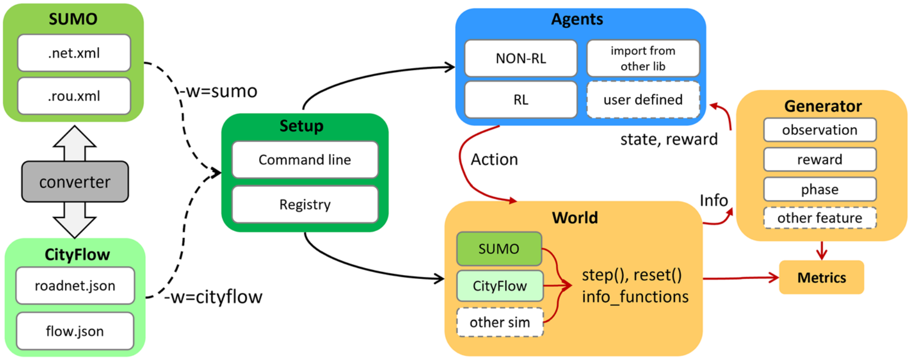


Fig. 1 Overall framework of *LibSignal*

3. A cross-simulator environment provides highly unified functions to interact with different baselines or user-defined models and supports performance comparisons among them.

3.1 Traffic signal control environment

LibSignal is open source and free to use/modify under the GNU General Public License 3. The code is built on top of GeneraLight (Zhang et al., 2020) and is available on GitHub at https://darl-libsignal.github.io/. The embedded traffic datasets are distributed with their licenses from Reinforcement Learning for Traffic Signal Control 2022 and Ault & Sharon (2021), whose licenses are under the GNU General Public License 3. SUMO is licensed under the EPL 2.0, and CityFlow is under Apache 2.0. The overall framework of *LibSignal* is presented in Fig. 1, and the implementation details will be introduced in the following sections.

Tasks We consider each traffic scenario a distinguishable task in the traffic signal control environment. The traffic scenario consists of traffic network configurations and pre-recorded traffic flows. In our library, we include nine traffic configurations, and multiple traffic flows under each traffic configuration. Details can be referred to Sect. 3.2. In each task, the goal is to minimize the average travel time of all vehicles entering and leaving the road network.

Agents *LibSignal* provides different configurable observation spaces and reward functions. This section provides a general description of observation space, actions space, and reward information.

- Observation space: The observation could be both lane-level and road-level features of each intersection. We provide the state of each lane and concatenate ones that belong to the same intersection as the observation. Since the intersection configuration could be different, the observation space varies based on the number of lanes at the intersection, and the state of each lane contains vehicle information. We provide different information that can be used as State in our library; each combination could

Table 2 Information returned from `World` class and used as State

State information	Description
lane_count	Number of vehicles on each lane
lane_waiting_count	Number of vehicles stopped each lane
lane_waiting_time_count	Waiting time of vehicles on each lane. Since their last action
lane_delay	The delay of vehicles on each lane
pressure	Difference of vehicle density between the in-coming lane and the out-going lane
vehicle_map	An image representation of vehicles' position in this intersection
passed_count	Number of vehicles that passed the intersection during time interval Δt after the last action
passed_time_count	Total time spent on approaching lanes of vehicles that passed the intersection during time interval Δt after the last action
cur_phase (index)	Combination of movement signals
cur_phase (one hot)	One hot encoded index of phase

be used as the observations. The state dimension should be the number of lanes in each intersection or the number of chosen information times the number of lanes. Road-level features are the aggregation of lane-level features which consists of the same road. The dimension should be the number of roads in each intersection or the number of chosen information times the number of roads. Details of information are in Table 2.

- Action Space: We take phase, which is a combination of non-conflict movement signals as action and does not have an assumption on a cycle-based signal plan. The action space is a predefined set of phases. After taking actions, the environment will execute a given period Δt called action interval. To make agent more flexible, we also provide `RLFXAgent` class which control phase and duration of the phase at the same time. Details illustration of phase can be found in Fig. 2.
- Reward information: Reward information is specific to each intersection. They are aggregated lane-level information over given action intervals at each intersection. The reward is a scale on each intersection. Details of reward information are shown in Table 3.
- Episode Dynamics: We set each task episode the same length of time, which is 3600 s in our setting. The simulation step length in different simulation environments is 1 s. And each action interval is default set as 10 s, though it could be configured manually.

3.2 Data preparation

To enable fair comparison, *LibSignal* preprocesses comprehensive datasets making it runnable under different simulators. Users can easily choose to specify datasets and simulators for their experiments.

Comprehensive datasets By surveying the recent literatures on traffic prediction, we selected 225 representative or survey papers (more details can be found in Table 1). We collected all the open datasets used by these papers and kept 9 datasets according to the factors of popularity, which can cover 65% papers of our reproduced model list and all the

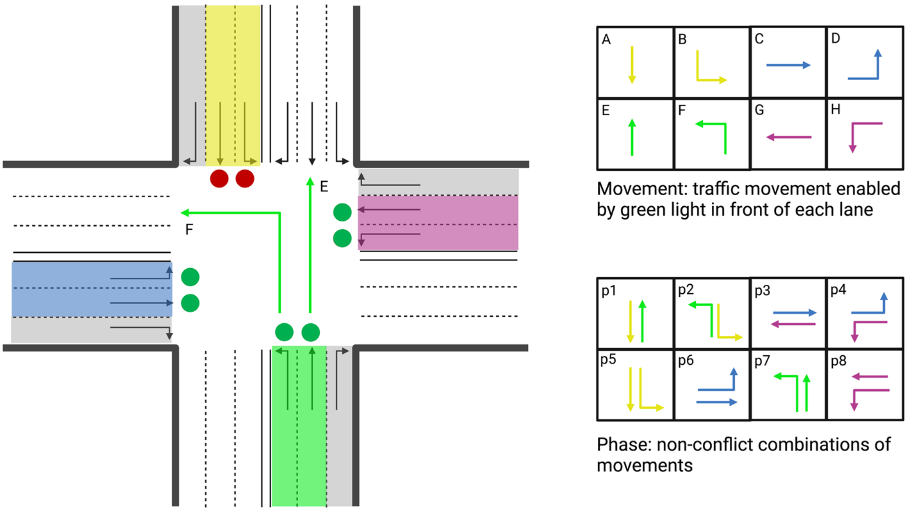

E
F
A
B
C
D
E
F
G
H
Movement: traffic movement enabled by green light in front of each lane
p1
p2
p3
p4
p5
p6
p7
p8
Phase: non-conflict combinations of movements

Fig. 2 Definition of movement and phase

Table 3 Information returned from `World` class and used as Reward

Reward information	Description
lane_count	Number of vehicles on the same intersection
lane_waiting_count	Number of stopped vehicles on the same intersection
lane_waiting_time_count	Waiting time of vehicles on the same intersection since their last action
lane_delay	The delay of vehicles on the same intersection
pressure	The sum of the difference in vehicle density between the in-coming lane and the out-going lane under currently enabled movement on the same intersection
passed_count	Number of vehicles that passed the intersection during time interval Δt after the last action
passed_time_count	Total time spent on approaching lanes of vehicles that passed the intersection during time interval Δt after the last action

two simulators *LibSignal* supports. To directly use these datasets in *LibSignal*, we have converted all the 9 datasets into the format of atomic files, and provided the conversion tools for new datasets. Please refer to our GitHub page for dataset statistics, preprocessed copies, and conversion tools at https://darl-libsignal.github.io/.

Cross-simulator atomic files To make the experimental configuration adaptive across different simulators, we consider two basic units called "atomic files" that can map to the different simulation environments. (1) *Road network file* stores the basic structure of a traffic network consisting of road, lane, and traffic light information. The atomic file under the SUMO environment is in the format of `.net.xml` while in CityFlow it's `.json`. (2) *Traffic flow file* stores the vehicles information and is in `.rou.xml` and `.json` format in SUMO and CityFlow respectively. To make experiments comparable among different simulators, we also provide a `converter.py` tool to convert basic atomic files between different simulators. For example, it takes in *Road network file* and *Traffic flow file* from the source simulator and generates new files in the target simulator's formation, which

```
python converter.py

# conversion type: CityFlow2SUMO(c2s) or SUMO2CityFlow(s2c)
--typ s2c

# SUMO road network file to be converted
--or_sumonet sumofile.net.xml

# SUMO traffic flow file to be converted
--or_sumotraffic sumofile.rou.xml

# SUMO configuration file
--sumocfg sumofile.sumocfg

# CityFlow road network file to be generated
--cityflownet cityflow_roadnet.json

# CityFlow traffic flow file to be generated
--cityflowtraffic cityflow_flow.json
```

Listing 1 A conversion example from SUMO to CityFlow

could later be used in experiments. Figure 4 shows the converted network between different simulators. We also provide a conversion example from SUMO to CityFlow in Listing 1.

3.3 Traffic signal control API

Once the necessary parameters have been set up for simulation and agents, we can start a traffic light control task experiment. The `World` environment is highly homogeneous across different simulators and could provide unified interfaces to communicate with different agents.

Homogeneous world In *LibSignal*, `World` module provides the basic information from different simulators in the unified interface, which could later be utilized for interacting between different simulators and `Agent`. The unified interface `TSCEnv` module is designed for unifying the management of different `Worlds` to interact with `Agent`; it is also compatible with OpenAI Gym (Brockman et al., 2016), which allows *LibSignal* easily deploy the other standard RL algorithms. The `TSCEnv` module contains `World` and `Agent`, and users can decide which simulator to use to interact with `Agent` by specifying the `World` and the corresponding engine. In `TSCEnv` module, it will call `reset` function to initialize the world environment when an episode starts, and call `step` function to make the `World` perform the new action, then return obs, rewards, and other information from the updated environment. In Listing 2, we present how to initialize `TSCEnv` class, specify which simulator and its engine are being used, and define its basic functions. To better support multi-agent reinforcement learning, we also provide `TSCMAEnv` which has the same interface with `TSCEnv`. This environment inherent PettingZoo (Terry et al., 2021) AEC environment and is safer and more flexible for multiple agents interacting with the environment. More details about `World` class implementation can be found in Appendix A.2.

```python
import gym

class TSCEnv(gym.Env):
    def __init__(self, world, agents, metric):
        self.agents = agents
        self.metric = metric

        # select which simulator is being used
        self.world = world
        self.eng = self.world.eng

    def step(self, actions):
        # call the world to execute the actions
        self.world.step(actions)

        # get the latest information from the updated environment
        obs = [agent.get_ob() for agent in self.agents]
        rewards = [agent.get_reward() for agent in self.agents]
        dones = [False] * len(self.agents)
        infos = {}
        return obs, rewards, dones, infos

    def reset(self):
        self.world.reset()
        obs = [agent.get_ob() for agent in self.agents]
        return obs
```

Listing 2 Different worlds managed in one environment

Unified interfaces *LibSignal* provides unified interfaces to process common information with `Generator` module and `Metrics` module, it also provides `Task` module and `Trainer` module to manage the entire process of a task to make the framework more extensible.

- `Generator` class. For lane-level and intersection-level information, including state, reward, phase, and other metrics, we provide `Generator` module, which could interact with different `World` classes and then sort and pass information to different `Agent` classes. The information will later be utilized by `Agent` module to train their models or decide next step actions and feedback to `World` module. Details of `Generator` are shown in Table 4.
- `Metrics` class. We also implement several metrics in `Metrics` module so that different agents can be compared under the same standard. It updates various metrics by interacting with `Agent` and `Generators` after each step during the training or evaluation process. *LibSignal* currently supports five metrics. Details of `Metrics` definition are shown in Table 5.
- `Task and Trainer` class. In *LibSignal*, `Task` module is designed to manage the entire process, including training and evaluation, aiming to improve its extensibility and make it suitable for different tasks. `Trainer` module decouples and implements the different stages of a `Task`. In Listing 3, we present core codes on building `TSCTask` and `TSCTrainer` for traffic signal control tasks.

Table 4 Details of generator class

Generator class	Supported outputs	Description
IntersectionPhaseGenerator	cur_phase	Generate information based on statistics of intersection phases
IntersectionVehicleGenerator	vehicle_map, passed_count, passed_time_count and cur_phase	Generate intersection- level information
LaneVehicleGenerator	lane_count, lane_waiting_count, lane_waiting_time_count, lane_delay and pressure	Generate lane-level information

Table 5 Metrics definition

Evaluation metrics	Description
Average travel time (*travel time*)	The average time that each vehicle spent on traveling within the network, including waiting time and actual travel time
Queue length (*queue length*)	The average queue length over time, where the queue length at time t is the sum of the number of vehicles waiting on lanes
Approximated delay (*delay*)	Averaged difference between the current speed of the vehicle and the maximum speed limit of this lane over all vehicles, calculated from $1 - \frac{\sum_{i=1}^{n} v_i}{n*v_{max}}$ (Ault & Sharon, 2021)
Real delay (*real delay*)	The time a vehicle has traveled within the network minus the expected travel time
Throughput (*throughput*)	The number of vehicles that have finished their trips until the current simulation step

3.4 Comprehensive models

LibSignal implements three baseline controllers and seven RL-based controllers covering Q-learning and Actor-Critic methods, as is shown in Table 6. These methods can also be integrated with existing RL implementation packages and customized on their state, action, and reward design.

Extensible design *LibSignal* provides a flexible interface to help users customize their own RL model and RL design (state, reward, and action).

- `Agent` class. Users can define their model through `Agent` module by completing abstract methods predefined in `BaseAgent` class. Existing RL libraries like `pfrl` can also be integrated into `Agent` class. *LibSignal* also provides different state and reward functions by instantiating `Generator` with subscribed function names in `info_functions` to retrieve queue length, pressure, average lane speed, etc. For example, for RL-based agent, it would create state, phase, reward, queue length and delay `Generators` to retrieve the corresponding information. We provide an example for creating `Generators` for RL-based agent in Listing 4. Users could also customize their own reward or state functions by constructing a key-value mapping between newly defined functions and `info_functions`, which could be carried to `Agent` later by `Generator` class.

Finally, users can use code in Listing 5 to run the model pipeline.

4 Experiment

This section presents our results and verifies that our implementation is consistent with previous publications. In the second part, we compare different algorithms' performance with varying datasets in both SUMO and CityFlow. Finally, we test the feasibility of *LibSignal* to verify it can properly run on large-scale and complex road networks. Further, we also adapt algorithms from widely used RL library to testify our `Agent` module is flexible and easy to manipulate. Along the experiments, we will discuss the answers to several questions that motivate *LibSignal*: *Which simulator should I conduct experiments on? Which evaluation metrics should I use? Which RL method should I choose? Is LibSignal suitable for my research?*

4.1 Validation and calibration

For testifying our PyTorch benchmark algorithms implementation, we compare the learning curves and final performances of the RL algorithms originally implemented in the TensorFlow library. The simulator setting and observed traffic information are chosen to be similar to those used in previous publications.

TensorFlow to PyTorch validation To validate the model's performance under the framework, *LibSignal* re-implemented some of the previous models from Tensorflow with PyTorch. For example, *IDQN* (Zheng et al., 2019) and *CoLight* (Wei et al., 2019) are originally implemented in Tensorflow, we reproduce the experiments of these models and compare their performance in CityFlow simulator. Figure 3 presents the learning curves and final performance of the original TensorFlow and our PyTorch implementation. On CityFlow1x1, the average travel time (in seconds) of our *IDQN* implementation converges to 108.44, which is also close to the original's 127.07 from Zheng et al. (2019). On HangZhou4x4, our *CoLight* implementation converges to 344.41 on average travel time metric, which is close to TensorFlow's 344.49 from Wei et al. (2019).

SUMO and CityFlow calibration To validate that the algorithms' performances are consistent in both SUMO and CityFlow, we calibrate under three roads networks Grid4x4, Cologne1x1 and HangZhou4x4. Their road network is shown in Fig. 4. In addition, we compare *MaxPressure*, *SOTL*, and *FixedTime* algorithms' performance since these three algorithms are deterministic given fixed network and traffic flow files. Table 7 shows the overall performance before and after calibration. We have the following observations:

- Under gird-like networks (Grid4x4, HangZhou4x4), SUMO and CityFlow could achieve similar performance. Different agents' performance is not identical across simulators, but their rank within the same simulator is relatively consistent.
- The discrepancy appears under more complex networks like Cologne1x1 before calibration. Before calibration, we can see that *FixedTime* and *SOTL* perform worse than

```
class TSCTask(BaseTask):
    def run(self):
        # train and evaluate the model
        self.trainer.train()
        self.trainer.test()

class TSCTrainer(BaseTrainer):
    def __init__(self):
        super().__init__()
        self.create_world()
        self.create_agents()
        self.create_metrics()
        self.create_env()

    def train(self):
        total_num = 0
        for e in range(self.episodes):
            obs = self.env.reset()
            for i in range(self.steps):
                actions = []
                for idx, ag in enumerate(self.agents):
                    action = ag.get_action(obs[idx])
                    actions.append(action)
                obs, rewards, dones, _ = self.env.step(actions)

                # update the metrics
                self.metric.update(rewards)
                total_num += 1

                # update the model and the target model
                if total_num % self.update_model_rate == 0:
                    [ag.train() for ag in self.agents]
                if total_num % self.update_target_rate == 0:
                    [ag.update_target_network() for ag in self.agents]

                # log the model’s performance
                self.writeLog(’TRAIN’,self.metric)

    def test(self):
        obs = self.env.reset()
        for i in range(self.steps):
            actions = []
            for idx, ag in enumerate(self.agents):
                action = ag.get_action(obs[idx])
                actions.append(action)
            obs, rewards, dones, _ = self.env.step(actions)
            self.metric.update(rewards)
            self.writeLog(’TEST’,self.metric)
```

Listing 3 An example to build Task and Trainer for traffic signal control tasks

MaxPressure. After calibration, the same agent’s performance is close and within an acceptable discrepancy between SUMO and CityFlow. Moreover, the ranking of the performances for different agents is consistent across different simulators after calibration. To double-check check our calibration is correct, *IDQN* algorithms are also trained under Cologne1x1 with different simulators. In CityFlow and SUMO simulator, the results are 58.0771 and 61.7014 in Cologne1x1 w.r.t average travel time (in seconds), proving our calibration is correct.

SUMO and CityFlow discrepancy We modified some default settings in SUMO to compare the algorithms’ performance under different simulators as comprehensively and

Table 6 Detailed design of implemented models in *LibSignal*

Agent	State	Action	Reward	Method	Description
FixedTime	–	Cyclic	–	Non-RL	This agent gives a predefined time duration and phase order
SOTL	–	Acyclic	–	Non-RL	This agent selects the phase among all to maximize the pressure calculated from the upstream and downstream queue length
MaxPressure	–	Acylic	–	Non-RL	This agent determines next phase by considering competitive phases
IDQN	Lane vehicle count, phase	Acylic	Lane waiting vehicle count	Q-Learning	This agent determines each intersection's action with its own intersection information
CoLight	Lane vehicle count, phase	Acylic	Lane waiting vehicle count	Q-Learning	This agent considers neighbor intersections' cooperation through graph attention networks
PressLight	lane vehicle count, phase	Acylic	Pressure	Q-Learning	This agent coordinates traffic signals by learning MaxPressure
IPPO	lane vehicle count, phase	Acylic	Lane vehicle waiting time count	Actor-Critic	This agent is imported from `pfrl` with proximal policy optimization
MAPG	lane vehicle count	Acylic	Lane waiting vehicle count	Actor-Critic	This agent optimizes agent control policy with muti-agent policy gradient method
FRAP	Lane vehicle count, phase	Acylic	Lane waiting vehicle count	Q-Learning	This agent captures the phase competition relation between traffic movements through a modified network structure.
MPLight	Pressure, phase	Acylic	Pressure	Q-Learning	This agent is based on FRAP and integrates pressure into state and reward design.

```python
class RLAgent(BaseAgent):
    def __init__(self, world, inter_obj):
        super().__init__(world)
        # create state generator
        self.ob_generator = LaneVehicleGenerator(world, inter_obj, ["lane_count"])

    def get_ob(self):
        # get observation from environment
        return self.ob_generator.generate()

    def get_action(self, ob):
        # generate action
        actions = self.model(ob)
        return numpy.argmax(actions, axis=1)
```

Listing 4 An example to create Generators for RL-based agent

fairly as possible, e.g., disabled the feature of dynamic routing and set teleport to -1. But it is worth noting that there are still some discrepancies that our current calibration cannot address. Here are the differences between CityFlow and SUMO, which will affect the performance of models:

- Traffic signals. There only exists red and green traffic signals in CityFlow, but in SUMO, more traffic signals are provided to deal with complicated traffic conditions. For example, SUMO has pedestrian traffic signals that CityFlow does not support, where vehicles will decelerate when they encounter the pedestrian traffic signals in SUMO, but in CityFlow the vehicles will pass the intersection at normal speed. In this case, the traffic signal information is not consistent between CityFlow and SUMO.
- Vehicles definition. In CityFlow, we can define a vehicle and its behavior by setting parameters such as shape, acceleration, deceleration, etc. While SUMO provides more parameters to define the behavior of the running vehicle, many of which are unavailable in CityFlow. For example, the vehicles in SUMO will randomly decelerate according to their attributes when approaching a traffic signal, even though the traffic signal is located further away. In comparison, the vehicles in CityFlow will not have such randomness.
- Lane select model. In CityFlow, the vehicle will automatically select the corresponding lanes according to the pre-defined route and generally will not change lanes during traveling, while in SUMO, it will change lanes according to the current road conditions so that the routes of the vehicle on different simulators are not identical.

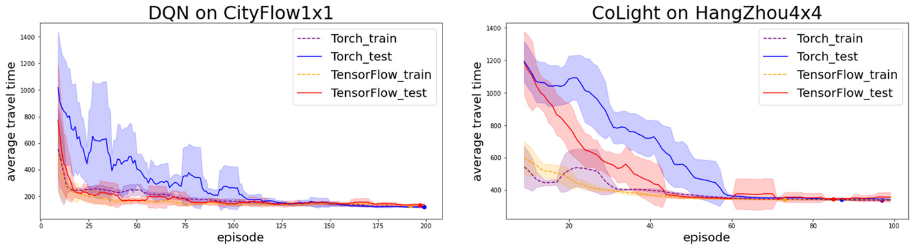
Due to these reasons, different methods' performance varies across simulators, but the rankings of these methods in the same simulator are generally consistent.

4.2 Overall performance

All benchmark RL models and traditional traffic control algorithms were compared under different simulator environments for comparative purposes. All observations and rewards are set to be the same if not explicitly mentioned, and all the hyperparameters are set

Listing 5 An example to run LibSignal

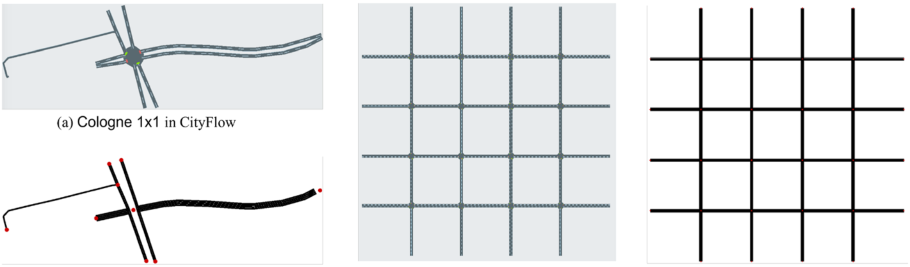
```
python run.py

# set the task as traffic signal control
--task tsc

# set the model as DQN
--agent dqn

# set the simulator environment as CityFlow
--world cityflow

# set the dataset as cityflow1x1
--network cityflow1x1
```



(a) *IDQN* in CityFlow1x1

(b) *CoLight* in HangZhou4x4

Fig. 3 Convergence curve of models implemented in their original form (Tensorflow) and *LibSignal* (PyTorch). Y-axis is the testing result w.r.t. average travel time (in seconds). Validation for more models can be found in Appendix A.3



(a) Cologne 1x1 in CityFlow

(b) Cologne 1x1 in SUMO

(c) Grid 4x4 in CityFlow

(d) Grid 4x4 in SUMO

Fig. 4 Road networks in different simulators for calibration. The pictures with a gray background are the visualization of networks in the SUMO simulator, and the ones with the white background are in CityFlow. These are the outlines of traffic structures transferred between each other(More networks can be found in Appendix A.4)

according to the original implementations. We represent the final results truncated at 200 training iterations for a fair comparison since most algorithms could converge within this period. While *IPPO* and *MAPG* are noticeable for their high demand for training time, we provide the full converge curve in Appendix A.6.2. The results are summarized in Table 8. We have the following observations:

Table 7 Performance comparison of agents w.r.t. average travel time (in seconds) before and after calibration

Calibration	Before						After					
Dataset	Grid4x4		Cologne1x1		HangZhou4x4		Grid4x4		Cologne1x1		HangZhou4x4	
Avg. travel time	Cityflow	SUMO	Cityflow	SUMO	Cityflow	SUMO	Cityflow	SUMO	Cityflow	SUMO	Cityflow	SUMO
MaxPressure	161.2878	154.1493	51.2785	55.2013	365.0634	255.0052	160.8778	154.1493	63.3874	55.2013	361.6611	347.6305
FixedTime	290.9525	222.1324	156.6599	240.4572	689.0221	300.6685	259.3320	222.1324	175.1646	240.4572	628.0342	572.9087
SOTL	185.8846	230.9540	1356.6037	66.2663	354.1250	389.7881	196.3191	230.9540	1187.2795	1219.0531	372.2521	397.0227

Table 8 Performance of agents in CityFlow and SUMO with **best** and second best performance highlighted

Network	Cityflow 1x1							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
FixedTime(t_fixed=10)	723.7759	106.0722	4.8922	1238	772.7964	90.7889	5.2711	1115
FixedTime(t_fixed=30)	532.8481	99.5528	5.3296	1479	470.7593	81.5806	5.7098	1512
MaxPressure	152.8916	46.9056	5.8929	1932	113.3881	24.4778	5.3566	**1966**
SOTL	234.9243	67.6417	5.2610	1835	194.8325	58.2083	5.2621	1905
IDQN	**108.4364**	**24.8528**	**0.6142**	**1984**	**104.3616**	**19.9750**	**0.5529**	**1966**
MAPG	338.9654	122.5583	0.6993	1698	118.1472	52.7528	0.6372	1413
IPPO	241.4231	52.7056	0.6944	1846	123.9152	81.0722	0.7326	1309
PressLight	112.6635	28.0639	0.6715	1976	107.8661	21.8472	0.5716	1956
FRAP	112.6571	26.9444	0.6194	**1977**	106.3967	21.2028	0.6137	1956
MPLight	113.6611	28.7083	0.6571	**1977**	110.6822	23.6972	0.6146	1954
Network	Cologne 1x1							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
FixedTime(t_fixed=10)	175.1646	48.0889	3.9234	1889	240.4572	60.1056	5.7777	1857
FixedTime(t_fixed=30)	166.9264	48.6222	4.1468	1904	176.0727	48.6500	5.8119	1927
MaxPressure	63.3874	7.6944	2.7739	2002	**55.2013**	**7.3944**	3.1452	**1997**
SOTL	1187.2795	94.6917	5.0666	618	1219.0531	111.3444	7.1133	421
IDQN	58.0771	9.0278	0.3506	**2003**	61.7014	9.8167	0.4423	1996
MAPG	73.6464	12.7278	0.3882	1992	85.0583	14.0889	0.4771	1974
IPPO	**51.0239**	**6.7083**	**0.3098**	1998	58.3778	9.0833	0.4320	1996
PressLight	58.2834	9.5806	0.3550	1994	62.0060	9.4333	**0.4238**	1996
FRAP	59.4615	10.7278	0.3994	1999	75.2950	15.4500	0.5426	1983
MPLight	60.2680	10.4389	0.4041	1993	85.3382	19.0417	0.5691	1984

Table 8 (continued)

Network	Cologne 1x3							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
FixedTime(t_fixed=10)	502.4304	16.0889	2.3916	1906	124.6869	6.9944	2.4559	2801
FixedTime(t_fixed=30)	99.4557	6.9824	1.8701	2782	153.0152	10.5796	3.2154	2769
MaxPressure	53.5789	1.4880	1.0825	2788	74.2238	1.6352	1.4291	2810
SOTL	992.7596	41.4046	3.3724	1145	1196.4375	64.1787	5.4957	930
IDQN	49.6132	0.4491	0.0964	**2792**	80.8113	**1.2676**	**0.1970**	2803
MAPG	50.8571	0.7991	0.1222	2791	**70.8351**	1.2991	0.2040	2814
IPPO	49.6507	**0.3463**	**0.0952**	**2792**	78.0800	8.8537	0.2666	2586
PressLight	**49.2661**	0.4537	0.0993	**2792**	71.0014	1.3250	0.2038	**2819**
FRAP	65.8111	2.3481	0.1973	**2792**	108.3285	6.8167	0.4021	2782
MPLight	59.5700	1.2157	0.1360	**2792**	91.9115	2.7620	0.2829	2801
Network	Grid4x4							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
FixedTime(t_fixed=10)	259.3320	2.9967	2.2219	**1473**	222.1324	2.6684	1.5831	1443
FixedTime(t_fixed=30)	298.8744	4.1672	2.4580	1468	284.1913	4.2457	2.1371	1427
MaxPressure	160.8778	0.6865	0.9726	**1473**	154.1493	0.8556	0.6315	1460
SOTL	196.3191	1.4200	1.3935	**1473**	230.9540	2.8828	1.6066	1436
IDQN	143.0957	0.2561	0.0525	**1473**	149.0938	0.6898	0.0424	1461
MAPG	752.6592	16.6799	0.2934	1329	458.2909	16.3719	0.2621	942
IPPO	545.4297	11.0559	0.1454	1437	237.5305	8.9458	0.1237	1131
PressLight	149.7339	0.4127	0.0630	**1473**	152.8056	0.8014	0.0486	1461

Table 8 (continued)

Network	Grid4x4							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
CoLight	151.5492	0.4675	0.0638	**1473**	154.0991	0.8144	0.0483	1462
FRAP	140.2899	0.2080	0.0497	**1473**	147.4010	0.6651	**0.0411**	**1464**
MPLight	**139.2166**	**0.1847**	**0.0479**	**1473**	**147.1888**	**0.6530**	0.0413	1463

Results with the best performance are highlighted in bold and underlined, and the second-best performance is underlined. For Travel time, Queue, and Delay, the lower, the better. For Throughput, the higher, the better

Table 9 Performance comparison w.r.t. the running time (in seconds) of different methods under two simulators

Running time	Simulator	FixedTime	MaxPressure	SOTL	IDQN	IPPO	PressLight
Cityflow1x1	CityFlow	5.7593	9.5857	6.0272	2461.7496	2450.7397	1960.7932
	SUMO(Libsumo)	6.0006	4.2174	4.2988	3691.8403	2833.0523	3619.1282
	SUMO(Traci)	73.6791	46.5712	51.488	30279.5201	37662.9677	27697.9098
Cologne1x1	CityFlow	3.3049	2.7601	4.6527	1641.6128	3148.0649	3066.7189
	SUMO(Libsumo)	4.3649	3.1411	3.2771	2649.0341	2581.8182	2863.9526
	SUMO(Traci)	133.6334	24.9327	71.7565	11243.2917	31431.1760	11535.0851

- Under the same dataset and simulator, the performance of the same model varies w.r.t. current four metrics. *Travel time* and *queue length* are consistent with each other in most cases. *Throughput* sometimes is hard to differentiate the identical results under certain datasets. For example, in Grid4x4, most of the methods served the same number of vehicles. *Delay* sometime aligns with *queue length* and *travel time* but can be different from all the other three metrics in some cases, e.g., Cologne1x1 and Grid4x4 under SUMO. This is because the delay is approximated from the average speed proposed by Ault & Sharon (2021) and is not the actual delay calculated by vehicles' total travel time and desired travel time under maximum speed.
- Traditional transportation methods like *MaxPressure* can achieve consistent satisfactory performance though it is not the best. *IDQN* performs the best in single intersection scenarios. With more complex road networks like Cologne1x3, *PressLight* achieves better performance.

We also conducted experiments for different network scalability and complexity, and the results can be found in Appendix A.6.

4.3 Discussion

Which simulator should I conduct experiments on?

From the running time comparison in Table 9 between SUMO and CityFlow simulator, we find that CityFlow and SUMO (with Libsumo)'s time cost is around ten times less than SUMO (with TraCI) which indicates its higher running efficiency. Different from CityFlow, SUMO provides a more accurate depiction of vehicles' state and more complex traffic operations, including changing lanes and 'U-turn'. Also, SUMO provides users with more realistic settings, including pedestrians, driver imperfection, collisions, and dynamic routing. Thus, it is more powerful on complex networks and reflecting real-world scenario.

Which evaluation metrics should I use?

Average travel time is generally a good metric to evaluate algorithms' performance on traffic control tasks. But for settings with dynamic routing, the travel time would not be a good metric as the average travel time of a vehicle can change with dynamic routing. In *LibSignal*, the simulation under SUMO disabled the dynamic routing feature so that the travel time would be good on the current settings. From Table 8, we can see that lane delay and throughput are not always consistent between different simulators and even in the same simulator environment. Sometimes they often show contradictory performances in different datasets. Therefore, we suggest researchers report travel time as a necessity and other metrics of their interest in their papers.

Which RL method should I choose?

From our experiments in Table 8, we can see that Actor-Critic based RL algorithms need a long time to converge. In Grid4x4, *MAPG* and *IPPO* algorithms still perform poorly after 2000 iterations. *IDQN* and other Q-learning-based algorithms are generally good choices in all four datasets. We can see that they outperform traditional non-RL algorithms all the time. Comparing results in large-scale networks, e.g., Grid4x4, we find *FRAP* and *MPLight* could bring improvement compared to *IDQN* algorithm.

When should I use *LibSignal*?

Since *LibSignal* provides a highly unified interface to help users choose or define their functions and extract information from the simulator's environment, it is a powerful platform for users to investigate the best combination of state and reward functions for current state-of-the-art or implemented models. Also, users could compare their algorithms with our implemented baseline model using the evaluation metrics we provided. In addition, since *LibSignal* supports multiple simulation environments, users could also conduct experiments in the different simulation environments to validate that their algorithms are robust and achieve generally good performance under different settings. It makes *LibSignal* a unique testbed for Sim2Real (Zhao et al., 2020; Peng et al., 2018) in the traffic signal control domain.

5 Conclusion

In this paper, we introduced *LibSignal*, a highly unified, extensible, and comprehensive library for traffic signal control tasks. We collected and filtered nine commonly used datasets and implemented ten different baseline models across two influential traffic simulators, including SUMO and CityFlow. We both conducted experiments to prove our PyTorch implementation could achieve the same level of performance as the original TensorFlow official code and calibrated simulators to improve the reliability of our cross-simulator environment. Moreover, the performance of all implemented algorithms was compared under various datasets and simulators. We further provided the discussion for researchers interested in this topic with our benchmarking results. In the future, we will implement more state-of-the-art RL-based algorithms and continually support more simulator environments. Further calibration efforts will be made to help different algorithms' performance comparisons across different simulators.

Appendix

A.1: Documentation and license

LibSignal is open source and free to use/modify under the GNU General Public License 3. The code and documents are available on Github at https://darl-libsignal.github.io/. The embedded traffic datasets are distributed with their own licenses from Reinforcement Learning for Traffic Signal Control 2022 and Ault & Sharon (2021), whose licenses are under the GNU General Public License 3. SUMO is licensed under EPL 2.0 and CityFlow is under Apache 2.0. All experiments can be reproduced from the source code, which includes all hyper-parameters and configurations. The authors will bear all responsibility in case of violation of rights, etc., ensure access to the data and provide the necessary maintenance.

A.2: Details of world class

`World` class can extract and integrates the information and then pass the information to `Agent`. Specifically, in the initialization phase of the `World`, it would create an engine for the user-specified simulator and read the road network from the atomic file in the format of `.json` or `.net.xml`, then create `Intersection`, `info_functions` object, and other necessary variables to describe and process the information. In the training or evaluating phase, it would take `step()` function to interact with the simulator and then update the information.

- `Intersection` class. `Intersection` is the basic component of the `World`. All of the information is stored in variables of `Intersection` class, for example, `roads`, `phases`, and etc.
- `info_functions` object. In the `World` class, we provide an `info_functions` object inside to help retrieve information from different simulator environments and update information after each simulator performs a step. The `info_functions` contain `state` information including `lane_count`, `lane_waiting_count`, `lane_waiting_time count`, `pressure`, `phase`, and `metrics` including `throughput`, `average_travel_time`, `lane_delay`, `lane_vehicles`. These `info_functions` will later be called by `Generator` class and pass information into `Agent`.
- `step()` function. It is another common function shared between different `World` classes. It takes in actions returned from `Agent` class and passes them into the simulator for next step execution. And `action` is either sampled from action space for exploration or calculated from the model after optimization. Generally, the action space contains eight phases. However, in highly heterogenous traffic structures, the action space may differ and is provided by the simulators whose action parameters are taken from configuration files.

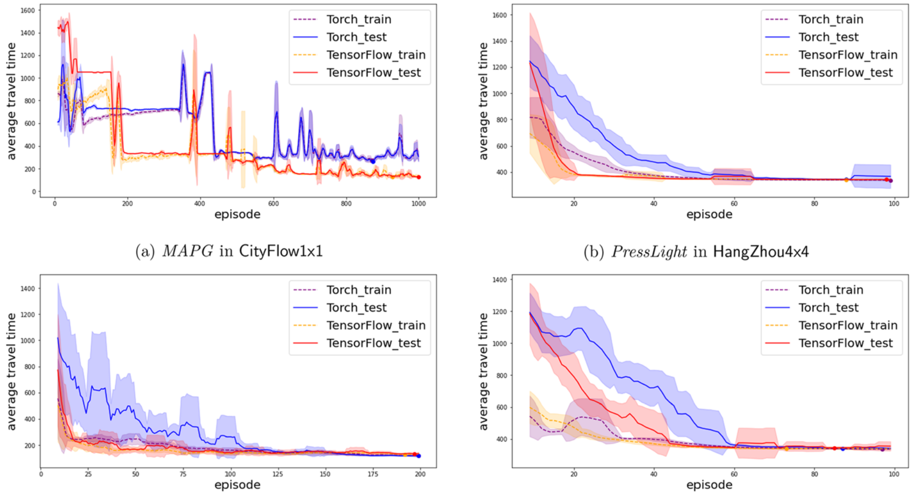
Listing 6 presents an example for creating a World for the CityFlow simulator environment, including 3 sections: (1) initialization. (2) create Intersections, roads, lanes, and other necessary parameters. 3) define info_functions to facilitate retrieve information.

```python
import cityflow
class World(object):
    def __init__(self, cityflow_config):
        # section 1: initialization
        # create the engine for the specified simulator
        self.eng = cityflow.Engine(cityflow_config)

        #read road network from atomic file
        self.roadnet = self._get_roadnet(cityflow_config)

        # section 2: get intersections, roads and lanes
        self.intersections = self._get_intersections()
        self.all_roads = self._get_roads()
        self.all_lanes = self._get_lanes()

        # section 3: define info_functions
        self.info_functions = {
            "lane_delay": self.get_lane_delay, # get road network delay
            "phase": self.get_cur_phase, # get the current phase
            "pressure": self.get_pressure # get the pressure
        }

    def step(self, actions):
        #Take actions and update information
        for i, action in enumerate(actions):
            self.intersections[i].step(action)

        # interact with simulator
        self.eng.next_step()

        # update information in self.info_functions
        self._update_infos()
```

Listing 6 An example to create World class

A.3: Validation

To validate our PyTorch re-implementations performance, we compare the performance of four originally implemented in TensorFlow. Figure 5 shows the converge curve of *MAPG*, *PressLight*, *IDQN*, and *CoLight* in both the train and test phase, which are not provided in Sect. 4.1. The final performance in Table 10 shows that all four new implementations are consistent with their original TensorFlow implementations.



(a) *MAPG* in CityFlow1x1

(b) *PressLight* in HangZhou4x4

(a) *IDQN* in CityFlow1x1

(b) *CoLight* in HangZhou4x4

Fig. 5 Convergence curve of models implemented in their original form (TensorFlow) and in *LibSignal* (PyTorch). Y-axis is the testing result w.r.t. average travel time (in seconds)

Table 10 Best episode performance w.r.t. average travel time (in seconds). The performance of models is consistent under TensorFlow and PyTorch

Library	TensorFlow	PyTorch
MAPG on CityFlow1x1	125.79	180.61
IDQN on CityFlow1x1	131.80	108.44
PressLight on HangZhou4x4	342.36	344.75
CoLight on HangZhou4x4	344.49	341.41

A.4: Network conversion

Current *LibSignal* includes 9 datasets which are converted and calibrated. Their road networks are shown in Fig. 6. Other configurations of CityFlow1x1 datasets are similar to CityFlow1x1 that appeared in the full paper in road network structure, which will not be shown here.

A.5: Calibration steps

To validate that the performance of the algorithms is consistent in both SUMO and CityFlow, we calibrate the simulators in the following aspects:

- **Calibration from SUMO to CityFlow**: To make the conversion of complex networks from SUMO compatible with CityFlow, we redesign the original convert files from Zhang et al. (2019) with the following: (1) For those `.rou` files in SUMO that only specify source and destination intersections and ignore roads that would

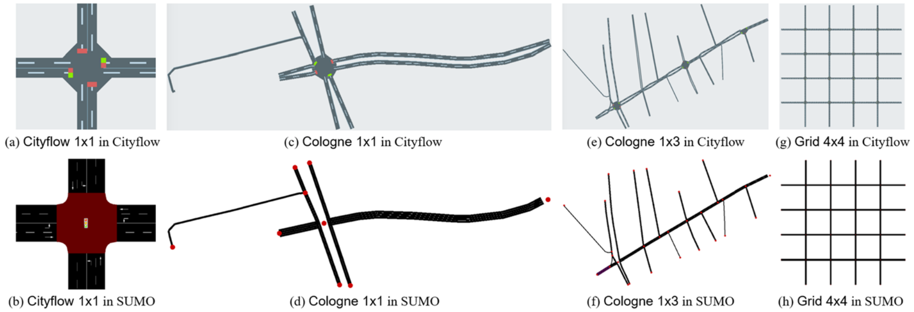

(a) Cityflow 1x1 in Cityflow
(c) Cologne 1x1 in Cityflow
(e) Cologne 1x3 in Cityflow
(g) Grid 4x4 in Cityflow
(b) Cityflow 1x1 in SUMO
(d) Cologne 1x1 in SUMO
(f) Cologne 1x3 in SUMO
(h) Grid 4x4 in SUMO

Fig. 6 Road networks in different simulators for calibration

be passing, the router command line in SUMO should be applied to generate full routes before converting it into CityFlow's `.json` traffic flow file. (2) We treat all the intersections without traffic signals in SUMO as "virtual" nodes in CityFlow's `.json` road network file. (3) We keep the time interval the same for red and yellow signals in SUMO and CityFlow. (4) SUMO has a feature of the dynamic routing of vehicles that CityFlow does not have, currently all the simulations under SUMO in *LibSignal* disables the dynamic routing. (5) To reduce the differences in the results of different simulators caused by the fact that the phases in the SUMO environment cannot fully be transferred to the phases in the CityFlow environment (SUMO provides more abundant phases than CityFlow), we modify the judgment conditions of phase transformation.

- **Calibration from CityFlow to SUMO**: (1) The vehicles in CityFlow's traffic flow file need to be sorted according to their departure time because the SUMO traffic file defaults to the depart time of the preceding vehicle earlier than the following vehicle. (2) The type of vehicle should be clearly indicated in order to limit the max speed of the vehicle.

A.6: Supplementary results

We conduct experiments on all nine datasets and also provide results of the best episode, full converge curves and standard deviations of the performance on the four datasets in the full paper.

A.6.1: Other comparison studies on datasets not shown in full paper

Table 11 shows the result of performance on the other five datasets. It shows *PressLight* and *IDQN* are the most stable algorithms most of the time.

Table 11 Performance of agents in CityFlow and SUMO on additional datasets that are not shown in Sect. 4.2 with **best** and second best performance highlighted

Network	Cityflow 1x1(Config2)							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
FixedTime(t_ fixed=10)	702.0847	90.3417	4.3542	914	444.5625	74.9	4.095	832
FixedTime(t_ fixed=30)	305.9428	62.9361	4.3038	1246	181.3847	42.9972	4.2947	1310
MaxPressurre	91.4333	11.3806	4.5774	1388	85.7407	8.7333	4.1472	1377
SOTL	116.2653	20.0139	3.6794	1381	96.8388	13.2667	3.1959	1371
IDQN	**79.6013**	**7.5806**	**0.4253**	**1390**	**78.0051**	**5.8111**	**0.3648**	**1379**
MAPG	262.1785	75.3056	0.6467	1237	125.3895	61.075	0.5812	955
IPPO	733.7248	130.7694	0.6963	748	90.9229	10.6	0.4447	1375
PressLight	84.7516	9.2889	0.4454	1385	85.873	8.9639	0.4318	1378

Network	Cityflow 1x1(Config3)							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
FixedTime(t_ fixed=10)	461.2692	33.5944	2.3989	531	284.4235	29.9806	2.2208	503
FixedTime(t_ fixed=30)	228.8520	24.0667	2.7021	659	173.1985	21.3167	2.8096	680
MaxPressurre	69.7295	2.2250	1.8415	729	69.9601	1.6167	1.4375	**726**
SOTL	89.2005	5.5556	1.8778	729	84.9169	4.4556	1.7441	722
IDQN	**66.9865**	**1.6833**	**0.1816**	730	**68.7369**	**1.2556**	**0.1396**	**726**
MAPG	183.2719	25.7778	0.4086	651	115.5654	26.3361	0.3421	543
IPPO	79.4778	3.8333	0.2672	729	78.8072	3.0500	0.2373	**726**
PressLight	67.4899	1.8861	0.1945	**731**	73.4509	2.4250	0.2169	723

Network	Cityflow 1x1(Config4)							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
FixedTime(t_ fixed=10)	686.7469	96.7222	4.4683	925	469.5721	81.7778	4.0367	811
FixedTime(t_ fixed=30)	339.3052	63.7750	4.4226	1290	204.3520	51.7611	4.5938	1358
MaxPressurre	136.5470	31.9083	5.1600	1556	98.8589	17.3444	4.6113	1602
SOTL	158.4722	40.2611	4.5164	1562	113.0794	23.7389	3.7165	1587
IDQN	101.9282	18.7278	0.5685	1614	**90.0737**	**12.8278**	0.4912	**1614**
MAPG	435.5595	77.6861	0.6195	1179	137.0363	78.4028	0.6875	1130
IPPO	226.4010	58.2306	0.6641	1453	115.8968	52.4528	**0.4787**	1037
PressLight	**90.1724**	**13.2667**	**0.4820**	**1630**	91.8277	13.7250	0.5153	1608

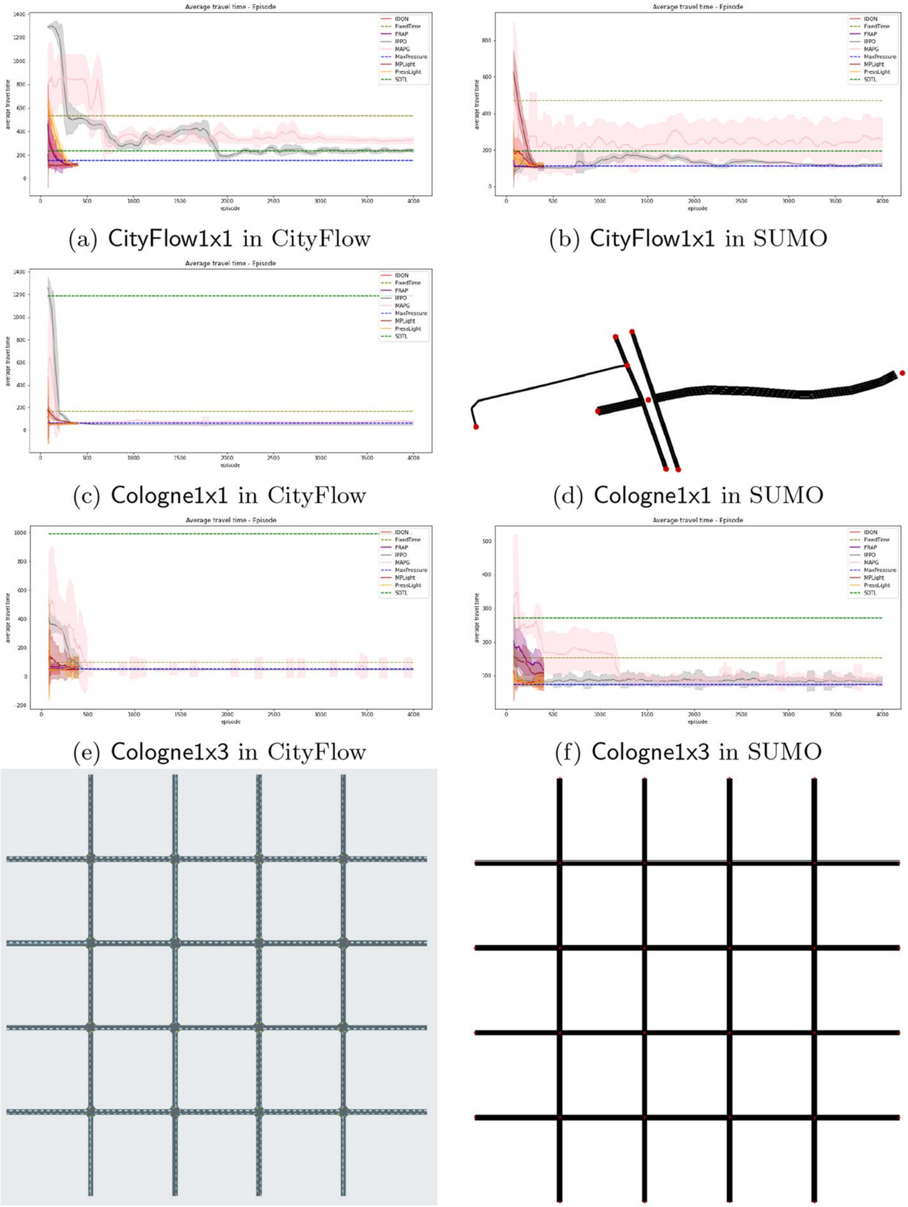
Table 11 (continued)

Network	HangZhou4x4							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
FixedTime(t_ fixed=10)	689.0221	13.9837	0.9636	2385	535.0060	7.9405	2.1130	2495
FixedTime(t_ fixed=30)	575.5565	12.3694	1.8439	2645	580.5826	10.3816	2.4246	2355
MaxPressurre	365.0634	3.5972	1.7891	2928	350.4125	1.2870	0.9678	2732
SOTL	354.1250	2.7229	1.0208	2916	386.7881	3.3717	1.4937	2695
IDQN	**322.9068**	**1.2141**	**0.0679**	2929	**341.8509**	**1.2097**	**0.0689**	2730
MAPG	782.3319	24.0474	0.1406	2331	436.2691	25.1415	0.2374	1442
IPPO	727.9460	15.5786	0.1068	2306	418.0365	7.2328	0.1560	2496
PressLight	329.7855	1.6380	0.0841	**2932**	354.0689	1.4884	0.0803	2728
Colight	331.4348	1.7658	0.0906	2931	343.4998	1.3083	0.0714	**2733**
Network	Manhattan7x28							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
FixedTime(t_ fixed=10)	1575.7847	20.7651	1.2703	7531	953.9797	19.3563	1.4353	2123
FixedTime(t_ fixed=30)	1582.1030	22.3561	1.6659	7801	1144.1702	20.2811	1.8234	2379
MaxPressurre	1335.7877	**17.3380**	1.3035	**9745**	**797.4855**	**15.2944**	**1.3298**	**4614**
SOTL	1612.2468	23.6984	1.4968	8244	869.6206	17.8344	1.4543	3010
IDQN	**1319.4959**	17.4697	**0.0916**	9035	–	–	–	–
MAPG	1586.3388	21.2705	0.1159	7481	–	–	–	–
IPPO	1468.4135	19.4304	0.1130	8300	–	–	–	–
PressLight	1338.7183	18.1332	0.0961	9123	–	–	–	–
CoLight	1493.4200	19.5024	0.1007	8287	–	–	–	–

* Results shown as (-) indicate that no RL methods can be trained within acceptable time and resource in SUMO in Manhattan's road network

A.6.2: Converge curve of Table 8

Figure 7 shows the full converge curve of 2000 episodes for *IPPO* and *MAPG* agents. The result shows that compared to Q-learning agents, Actor-Critic agents are hard to converge on some large or complex datasets, and the convergence time needed is more than ten times of Q-learning methods.



(a) CityFlow1x1 in CityFlow

(b) CityFlow1x1 in SUMO

(c) Cologne1x1 in CityFlow

(d) Cologne1x1 in SUMO

(e) Cologne1x3 in CityFlow

(f) Cologne1x3 in SUMO

(g) Grid4x4 in CityFlow

(h) Grid4x4 in SUMO

Fig. 7 Full converge curve of Table 8

Table 12 The episode of best results for different agents w.r.t. different methods

Network	Simulator	IDQN	MAPG	IPPO	PressLight
Cityflow1x1	CityFlow	193	1205	185	197
	SUMO	187	188	194	185
Cologne1x1	CityFlow	95	1870	172	101
	SUMO	189	1992	346	193
Cologne1x3	CityFlow	133	597	1977	159
	SUMO	177	30*	195*	164
Grid4x4	CityFlow	163	6*	172*	172
	SUMO	186	2*	188*	143

* Though *MAPG* and *IPPO* has the best results in the first few episodes,

their performances are still worse than the other agents

Result of best episode

Table 12 gives the episode number of all datasets. It supports the conclusion that *PressLight*, followed by *IDQN*, has the best sample efficiency compared with other algorithms.

A.6.4: Performance on the benchmark with standard deviations

Table 13 shows the standard deviation of the performance on the four datasets in the full paper.

Extension to other simulators

LibSignal is a cross-simulator library for traffic control tasks. Currently, we support the most commonly used CityFlow and SUMO simulators, and our library is open to other new simulation environments. CBEngine is a new simulator that served as the simulation environment in the KDD Cup 2021 City Brain Challenge [3] and is designed for executing traffic control tasks on large traffic networks. We integrate this new simulator into our traffic control framework to extend *LibSignal* 's usage in other simulation environments. We show the result of *MaxPressure*, *SOTL*, *FixedTime*, and *IDQN*' performance under CBEngine in Table 14.

[3] http://www.yunqiacademy.org/poster

Table 13 The standard deviations of Table 8

Network	Cityflow 1x1							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
IDQN	9.474	2.5995	0.0221	3.0496	2.507	0.6804	0.4424	0.0092
MAPG	10.746	0.2149	1.7188	0.0152	14.182	0.6544	0.4391	0.0031
IPPO	23.020	0.0011	1.9196	0.0055	12.179	0.0446	7.7187	0.015
PressLight	3.335	1.334	0.0127	2.5884	1.960	3.261	3.4475	0.0141
FRAP	3.0118	1.4411	1.5028	0.0106	0.6933	0.3848	0.4273	0.0095
Network	Cologne 1x1							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
IDQN	2.100	0.6913	0.7502	0.0095	0.144	0.6317	0.3986	0.0087
MAPG	2.186	0.0359	0.2659	0.0072	12.283	0.0088	0.0744	0.0001
IPPO	28.048	0.0003	0.0495	0.0015	0.950	0.0013	0.1982	0.0008
PressLight	2.274	0.3753	0.3161	0.0066	0.767	0.6011	0.6375	0.0106
Network	Cologne 1x3							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
IDQN	5.891	0.9075	0.1778	0.0053	0.875	0.0711	0.0111	0.0017
MAPG	2.883	0.0637	0.1467	0.0047	15.494	3.5459	0.5764	0.0134
IPPO	0.950	0.0	0.0159	0.002	4.107	0.0002	0.1743	0.0078
PressLight	0.619	0.124	0.0405	0.0026	4.982	2.3948	0.798	0.0123
Network	Grid4x4							
Simulator	CityFlow				SUMO			
Metric	Travel time	Queue	Delay	Throughput	Travel time	Queue	Delay	Throughput
IDQN	2.006	0.0117	0.0006	0.0	0.795	0.1733	0.0098	0.0005
MAPG	13.330	1.1346	0.071	0.0044	5.420	3.6906	0.2473	0.0049
IPPO	2.432	0.0003	0.0495	0.0015	3.465	0.0004	0.0346	0.0004
PressLight	0.880	0.0491	0.0022	0.0	0.636	0.1045	0.0132	0.0003
CoLight	0.8315	0.0161	0.0009	0.0	1.1633	0.621	0.0384	0.0017

Table 14 Performance on CBEngine simulator

CityFlow1x1	FixedTime	MaxPressure	SOTL	IDQN
Avg. Travel Time	654.4848	150.7677	96.0025	84.5404

Hyperparameters

Table 15 provides the parameters of each algorithm, training environment, and hardware parameters on the server.

Table 15 Hyperparameters of models, servers and training

Model parameters								
FixedTime	t_fixed	30	buffer_size	0	learning_rate	0	learning_start	0
	update_model_rate	0	update_target_rate	0	save_rate	0	train_model	False
	test_model	true	one_hot	false	phase	false	episodes	1
MaxPressure	t_min	10	buffer_size	0	learning_rate	0	learning_start	0
	update_model_rate	0	update_target_rate	0	save_rate	0	train_model	False
	test_model	true	one_hot	false	phase	false	episodes	1
SOTL	t_min	5	min_green_vehicle	3	max_red_vehicle	6	buffer_size	0
	learning_rate	0	learning_start	0	update_model_rate	0	update_target_rate	0
	save_rate	0	train_model	false	test_model	true	one_hot	False
	phase	false	episodes	1				
IDQN	learning_rate	0.001	learning_start	1000	graphic	false	buffer_size	5000
	batch_size	64	episodes	200	epsilon	0.1	epsilon_decay	0.995
	epsilon_min	0.01	update_model_rate	1	update_target_rate	10	save_rate	20
	one_hot	true	phase	true	gamma	0.95	steps	3600
	test_steps	3600	vehicle_max	1	grad_clip	5	train_model	False
	test_model	true	action_interval	10				
MAPG	tau	0.01	learning_rate	0.001	learning_start	5000	graphic	False
	buffer_size	10000	batch_size	256	episodes	2000	epsilon	0.5
	epsilon_decay	0.99	epsilon_min	0.01	update_model_rate	30	update_target_rate	30
	save_rate	1000	one_hot	FLASE	phase	false	gamma	0.95
	steps	3600	test_steps	3600	vehicle_max	1	grad_clip	5
	train_model	false	test_model	true	action_interval	10		
IPPO	learning_rate	0.00025	learning_start	360	graphic	false	value_func_coef	1.0
	batch_size	64	episodes	2000	clip_eps_vf	0.02	entropy_coef	0.001
	max_grad_norm	0.5	gamma	0.99	lambda	0.95	save_rate	1000
	one_hot	true	phase	true	train_model	true	steps	3600
	test_model	true	test_step	3600	max_vehicle	1	action_interval	10

Table 15 (continued)

Model parameters								
PressLight	d_dense	20	n_layer	2	normal_factor	20	patience	10
	learning_rate	0.001	learning_start	1000	graphic	false	buffer_size	5000
	batch_size	64	episodes	200	epsilon	0.1	epsilon_decay	0.995
	epsilon_min	0.01	update_model_rate	1	update_target_rate	10	save_rate	20
	one_hot	true	phase	true	gamma	0.95	steps	3600
	test_steps	3600	vehicle_max	1	grad_clip	5	train_model	False
	test_model	true	action_interval	10				
FRAP	d_dense	20	n_layer	2	one_hot	false	phase	True
	learning_rate	0.001	learning_start	1000	graphic	false	buffer_size	5000
	rotation	true	conflict_matrix	true	merge	multiply	demand_shape	1
	batch_size	64	episodes	200	epsilon	0.1	epsilon_decay	0.995
	epsilon_min	0.01	update_model_rate	1	update_target_rate	10	save_rate	20
	test_steps	3600	test_model	true	action_interval	10	train_model	True
MPLight	d_dense	20	n_layer	2	one_hot	false	phase	True
	learning_rate	0.001	learning_start	-1	graphic	false	buffer_size	10000
	rotation	true	conflict_matrix	true	merge	multiply	demand_shape	1
	batch_size	32	episodes	200	eps_start	1	eps_end	0
	eps_decay	220	target_update	500	gamma	0.99	save_rate	20
	test_steps	3600	test_model	true	action_interval	10	train_model	True

Table 15 (continued)

Model parameters								
CoLight	neighbor_num	4	neighbor_edge_num	4	n_layer	1	input_dim	[128,128]
	output_dim	[128,128]	node_emb_dim	[128,128]	num_heads	[5,5]	node_layer_dims_each_head	[16,16]
	output_layers	[]	learning_rate	0.001	learning_start	1000	graphic	True
	buffer_size	5000	batch_size	64	episodes	200	epsilon	0.8
	epsilon_decay	0.9995	epsilon_min	0.01	update_model_rate	1	update_target_rate	10
	save_rate	20	one_hot	true	phase	false	gamma	0.95
	steps	3600	test_steps	3600	vehicle_max	1	grad_clip	5
	train_model	false	test_model	true	action_interval	10		
Server parameters								
pc1	CPU	Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz			cpu cores	24	Mem total	251.55GB
pc2	CPU	Intel(R) Xeon(R) Platinum 8124 M CPU @ 3.00GHz			cpu cores	18	Mem total	251.54GB
Training parameters								
Thread	4	ngpu	-1	action_pattern		“set”	if_gui	True
Debug	False	Interval	1	Savereplay		True	rltrafficlight	True

Author contribution All authors contributed to the experimental design. Code and experiments are written and conducted by HM, XL, and HW. Documentation is provided by HM, XL, and LD. The manuscript was written by HM, XL, BS, and HW. All authors read and approved the final manuscript.

Funding The work was supported by NSF award #2153311.

Data availability All dataset are publicly available at https://github.com/DaRL-LibSignal/LibSignal/tree/master/data/raw_data.

Code availability All code associated with this paper is publicly available from https://github.com/DaRL-LibSignal/LibSignal.

Declarations

Conflicts of interest Not applicable.

Ethical approval Not applicable.

Consent to participate Hao Mei agrees to participate. Xiaoliang Lei agrees to participate. Longchao Da agrees to participate. Bin Shi agrees to participate. Hua Wei agrees to participate.

Consent for publication Hao Mei agrees that his individual data and image are published. Xiaoliang Lei agrees that her individual data and image are published. Longchao Da agrees that his individual data and image are published. Bin Shi agrees that his individual data and image are published. Hua Wei agrees that his individual data and image are published.

References

Ault, J., & Sharon, G. (2021). Reinforcement learning benchmarks for traffic signal control. In 35th Conference on neural information processing systems datasets and benchmarks track (Round 1).

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym

Cao, M., Li, V. O., & Shuai, Q. (2022). A gain with no pain: Exploring intelligent traffic signal control for emergency vehicles. *IEEE Transactions on Intelligent Transportation Systems, 23*(10), 17899–17909.

Chen, C., Wei, H., Xu, N., Zheng, G., Yang, M., Xiong, Y., Xu, K., & Li, Z. (2020). Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In Proceedings of the AAAI conference on artificial intelligence (vol. 34, pp. 3414–3421).

Chu, T., Wang, J., Codecà, L., & Li, Z. (2019). Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems, 21*(3), 1086–1095.

Devailly, F.-X., Larocque, D., & Charlin, L. (2021). Ig-rl: Inductive graph reinforcement learning for massive-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems, 23*(7), 7496–7507.

Kheterpal, N., Parvate, K., Wu, C., Kreidieh, A., Vinitsky, E., & Bayen, A. (2018). Flow: Deep reinforcement learning for control in sumo. *EPiC Series in Engineering, 2*, 134–151.

Lopez, P.A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., & Wießner, E. (2018). Microscopic traffic simulation using sumo. In 2018 21st international conference on intelligent transportation systems (ITSC) (pp. 2575–2582). IEEE.

Ma, J., & Wu, F. (2020). Feudal multi-agent deep reinforcement learning for traffic signal control. In Proceedings of the 19th international conference on autonomous agents and multiagent systems (AAMAS) (pp. 816–824).

Oroojlooy, A., Nazari, M., Hajinezhad, D., & Silva, J. (2020). Attendlight: Universal attention-based reinforcement learning model for traffic signal control. *Advances in Neural Information Processing Systems, 33*, 4079–4090.

Peng, X.B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In 2018 IEEE international conference on robotics and automation (ICRA) (pp. 3803–3810). IEEE.

Raeis, M., & Leon-Garcia, A. (2021). A deep reinforcement learning approach for fair traffic signal control. In 2021 IEEE international intelligent transportation systems conference (ITSC) (pp. 2512–2518). IEEE.

Rasheed, F., Yau, K.-L.A., Noor, R. M., Wu, C., & Low, Y.-C. (2020). Deep reinforcement learning for traffic signal control: A review. *IEEE Access, 8*, 208016–208044.

Reinforcement Learning for Traffic Signal Control. https://traffic-signal-control.github.io/. Accessed 22 May 2022.

Rizzo, S.G., Vantini, G., & Chawla, S. (2019). Reinforcement learning with explainability for traffic signal control. In 2019 IEEE intelligent transportation systems conference (ITSC) (pp. 3567–3572). IEEE.

Rizzo, S.G., Vantini, G., & Chawla, S. (2019). Time critic policy gradient methods for traffic signal control in complex and congested scenarios. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 1654–1664).

Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L. S., Dieffendahl, C., Horsch, C., Perez-Vicente, R., et al. (2021). Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems, 34*, 15032–15043.

Tran, T.V., Doan, T.-N., & Sartipi, M. (2021). Tslib: A unified traffic signal control framework using deep reinforcement learning and benchmarking. In 2021 IEEE international conference on big data (Big Data) (pp. 1739–1747). https://doi.org/10.1109/BigData52589.2021.9671993

Wang, M., Wu, L., Li, J., & He, L. (2021). Traffic signal control with reinforcement learning based on region-aware cooperative strategy. *IEEE Transactions on Intelligent Transportation Systems, 23*(7), 6774–6785.

Wei, H., Chen, C., Zheng, G., Wu, K., Gayah, V., Xu, K., & Li, Z. (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 (pp. 1290–12980).

Wei, H., Xu, N., Zhang, H., Zheng, G., Zang, X., Chen, C., Zhang, W., Zhu, Y., Xu, K., & Li, Z. (2019). 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)

Wei, H., Zheng, G., Gayah, V., & Li, Z. (2019). A survey on traffic signal control methods. arXiv preprint arXiv:1904.08117

Wei, H., Zheng, G., Yao, H., & Li, Z. (2018). Intellilight: A reinforcement learning approach for intelligent traffic light control. In Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 2496–2505).

Wei, H., Zheng, G., Gayah, V., & Li, Z. (2021). Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation. *ACM SIGKDD Explorations Newsletter, 22*(2), 12–18.

Wu, L., Wang, M., Wu, D., & Wu, J. (2021). 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).

Xiong, Y., Zheng, G., Xu, K., & Li, Z. (2019). Learning traffic signal control from demonstrations. In Proceedings of the 28th ACM international conference on information and knowledge management (pp. 2289–2292).

Xu, B., Wang, Y., Wang, Z., Jia, H., & Lu, Z. (2021). Hierarchically and cooperatively learning traffic signal control. In Proceedings of the AAAI conference on artificial intelligence (vol. 35, pp. 669–677).

Yau, K.-L.A., Qadir, J., Khoo, H. L., Ling, M. H., & Komisarczuk, P. (2017). A survey on reinforcement learning models and algorithms for traffic signal control. *ACM Computing Surveys (CSUR), 50*(3), 1–38.

Yen, C.-C., Ghosal, D., Zhang, M., & Chuah, C.-N. (2020). A deep on-policy learning agent for traffic signal control of multiple intersections. In 2020 IEEE 23rd international conference on intelligent transportation systems (ITSC) (pp. 1–6). IEEE.

Zang, X., Yao, H., Zheng, G., Xu, N., Xu, K., & Li, Z. (2020). Metalight: Value-based meta-reinforcement learning for traffic signal control. In Proceedings of the AAAI conference on artificial intelligence (vol. 34, pp. 1153–1160).

Zhang, H., Feng, S., Liu, C., Ding, Y., Zhu, Y., Zhou, Z., Zhang, W., Yu, Y., Jin, H., & Li, Z. (2019). Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. In The world wide web conference (pp. 3620–3624).

Zhang, H., Liu, C., Zhang, W., Zheng, G., & Yu, Y. (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 (pp. 1783–1792).

Zhao, W., Queralta, J.P., & Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: A survey. In 2020 IEEE symposium series on computational intelligence (SSCI) (pp. 737–744). IEEE.

Zheng, G., Xiong, Y., Zang, X., Feng, J., Wei, H., Zhang, H., Li, Y., Xu, K., & Li, Z. (2019). Learning phase competition for traffic signal control. In Proceedings of the 28th ACM international conference on information and knowledge management (pp. 1963–1972).

Zheng, G., Zang, X., Xu, N., Wei, H., Yu, Z., Gayah, V., Xu, K., & Li, Z. (2019). Diagnosing reinforcement learning for traffic signal control. arXiv . https://doi.org/10.48550/ARXIV.1905.04716. https://arxiv.org/abs/1905.04716

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.