IMPROVING TRAFFIC MANAGEMENT EFFICIENCY THROUGH REINFORCEMENT LEARNING-BASED TRAFFIC SIGNAL CONTROL AND CITYWIDE TRANSIT SIMULATION

by

Toan V. Tran

IMPROVING TRAFFIC MANAGEMENT EFFICIENCY THROUGH REINFORCEMENT LEARNING-BASED TRAFFIC SIGNAL CONTROL AND CITYWIDE TRANSIT SIMULATION

by

Toan V. Tran

A Thesis Submitted to the Faculty of the University of
Tennessee at Chattanooga in Partial
Fulfillment of the Requirements of the Degree of
Master of Science: Computer Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

August 2023

ABSTRACT

Traffic congestion reduces productivity and harms the environment. Enhancing traffic signal control and public transportation are effective solutions. However, prior research has limitations stemming from the absence of real-time reliable data. Recent computer vision systems have made collecting traffic data easier. This thesis explores leveraging these data sources to enhance existing traffic signal controls (TSCs) and citywide transit simulations. For TSC, a comprehensive framework that facilitates rapid prototyping of reinforcement learning (RL) and an automatic feature engineering method are proposed. Additionally, RL techniques are implemented to a digital twin of Chattanooga smart corridor. Regarding transit simulations, a toolkit for calibrating large-scale simulations and an efficient solution for simulating changes in transit system settings are developed. Finally, we delve into a fundamental question of optimization for training neural networks and demonstrate that a novel approach using Neuroevolution outperforms Gradient Descent methods.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Dr. Mina Sartipi, for her unwavering support and guidance. I am also grateful to the UTC professors who offered courses that have enhanced my knowledge in Computer Science. Additionally, I would like to thank the SimCenter for providing computing resources that have been essential to my research. Finally, I kindly thanks to the committee members – Dr. Dalei Wu, Dr. Yu Liang, and Dr. Mina Sartipi for their valuable comments and suggestions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Traffic congestion is a major problem in many cities around the world. Despite massive investments in infrastructure, the increase in mobility demand within metropolitan areas has outstripped the capacity of the transportation network. This has led to a number of negative consequences, including loss of productivity, pollution and environmental damage, and poor health due to stress. According to the Urban Mobility report, traffic congestion in the United States caused 4.3 billion hours of delay, wasted 101 billion gallons of fuel, and damaged the economy by $101 billion in 2021 [3]. These costs are likely to increase in the future as traffic congestion continues to worsen. Two of the most simple but effective ways to reduce traffic congestion are to improve traffic signal control (TSC) and public transportation system.

However, the existing traffic signal control systems which are running in the real world are quite inefficient. Most of them follow static plans or limitedly adjust to data from loop detectors. On the public transportation side, due to a lack of citywide traffic data, there are not many studies that develop reliable simulations, which are an essential component of designing, planning, and optimizing routes research. Fortunately, the amount and speed at which traffic data is collected has increased significantly due to the widespread use of new technologies such as cameras, Internet of Things (IoT) devices, and vehicular networks. These technologies have made it easier to access real-time and reliable data [4]. That has created an emerging question – how to exploit these data to improve the existing TSCs and public transportation simulations.

1.1 Traffic Signal Control

About traffic signal control, the application of reinforcement learning to this field has been an area of active research in recent years. Reinforcement learning is a subfield of artificial intelligence that involves training agents to learn how to take actions in an environment to maximize a reward signal. One of the earliest studies in this area was conducted by [5], who developed a deep Q-learning algorithm for controlling a single traffic junction. The algorithm was able to learn optimal signal timings through trial and error, and outperformed fixed-time control, especially under varying traffic conditions. However, the algorithm was limited to a single junction and could not account for interactions with neighboring junctions. Since then, there have been several studies that have focused on developing reinforcement learning algorithms for coordinated traffic signal control across multiple intersections. One approach has been to use decentralized reinforcement learning, where each junction learns independently based on its local observations [6]. Another approach has been to use centralized reinforcement learning, where a central controller coordinates the signal timings of all junctions [7]. Overall, reinforcement learning-based traffic signal control has shown promising results in simple simulation settings, but there is still much work to be done to demonstrate its effectiveness in real-world settings.

1.2 Citywide transit simulation

Regarding to public transportation simulation which is a citywide problem, [8] and [9] proposed a method for creating a simulated city transit system based on various data sources like open street map, origin destination matrix, and general transit feed specification to deal with the public transportation simulation problem faced by the entire city. However, there is still a need for more research into calibrating city-level simulations, especially for large and complex simulations, in order to obtain accurate results [10]. Additionally, there is still room for more research into finding an efficient way to simulate diverse transit system settings.

1.3 Thesis objectives

This thesis investigates various problems in reinforcement learning for traffic signal control and public transit simulation. For TSC, we develop a unified framework named TSLib which is a Python framework for fast prototyping traffic signal control systems. It is designed to be modular and reusable, so that researchers can quickly implement and evaluate new ideas in TSC. TSLib includes a comprehensive implementation of some well-known TSC algorithms, as well as performance measurements. Subsequently, we investigate the feature engineering procedure of RL-based TSCs and propose a novel method named WorldLight using world models. WorldLight outperforms the previous methods in certain cases. Additionally, RL-based TSCs are implemented on a digital twin of the smart corridor at Chattanooga, TN, USA. Regarding public transit simulation, SIMCal which is a high-performance toolkit for calibrating traffic simulation is developed. It is designed to help researchers and practitioners to quickly and easily calibrate traffic simulations. SIMCal supports a variety of calibration methods, such as genetic algorithm, particle swarm optimization and firefly algorithm. Furthermore, we address a novel simulation problem – how to efficiently conduct a new simulation where only the transit system's setting (e.g., number of buses) changes while the rest of the transportation system remains stable. Finally, we delve into the fundamental question of optimization for neural network training and propose a novel approach using neuroevolution, which surpasses the widely popular Gradient Descent method.

Part I

Reinforcement Learning for Traffic Signal Control

CHAPTER 2

TSLib: A Unified Traffic Signal Control Framework Using Deep Reinforcement Learning

Though the extensive efforts on proposing RL-based TSC methods as mentioned in Chapter 1, previous research on traffic signal control has not produced scalable, flexible, and easy-to-use frameworks. Existing frameworks are typically designed for a specific traffic network, traffic light setting, TSC method, and simulator, making them difficult to replicate and apply to new contexts. Additionally, there are still challenges with TSC methods, such as benchmarking, learning efficiency, safety, and transferring from simulation to reality. These challenges need to be addressed in order to develop TSC methods that are effective, efficient, and safe.

We develop TSLib [11] which is a modular framework for traffic signal control problems. It is designed to be flexible and extensible, so it can be used in different simulators and real-world environments. The framework provides a collection of well-known TSCs and an intuitive interface, which makes it easy for researchers to implement new DRL-based TSC algorithms. TSLib also supports various settings to facilitate benchmarking. To demonstrate the strength of TSLib, the authors include many widely adopted and state-of-the-art TSC methods within the framework and conduct a benchmarking comparison on various criteria.

2.1 Brief Description of TSLib

Figure 2.1 shows the design of TSLib. The framework is divided into three components: the interface, the environment, and the controller. This separation of modules makes it easy to scale up the framework. The following sections describe each component in more

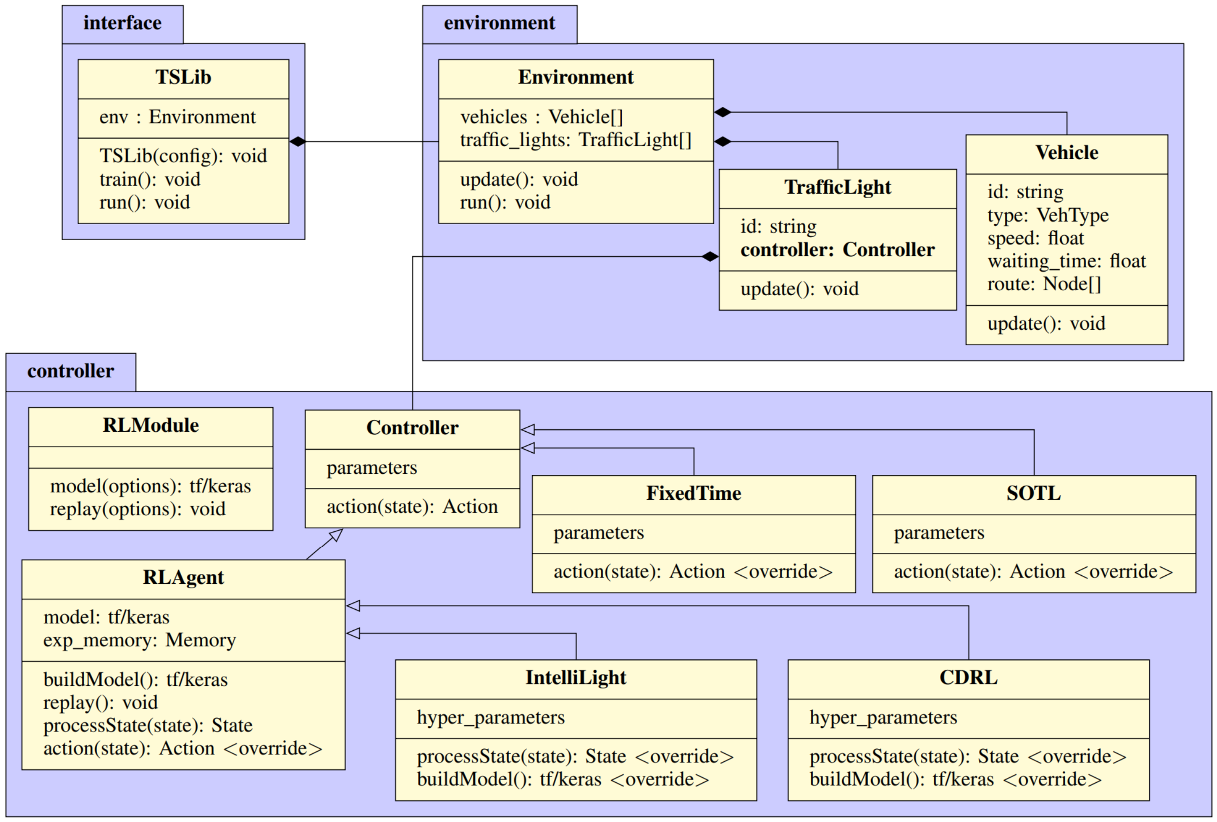


Figure 2.1 The class diagram of TSLib. To reuse code, we design inheritance relationships.

detail.

2.1.1 Interface

Users interact with TSLib through the interface class. This class provides a simple interface to configure simulation. There are two main functions including “train” and “run”. The function named “run” is used for inference stage or testing trained models. The below code is a python-interface example of applying available TSCs using our source:

```python
from tslib import TSLib

config = {
    "net": "atlanta/road.net.xml",
    "veh_type": "type.xml",
    "route": "atlanta/flow.route.xml",
 7     "traffic_lights": [
 8         {"node_id": "A", "method": "FixedTime"},
 9         {"node_id": "B", "method": "IntelliLight", "yellow_duration": 3, "cycle_control": 5,
      "model": "model/atlanta/IntelliLight.h5"},
10     ],
11     "log_folder": "log/atlanta",
12     "simulator" : "SUMO", # TSlib supports both SUMO and CityFlow
13     "gui": False,
14 }
15 
16 sim = TSLib(config)
17 sim.run() # to run simulation with trained models
18 #  sim.train() # to train and save models
```

Users must provide a JSON configuration file. The configuration file contains paths to files that define the traffic road structure, vehicle characteristics (e.g., maximum speed, length of cars), traffic flows, an array of traffic lights that determines which methods are applied to intersection nodes, an address for logs, and an option to enable the GUI. The example code shows how to do this. Currently, our source code supports both SUMO and CityFlow, as shown in Figures 2.2 and 2.3.

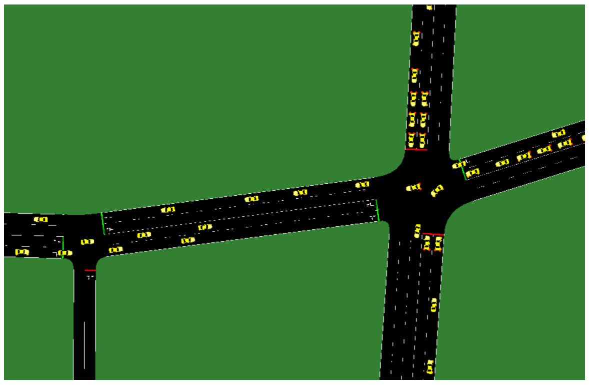


Figure 2.2 TSLib works in SUMO, a popular simulator in traffic engineering [1]

To train a model for DRL-based traffic signal controllers (TSCs), the configuration is the same as the one used to run a simulation. This is illustrated in lines 17 and 18 of the above example code. In addition, TSLib allows users to visualize metrics during training using Tensorboard. This can help users track the convergence of their TSC methods.

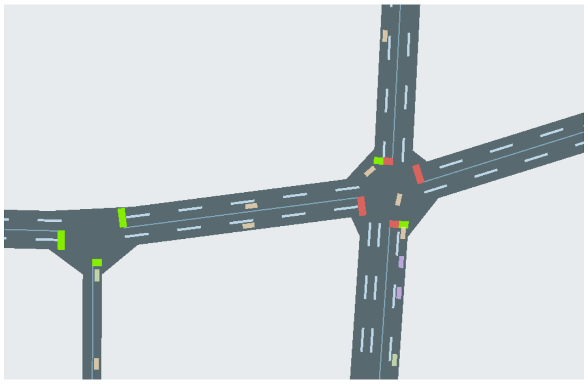


Figure 2.3 TSLib works in CityFlow, a high-computing-performance simulator [2]

2.1.2 Environment

The environment component in Figure 2.1 contains information about vehicles and traffic lights. This component is the only one that depends on the simulator being used. For example, the APIs for getting information about vehicles are different for different simulators. To support new environments, we need to customize the functions of this component to correspond to the new platform. This includes how to get information about vehicles (e.g., using the simulator's API, vehicular networks, or cameras). The most important class in this component is the "TrafficLight" class. This class contains a comprehensive state that includes information about the current vehicles in the traffic light's zone and the current phase setting.

2.1.3 Controller

The core of our library is the controller component. This component contains a class named RLAgent that includes all the necessary functions for a traffic signal controller (TSC) using reinforcement learning (DRL). To implement a new DRL-based TSC, users can inherit this class and save a lot of work, instead of building it from scratch. For example, we provide configurable RL modules that help users easily implement new ideas for DRL-based TSCs. Basically, the two classes "RLModule" and "RLAgent" serve as a small version of a reinforcement learning library.

Users can also implement their own functions for the state, such as implementing the function named “processState” that inputs the comprehensive state from the “TrafficLight” class. In particular, users can reuse the existing code for state and reward designs of available TSCs.

In other words, the controller component is the heart of our library. It provides a lot of functionality that users can use to implement new DRL-based TSCs. Users can save a lot of time and effort by inheriting the “RLAgent” class and using the configurable RL modules. Users can also implement their own functions for the state and reward designs. For example, the below example is used to build a new TSC using Deep Q Network (DQN):

```python
from rlmodules import RLModule
from RLAgent import RLAgent
from CDRL import CDRL
from VFB import VFB

class New_TSC(RLAgent):
    def __init__(self, config, road_structure, phase_config):
        RLAgent.__init__(self, config, road_structure)
        self.incoming_lanes = getIncomingLanes(road_structure)
        self.num_phases = getNumOfPhases(phase_config)

    def buildModel(self):
        return RLModule.buildModel(type='DQN',input_shape=(len(self.incoming_lanes),),
    action_space=self.num_phases)

    def processState(self, state):
        return state['queue_length']

    def computeReward(self, state, historical_data):
        return CDRL.computeReward(state, historical_data)
```

This example shows how to use the TSLib library to create a new traffic signal controller (TSC). The example inherits the “RLAgent” class, uses the “RLModule” module to build a new model, and defines the state as the queue length. The example also reuses the reward function from another method called Coordinated Deep Reinforcement Learners (CDRL) [5]. We believe that TSLib can significantly reduce the workload of implementing

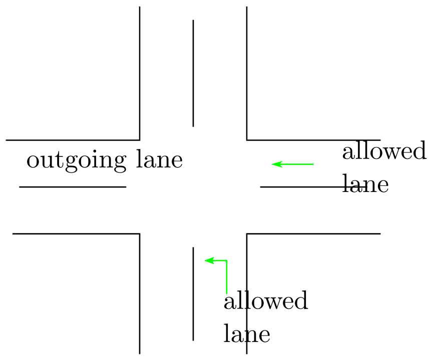


Figure 2.4 An example of outgoing and allowed lanes of a phase that allows two movements as green arrows

new TSCs by providing these features.

2.2 Benchmarks

2.2.1 Methods

We consider the following methods in our benchmarks:

- **Fixed Time control (FT)**: FT is a static control which follows pre-defined fixed-time-phase configurations.

- **Self-Organizing Traffic Light control (SOTL)** [12]: SOTL is an adaptive method that uses only the volume of lanes as the input.

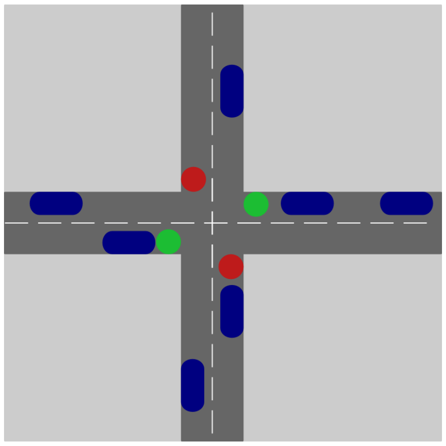
- **Max-Pressure control (MP)** [13]: The MaxPressure algorithm calculates the pressure of all phases in a traffic signal system. At each cycle, the phase with the highest pressure is activated. The pressure of a phase is calculated by the difference between the number of vehicles on allowed lanes and the number of vehicles on outgoing lanes.

Table 2.1 Rules of SOTL

Request by green-phase lanes	Request by red-phase lanes	Action
Yes	Yes	Keep the current phase
	No	
No	Yes	Change to the next phase
	No	Keep the current phase

- **Coordinated Deep Reinforcement Learners (CDRL) for traffic light con-**

trol [5]: CDRL is a RL-based method using deep Q learning. It represents the state as an image which describes comprehensively the position of vehicles, illustrated in Figure 2.5. The action of CDRL is a phase index which will be executed in the next interval. The reward is a linear function which combines multiple objectives – minimizing signal changes, delay, and waiting time.



(a) Traffic situation

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.2 & 0.8 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) State representation in 8 x 8 matrix

Figure 2.5 An example of CDRL's state representation

- **Traffic light control using deep policy-gradient and Value-Function Based reinforcement learning (VFB) for traffic light control** [14]: VFB has the same state representation and action design to CDRL's. The differences are the reinforcement learning method and reward function. VFB implements value-function methods and considers the total cumulative delay as its reward.

- **IntelliLight** [6]: The authors designed the action as switching to the next phase or keeping the current phase. Moreover, IntelliLight implements a phase-gate architecture which inputs both extracted features and vehicle-position images.

- **A deep reinforcement learning network for Traffic Light Cycle Control (TLCC)** [15]: TLCC applies an action to adjust the green time of phases. It implements Double dueling Deep Q-network, presented in Figure 2.7. The state representation is an image that includes both vehicle positions and speeds. The reward is the total waiting time of all vehicles.

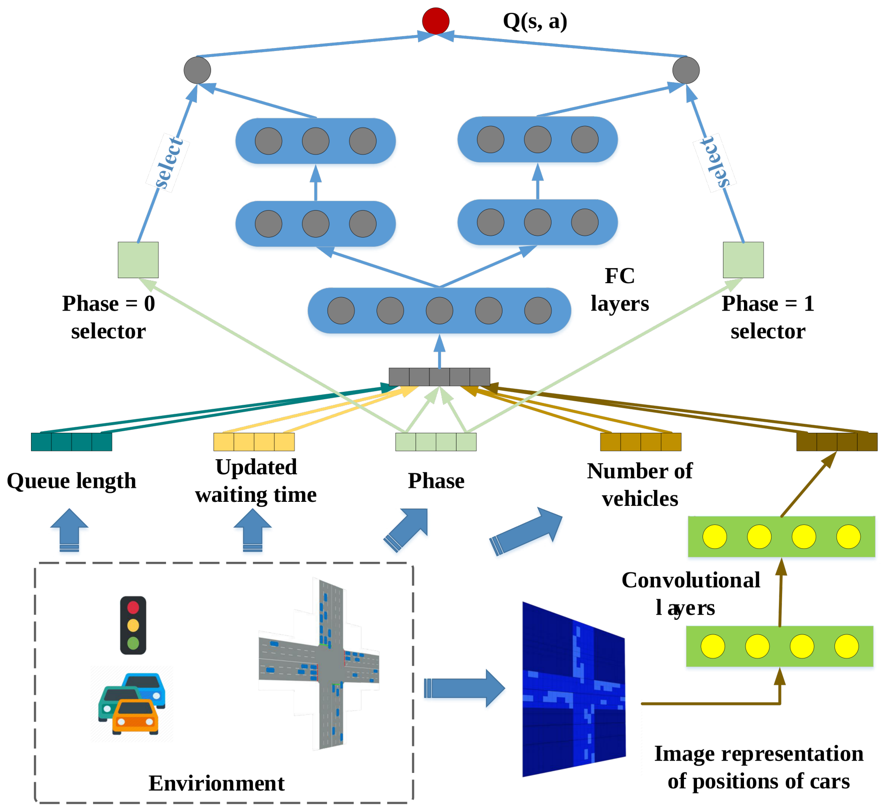


Figure 2.6 IntelliLight's Q network

2.2.2 Experiments

We conducted our experiments using the SUMO simulator [1], which is a popular tool for this type of research. To ensure a fair comparison, we used an identical parameter setting and same experience replay strategy for all of the DRL-based methods.

TSLib offers a variety of metrics. These metrics are used to measure the performance of individual lanes, intersections, and vehicles. For lanes and intersections, TSLib measures the following characteristics:

- Queue length: The number of vehicles waiting to enter the lane or intersection.
- Lane speed: The average speed of vehicles traveling in the lane.
- Lane waiting time: The average amount of time vehicles spend waiting to enter the lane.

For vehicles, TSLib measures the following features:

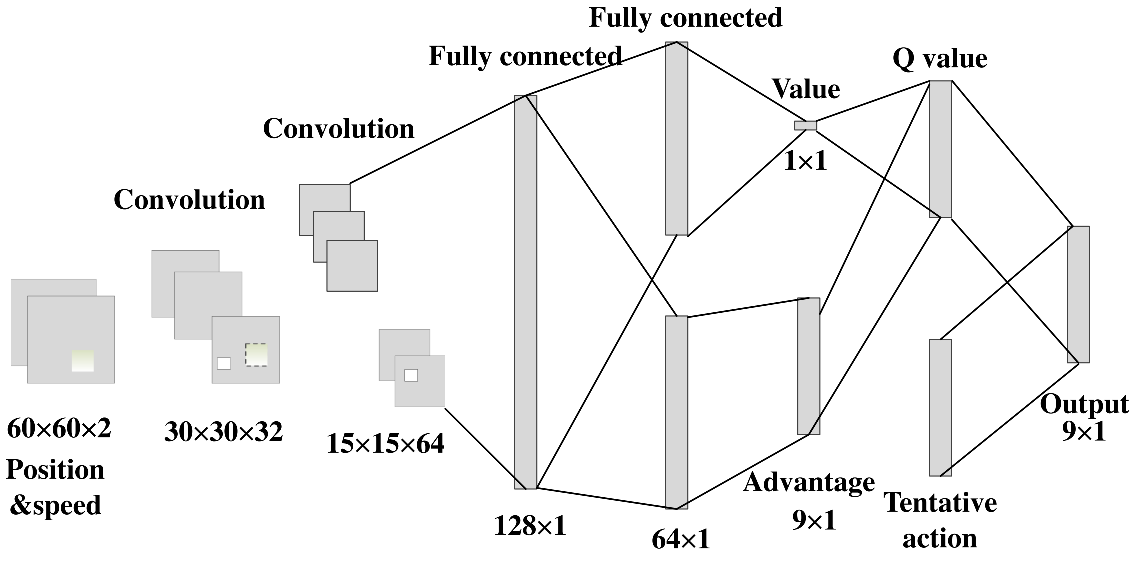


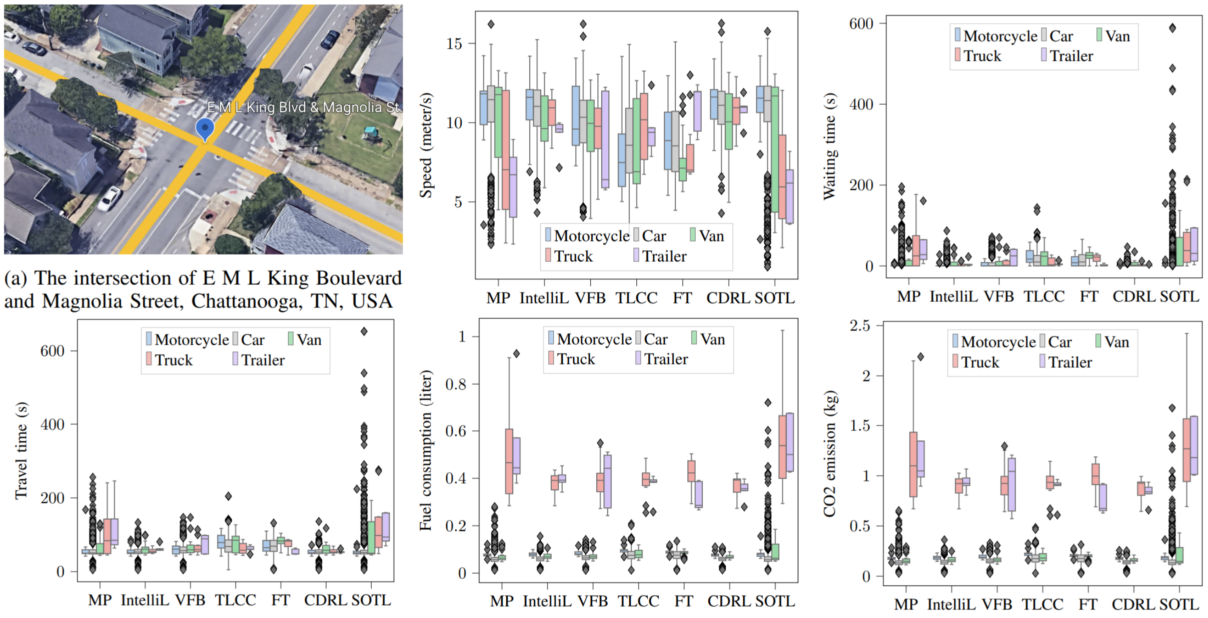
Figure 2.7 3DQN of TLCC

- Speed: The average speed of the vehicle.
- Travel/waiting time: The total amount of time the vehicle spends traveling or waiting.
- Fuel consumption: The amount of fuel used by the vehicle.
- CO/CO2 emission: The amount of CO/CO2 emitted by the vehicle.

By measuring these metrics, TSLib can provide a comprehensive assessment of the performance of transportation systems. This information can be used to identify areas where improvements can be made to improve the efficiency and sustainability of transportation systems.

1) Isolated intersection. We simulate a single real-world intersection in SUMO. The data for the intersection was collected from cameras at the intersection of Martin Luther King Boulevard and Magnolia Street in Chattanooga, TN, USA (illustrated in Figure 2.8) during the peak hour from 4pm to 5pm on February 13, 2021.

The seven methods are compared on their performance for different vehicle types. Figure 2.8 provides the performance comparison of all the methods. Generally, the DRL-based methods (IntelliLight, VFB, and CDRL) outperform the heuristic methods after 50 episodes of training. CDRL performs the best on all metrics. TLCC does not perform as



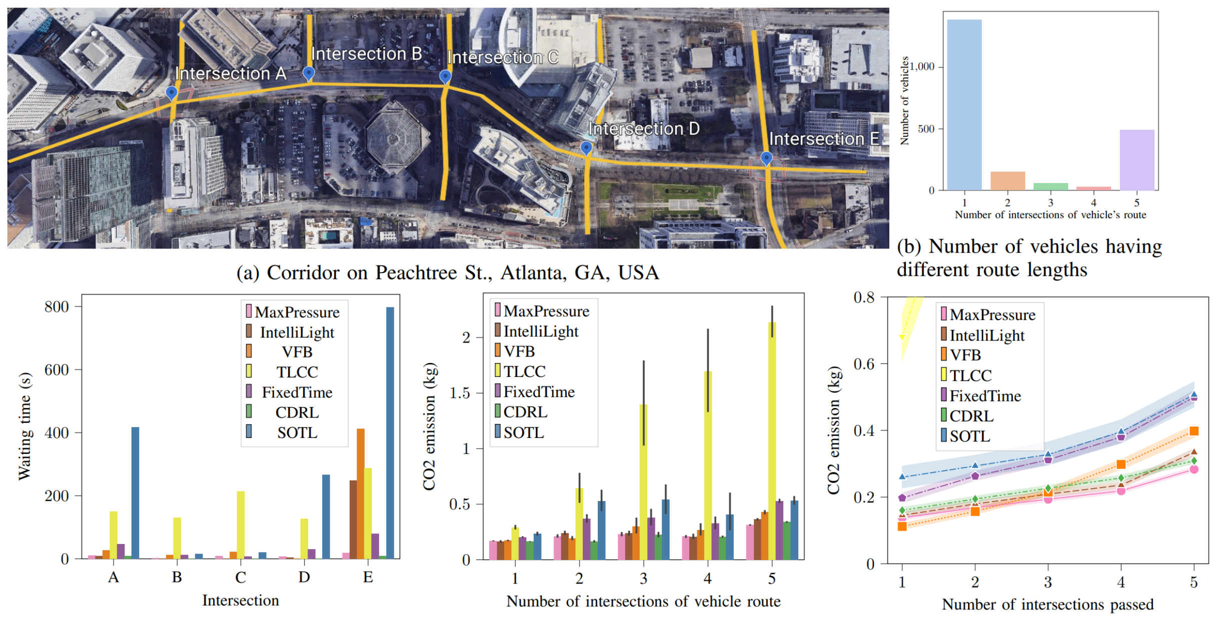
(a) The intersection of E M L King Boulevard and Magnolia Street, Chattanooga, TN, USA

Figure 2.8 An real-world intersection and performance of methods on different vehicle types in Chattanooga, TN, USA

well, possibly because it has not converged yet. The original paper [15] showed that TLCC converged after 600 episodes, but in this study it is only trained for 50 episodes for a fair comparison. For a one-hour workload, CDRL saves a total of 5.58 hours, an average of 15.3 seconds per driver, and reduces 48.3 kg of CO2 emissions compared to FixTime.

2) Corridor. The data of this experiment was collected on five intersections on Peachtree Street in Atlanta, GA, on November 8, 2006 by [16], [17]. The data was collected by eight cameras between 12:45 p.m. and 1:00 p.m. and again between 4:00 p.m. and 4:15 p.m. The intersections have different traffic volumes, and intersection B is a three-way intersection.

The average waiting time at each intersection is shown in Figure 2.9(c). Similar to the previous experiment, TLCC does not seem to have converged after 50 episodes. Other DRL-based methods are still improving. In this workload, MaxPressure has competitive performance compared to other DRL-based methods. The comprehensive data recorded by TSLib helps us understand the effects of vehicle characteristics such as vehicle type and vehicle route. Figure 2.9(d) shows the CO2 emissions of vehicles with different route lengths.



(a) Corridor on Peachtree St., Atlanta, GA, USA

(b) Number of vehicles having different route lengths

(c) Waiting time at intersections using different methods

(d) CO2 emission of vehicles having different route lengths and methods

(e) CO2 emission of vehicles having the route length of five intersections

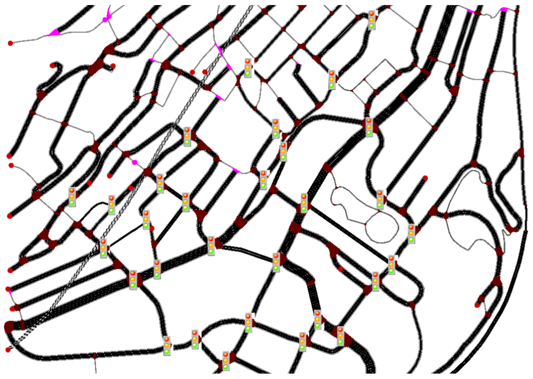
Figure 2.9 Performance on five intersections and effects of methods on different vehicle's route lengths

For vehicles with routes that include one to four intersections, CDRL is the best method for CO2 emissions. TLCC causes vehicles with a route length of 5 intersections to emit around 2kg of CO2, while the amount of CO2 emissions caused by other methods is less than 0.5kg. Furthermore, for vehicles with routes that include more than 5 intersections, TLCC has the highest CO2 emissions.

3) Downtown intersections. In this experiment, TSLib is used to control traffic flows in a downtown area in Monaco. The dataset used for this experiment was collected by [18] and included activity-based mobility data for one hour. The test site had a variety of traffic network characteristics that TSLib had to handle, such as different numbers of lanes, one-way roads, two-way roads, and different phase settings of traffic lights. TSLib is able to handle these different traffic network characteristics. This demonstrates the versatility and flexibility of TSLib.

The results of traffic signal control methods on the test site in Monaco are presented in Figure 2.10. Deep reinforcement learning (DRL)-based methods outperform heuristic

(a) Downtown intersections in Monaco


(b) Downtown intersections of Monaco simulated in SUMO

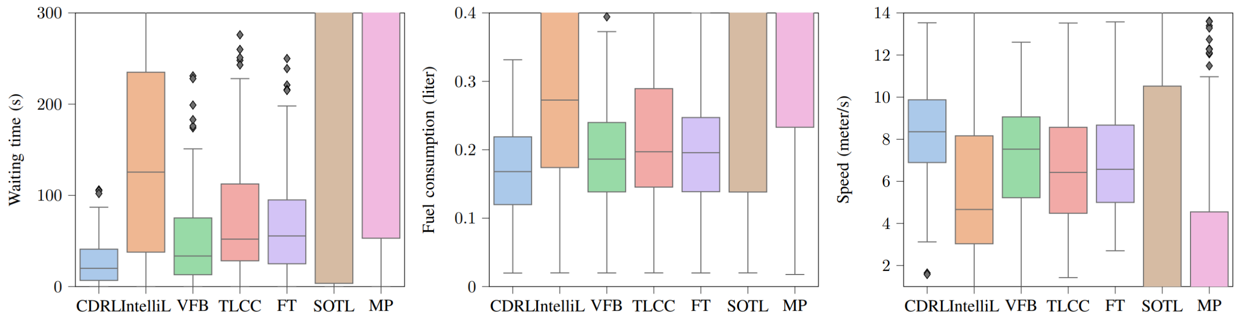


Figure 2.10 Performance on downtown intersections in Monaco

methods. SOTL and MaxPressure can not handle many different phase settings in a large network. Among DRL methods, CDRL is the best, saving around 30 seconds per driver compared to FixedTime. TLCC is the most inefficient method in previous experiments, but it performs better than IntelliLight, SOTL, and MaxPressure in the city-wide network. IntelliLight is the least effective method in a large network, but it performs well in a small network. In other words, DRL-based methods are more effective than heuristic methods at controlling traffic in a large network. CDRL is the best DRL-based method, and TLCC is the most efficient heuristic method.

CHAPTER 3

Pixel-based Traffic Signal Controls using Reinforcement Learning with World Models

In addition to Chapters 1 and 2, reinforcement learning for traffic signal control can be categorized into two types: pixel-based and feature-based, illustrated in Figures 3.1 and 3.2 respectively. Pixel-based methods [5, 15] use images of traffic states as input to the RL controller, while feature-based methods [7, 19, 20] extract features from traffic states and input these features instead of the image itself.

More specifically, the first studies that have applied RL for TSC were pixel-based methods, but feature-based methods have become more popular in recent years because they are more efficient and can be used to control larger traffic networks. Moreover, the high-dimensional images are difficult for RL to learn during the trial-and-error process. Therefore, the pixel-based TSCs usually outperform both the feature-based methods [11, 21]. Despite good performance, the feature-based TSCs have some limitations. First, feature-based state representation lacks comprehensive information. Second, feature designing requires manual work. Finally, each existing feature design is for a particular reward function. For example, the authors [19] used different methods to study queue lengths, and they discovered that the number of vehicles in each lane was the most important factor affecting queue length.

We proposes a new pixel-based traffic state classification method called WorldLight. WorldLight differs from previous methods in that it does not only directly learn from traffic-state images. Instead, it uses a world model [22, 23] to learn a representation of the images. This representation is more comprehensive than previous representations, as it includes information on both the current and future traffic states. WorldLight can achieve competitive performance with feature-based TSC methods, and even outperforms them in some scenarios.

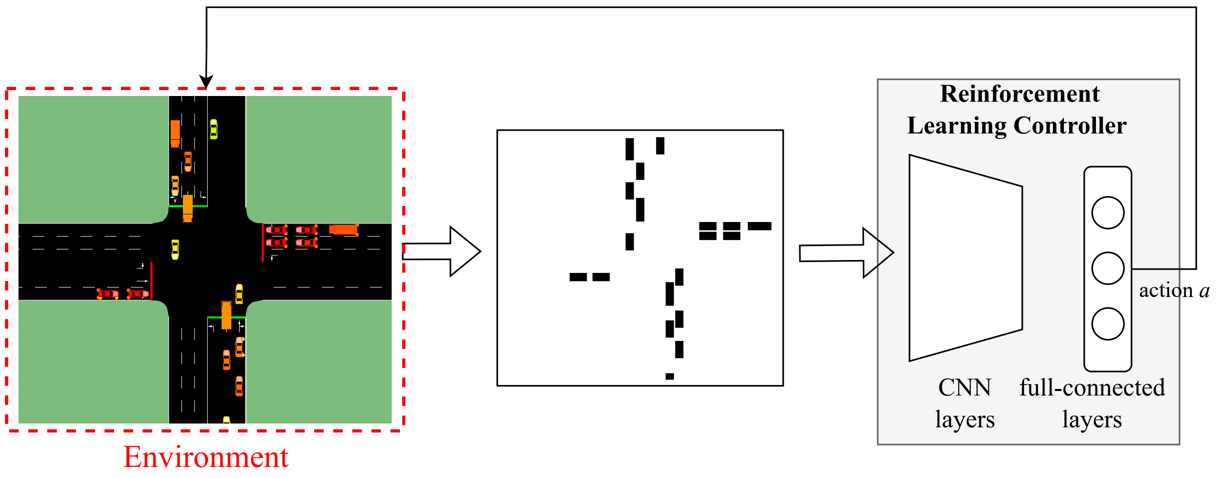


Figure 3.1 Pixel-based reinforcement learning for TSC

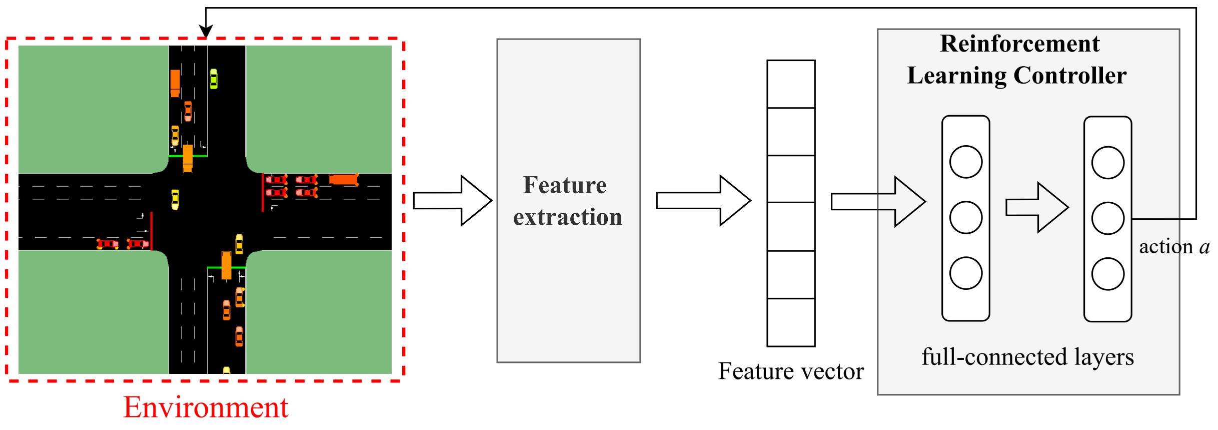


Figure 3.2 Feature-based reinforcement learning for TSC

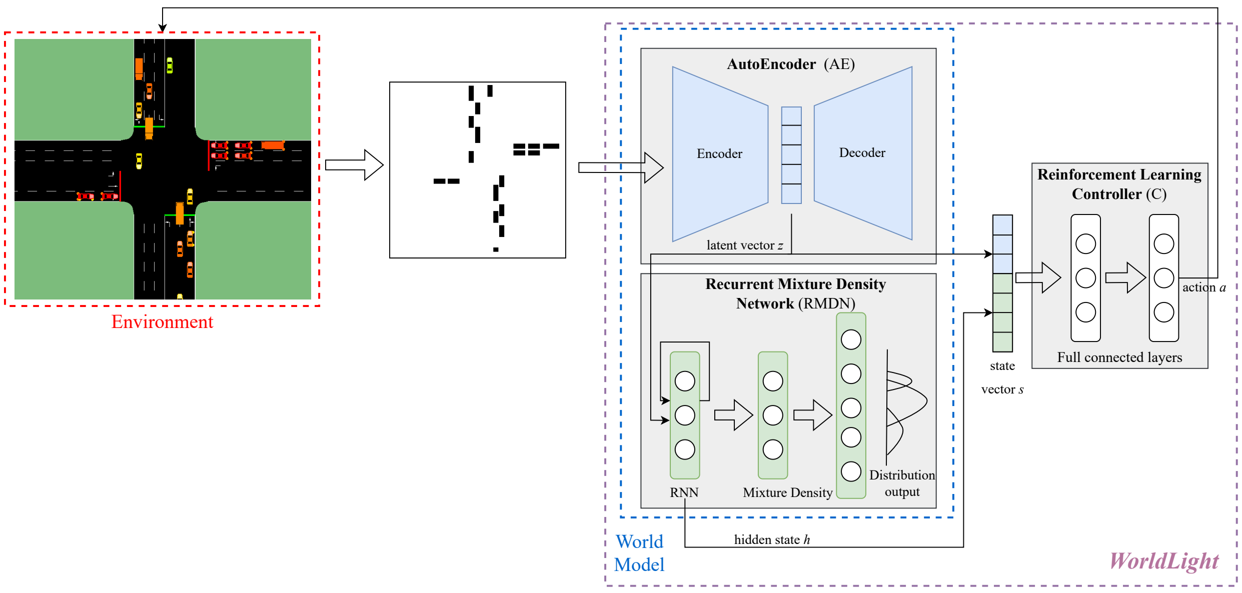


Figure 3.3 WorldLight's architecture

3.1 Proposed Methodology – WorldLight

3.1.1 WorldLight's components

WordLight is a pixel-based method for traffic signal control. It takes an image as input, which represents the current traffic status. WordLight then extracts features from the image using a world model, which consists of two main components: an AutoEncoder (AE) and a Recurrent Mixture Density Network (RMDN). The AE is used to represent the image as a latent vector z, which has a much smaller dimension than the original image. The RMDN then uses the latent vector z to predict the next latent vector z_{t+1} at time $t+1$. This process is repeated to generate a sequence of latent vectors, which can be used to predict future traffic conditions. More precisely, the RMDN is a model that uses RNN layers to model the time series of traffic data and a Mixture Density Network to understand the uncertainty of traffic. The world model creates a state vector that combines the latent vector z and the hidden state h. This state vector includes information about the current traffic status and the traffic modeling/prediction. The RL controller inputs the state vector and returns an optimal action. The RL controller implements two full connected layers because the raw image is extracted to features by the world model.

• ***AutoEncoder***. We implement a variational autoencoder that can compress high-dimensional traffic-state images into smaller latent vectors while preserving most of the information in the original images. Figure 3.4 shows how our autoencoder works. The reconstructed images are more accurate as the size of the latent vector increases. However, larger latent vectors can make it more difficult for the reinforcement learning (RL) controller to learn how to control the traffic.

• ***Recurrent Mixture Density Network***. The Recurrent Mixture Density Network (RMDDN) is a model that can predict future traffic conditions, i.e., the modeling task $P(z_{t+1}|a_t, z_t, h_t)$. It uses a Recurrent Neural Network (RNN) to understand time series patterns and a Mixture Density Network (MDN) to model traffic uncertainty. The RNN uses LSTM layers to learn the temporal dependencies in the traffic data, while the MDN outputs a set of Gaussian distributions for each element of the predicted traffic state. This allows the model to capture the uncertainty in the traffic data. The number of units in the last layer of RMDN is equal to the number of elements in the traffic state multiplied by the number of Gaussian distributions. RMDN's probability density function $p(x)$ is shown in Equation 3.1.

$$p(x) = \sum_{k}^{K} \pi_k \mathcal{N}(x|\mu_k, \sigma^2) \tag{3.1}$$

From the probability density function, we can train RMDN by minizing the log likelihood loss function $\mathcal{L}(w)$.

$$\mathcal{L}(w) = -\sum_{i=0}^{|z|} \ln \left(\sum_{k=1}^{K} \pi_k(w) \mathcal{N}\left(z_i|\mu_k(w), \sigma_k^2(w)\right) \right) \tag{3.2}$$

• ***Reinforcement Learning Controller***. Because the input of the controller is a 1D vector, the controller implements a standard full-connected neural network including 2 hidden layers with 64 units. The neural network is trained using the Proximal Policy Optimization (PPO) algorithm [24]. The controller takes a vector of traffic state information as input and outputs a phase index. The PPO algorithm is a policy gradient method for reinforcement learning that learns how to take actions in an environment in order to maximize a reward. In this case, the agent is learning how to choose a phase which will be executed within interval τ to reduce the total negative queue lengths of incoming lanes.

3.1.2 Training WorldLight

The training process for WorldLight is summarized as the follows:

1. Collect roll outs from a random policy/actuated control.
2. Train the AutoEncoder.
3. Train the Recurrent Mixture Density Network to model $P(z_{t+1}|a_t, z_t, h_t)$.
4. Train the Controller via a trial-and-error reinforcement learning process.

3.2 Experiments and Results

3.2.1 Experiment setting

We use the SUMO traffic simulator to simulate a real-world intersection in Chattanooga, TN, USA. The traffic demand for this intersection is obtained from a smart corridor [25] that continually captures traffic data such as the number of vehicles, arrival time of each vehicle, vehicle movement vehicle class, and vehicle length. By using this dataset, we are able to create a realistic simulation of the intersection.

We consider various of baselines including: Actuated Control which is running in the world; CNN-L which is a traditional pixel-based RL method [5]; MPLight [7] and LIT [19] which are feature-based RL methods.

3.2.2 Experiment 1: Overall performance

• ***Setting***. We conduct an experiment on a single intersection during the peak hour of 4pm to 5pm. The data was collected from the MLK Smart Corridor, so our simulation accurately represents the traffic flow distribution. Each method is trained using 250 episodes.

• ***Result***. Figure 3.5 shows the overall performance of each method used to control traffic flow. In general, all reinforcement learning (RL)-based methods outperform the actuated control method. Feature-based RL methods (LIT and MPLight) outperform the traditional pixel-based RL method (CNN-L). WorldLight is a pixel-based method, but it is better than the feature-based methods by using world models. Figure 3.5a shows the total queue length during the training process. WorldLight and CNN-L converge faster than LIT and MPLight. This is likely due to the different reinforcement learning algorithms used. WorldLight and CNN-L use proximal policy optimization (PPO), while LIT and MPLight use deep Q learning (DQN). Figures 3.5b, 3.5c, and 3.5d show the average travel time, queue length, and fuel consumption in the testing simulation. WorldLight is slightly better LIT and provides more stable performance across vehicles. Actuated is also stable because it is a rule-based strategy, while CNN-L can cause long delays for some vehicles. In conclusion, WorldLight is the best-performing RL-based method in this study. It outperform the other methods in terms of convergence speed, average travel time, queue length, and fuel consumption.

3.2.3 Experiment 2: Effects of reward function

• ***Setting***. In this experiment, we investigate performance of state representations of methods with different reward functions. The simulation setting remains the same as the one in Experiment 1 and each model is trained for 250 episodes. We considered four reward functions: one lane-based, one movement-based, and two vehicle-based. r_1 (Equation 3.3) and r_2 (Equation 3.4) are vehicle-based reward functions that consider waiting time and average speed, respectively.

$$r_1 = -\sum_{v \in V(t)} w(v), \tag{3.3}$$

$$r_2 = \sum_{v \in V(t)} \frac{s(v)}{|V(t)|}, \tag{3.4}$$

where $V(t)$ is the set of the vehicles which are in the intersection area at time t; $w(v)$ and $s(v)$ are the waiting time and the speed of vehicle v.

Subsequently, r_3 is a lane-based reward function considering queue lengths.

$$r_3 = -\sum_{l \in L^{in}} Q(l), \tag{3.5}$$

where L^{in} is a set of in-coming lanes of the intersection; $Q(l)$ is the queue length – number of waiting vehicles of lane l.

Finally, the movement-based reward function r_4 focuses on pressure.

$$r_4 = -\sum_{m_i \in M} p_{m_i}, \tag{3.6}$$

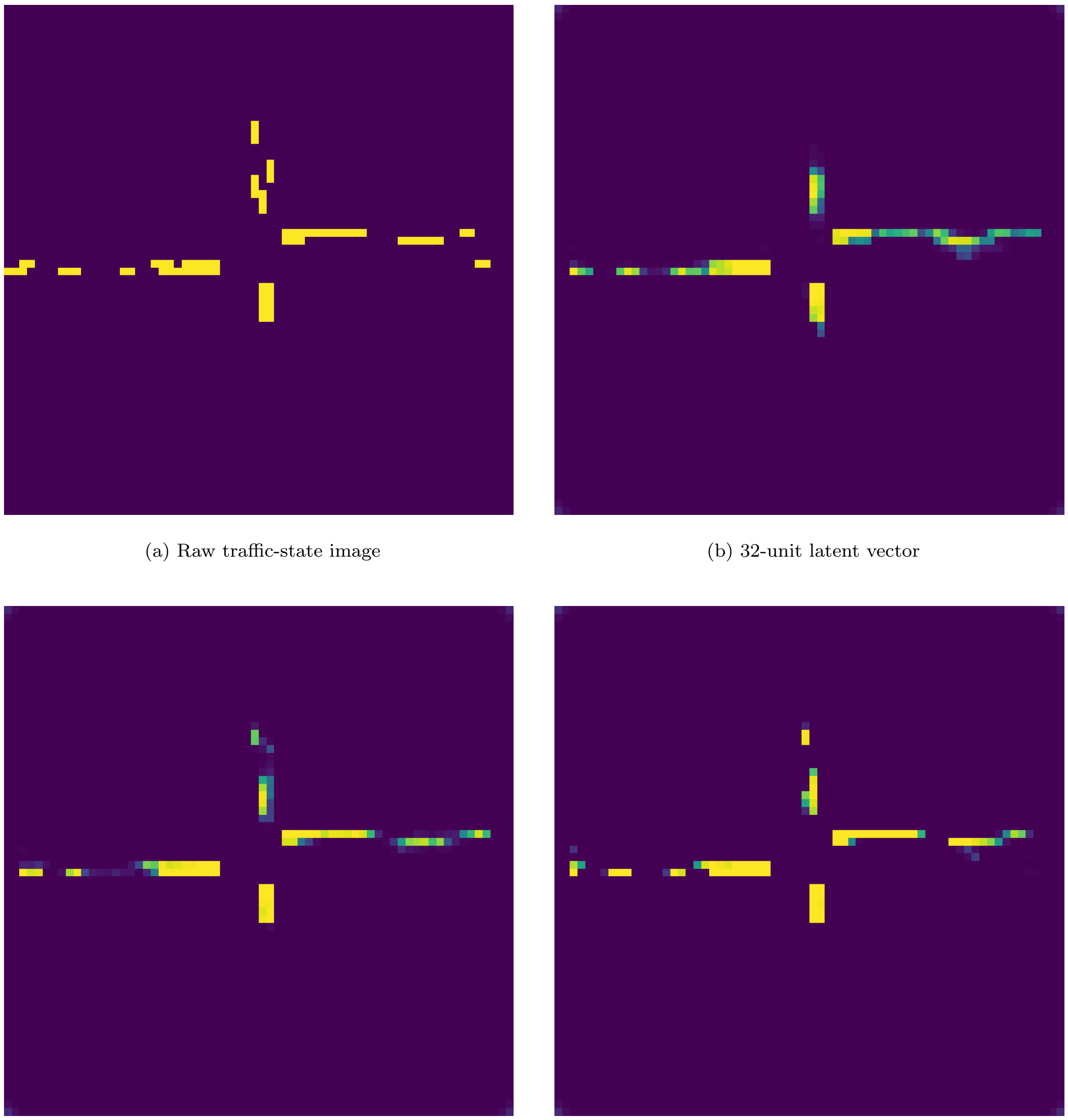
where M is the set of movements. There are usually 12 movements for a regular 4-leg intersection.

• ***Result***. Figure 6 shows the reward values of different methods. Feature-based methods outperform traditional pixel-based methods for all reward functions. This trend is consistent to Experiment 1 and previous studies [26]. However, our method with world models has narrowed the gap between feature-based and pixel-based strategies. WorldLight performed best for r3 (queue length) and r4 (pressure), while LIT was better than WorldLight for r1 (waiting time) and r2 (speed). This experiment shows that there is no single optimal state representation that works for all reward functions. This finding is consistent with previous research. Therefore, designing state representation is a critical task that depends on the objective (i.e., reward function).

3.2.4 Experiment 3: Effects of RL algorithms

• ***Setting***. In this experiment, we studied how WorldLight performs when using different reinforcement learning (RL) algorithms. The simulation settings are the same as in previous experiments. We investigated three common RL algorithms: Proximal Policy Optimization (PPO) [24], Advantage Actor Critic (A2C) [27], and Deep Q Learning (DQN) [28].

• ***Result***. Figure 3.7 shows how well WorldLight performs using different reinforcement learning (RL) algorithms. A2C is the fastest training method, converging after around 20 episodes. PPO and DQN require 50 and 80 episodes, respectively. In the testing phase, PPO outperforms A2C and DQN. For example, PPO reduces the average travel time of vehicles by 0.9% and 2.3% compared to A2C and DQN, respectively. Additionally, PPO increases the average speed by 0.6% and 1.9% compared to A2C and DQN. This experiment demonstrates that policy optimization methods (i.e., PPO and A2C) are better than the Q-learning method for WorldLight.



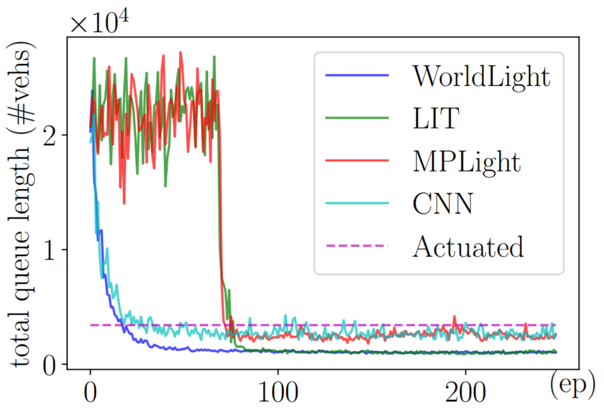
(a) Raw traffic-state image

(b) 32-unit latent vector

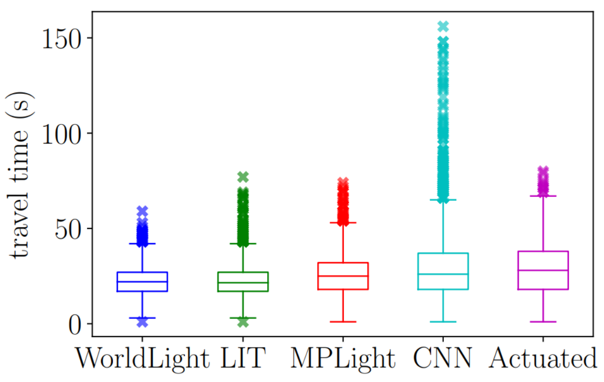
(c) 64-unit latent vector

(d) 128-unit latent vector

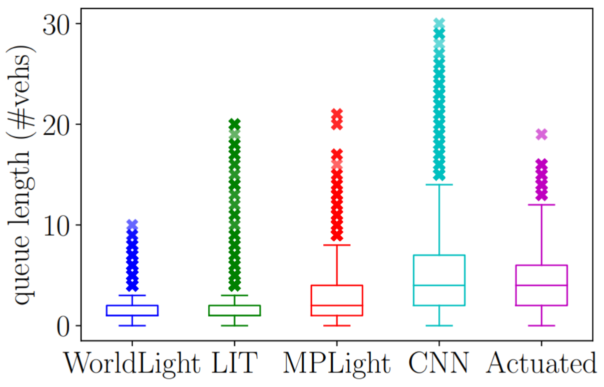
Figure 3.4 a raw traffic-state image; images constructed by the autoencoder with the latent vector size of 32, 64, and 128, respectively



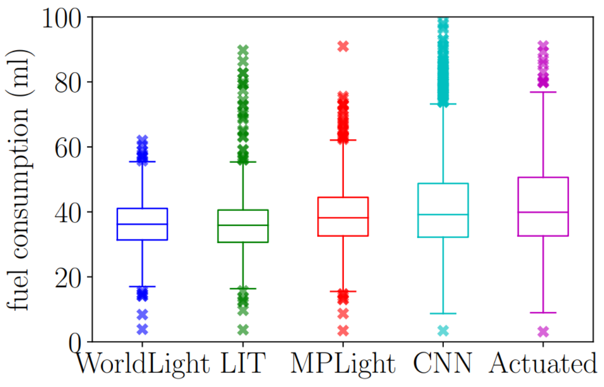
(a) Total queue length during training



(b) Average travel time of vehicles during testing



(c) Average queue length of steps during testing



(d) Average fuel consumption of vehicles during testing

Figure 3.5 Performance of WorldLight and previous methods

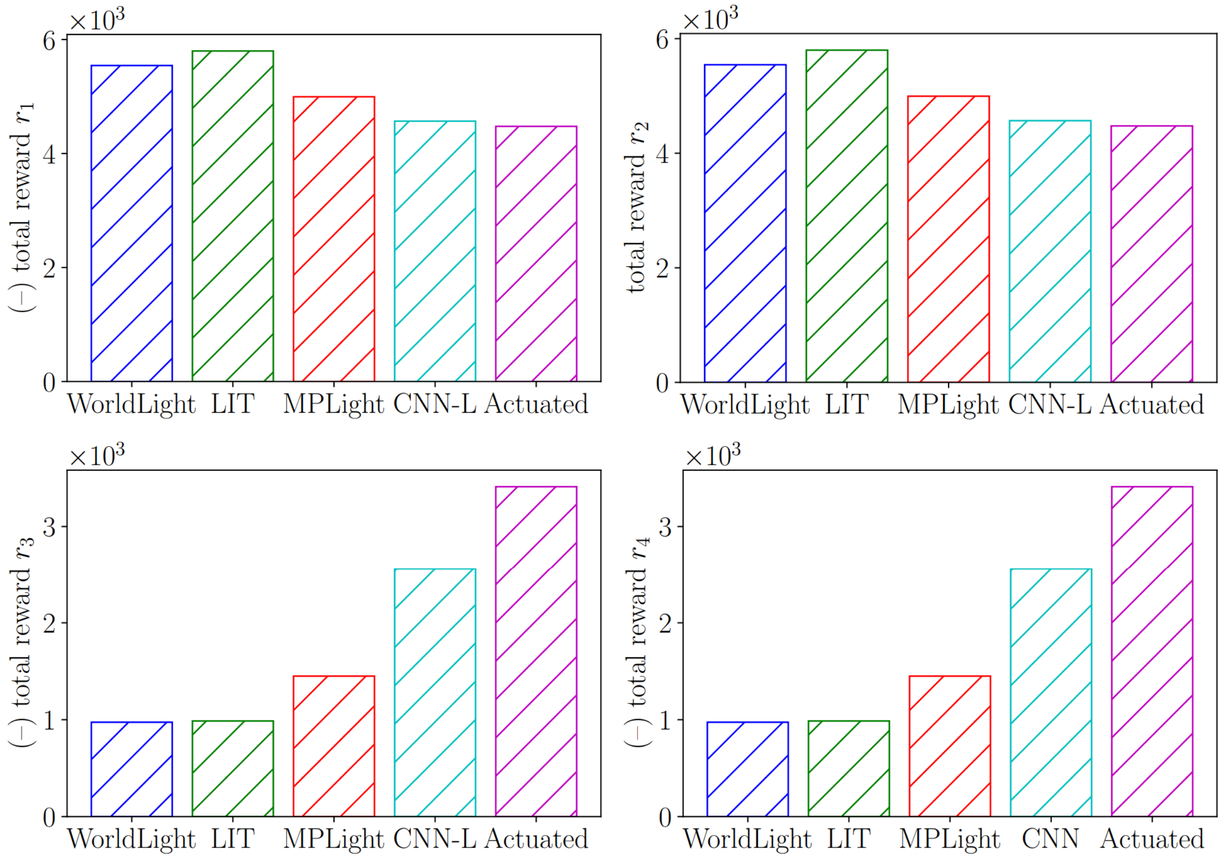
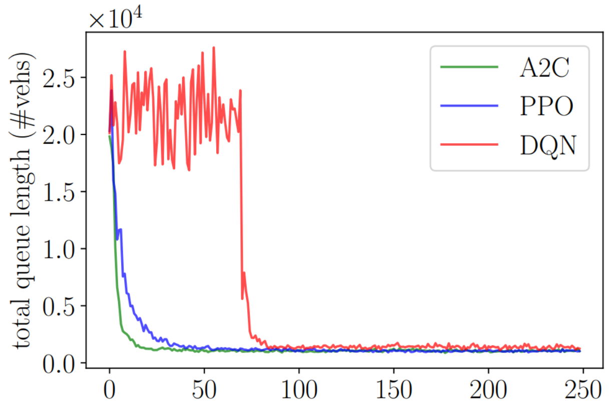
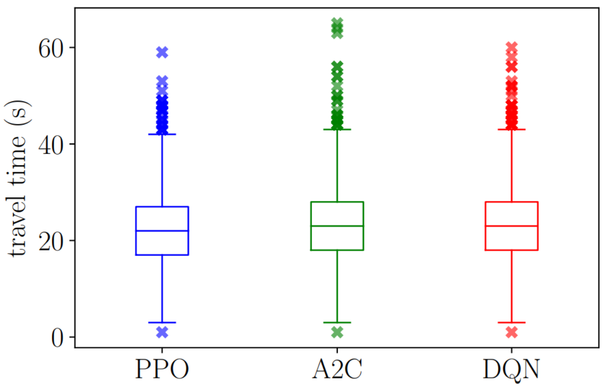


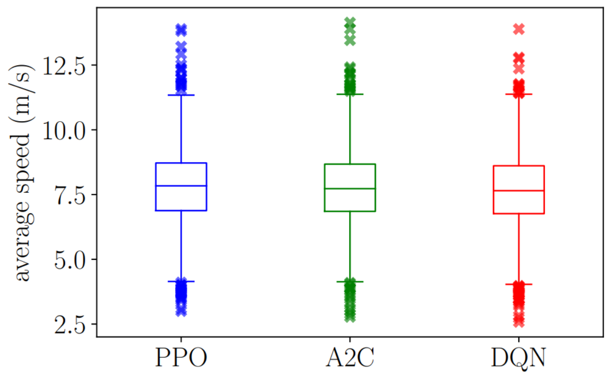
Figure 3.6 Performance of the state representation methods when using the same reward functions. For the negative total reward figures (i.e., $r1$, $r2$, and $r4$), the lower value is better. On the other hand, for the figure about $r3$, the higher value is better.



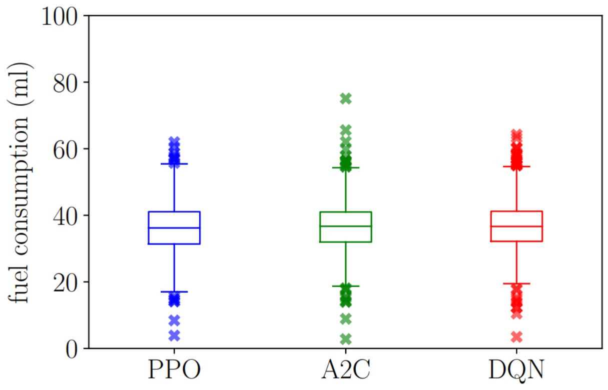
(a) Total queue length during training



(b) Average travel time of vehicles during testing



(c) Average speed of vehicles during testing



(d) Average fuel consumption of vehicles during testing

Figure 3.7 Performance of WorldLight when using different RL algorithms for the controller

CHAPTER 4
Optimizing MLK Traffic Controllers Using RL and Digital Twin

Despite of many previous studies investigating fundamentals of reinforcement learning for traffic signal control as mentioned in previous chapters, these studies were conducted using simulations that are not perfect representations of reality including the traffic volumes, driving behaviors, and implementation of the baseline algorithm.

Traffic volumes: Existing research on traffic control algorithms has assumed a fixed percentage of turning ratios among all approaching vehicles, such as 10% turning left, 60% going straight, and 30% turning right [29]. Additionally, most experiments have only guaranteed the correct setting of the number of vehicles, and assumed that vehicles approach intersections uniformly. However, in reality, the distribution of vehicles may not be uniform, as it is determined by adjacent intersections. This means that existing research may not accurately represent the workload distribution, which is critical for evaluating any control algorithm.

Driving behaviors: Another issue with existing research is that it lacks calibration for driving behaviors. This means that the algorithms may not be able to accurately predict how drivers will behave in different situations. For example, some drivers may be more aggressive than others, or they may be more likely to make certain types of turns.

Implementation of the baseline algorithm: Finally, many works have assumed the standard four-phase design for traffic lights and selected default parameter values. However, in the real world, traffic lights are often designed with different phases and parameters for different time slots of the day. This means that the results of existing research may not be applicable to real-world traffic lights.

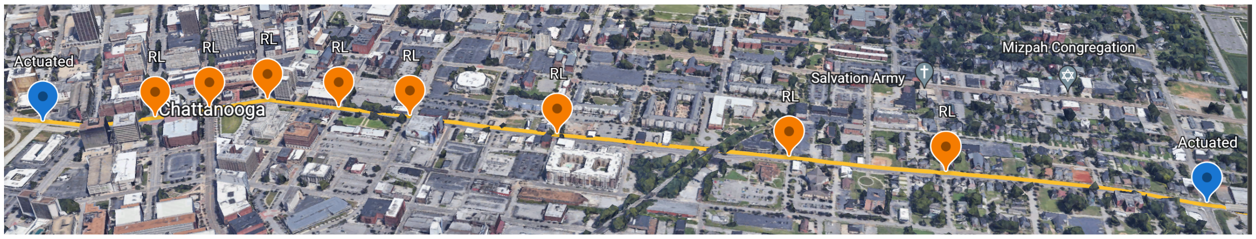


Figure 4.1 Eleven selected intersections along the testbed

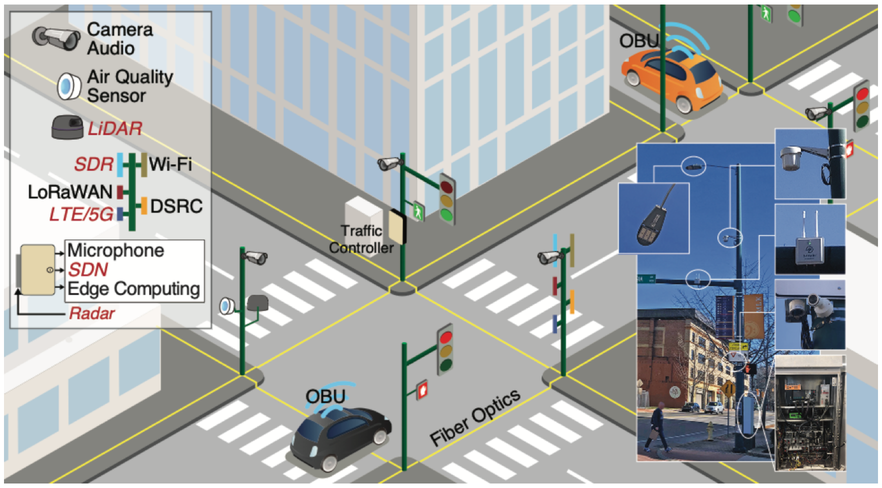


Figure 4.2 An example of one intersection on the MLK Smart Corridor. The bottom right image is the actual picture from one intersection on the corridor. The top left box lists the advanced sensor technologies at each intersection.

4.1 Proposed framework

This study proposes a framework [30] for optimizing traffic controllers in Chattanooga, Tennessee. The framework uses a smart corridor, illustrated in Figures 4.1, 4.2, which is equipped with sensors, wireless communications technologies, and edge computing to collect high-quality data. This data is used to develop a reliable digital twin that addresses the issues of workload volume and distribution by advanced tracking vehicle system, driving behavior by careful calibration, and baseline implementation by Signal Phasing and Timing (SPaT) data.

A digital twin model of a corridor is created using real-time data on traffic volume, turn movement counts, and SPaT messages. This model is used to develop an adaptive signal timing plan that is optimized for fuel consumption. The digital twin model provides data and performance measures that are not available from the field, such as approach delay, queue length per lane, and vehicle information such as vehicle type, trajectory, speed, and acceleration. These dynamic outputs from the digital twin inform the development of the signal timing optimization algorithm.

In simpler terms, a digital twin is a virtual replica of a physical system. In this case, the digital twin is a virtual model of a corridor. The digital twin is used to collect data on traffic conditions and to develop an adaptive signal timing plan that optimizes fuel consumption. The adaptive signal timing plan changes in real time to reflect changes in traffic conditions. This helps to reduce congestion and improve fuel efficiency.

To develop and test the optimization algorithm, we deploy it to a simulated environment called the “Field Testbed Digital Twin”. Figure 4.3 presents the Field Testbed implementation architecture of our framework. This simulated environment provides the current signal state, volume, and turn movement ratio information. The optimization algorithm uses this information to generate optimized signal timings, which are then implemented in the Field Testbed Digital Twin. The Field Testbed Digital Twin then simulates the optimized signal timing plan for the historic day’s corridor traffic conditions. This allows us to test different scenarios and develop the optimization algorithm by simulating trials.

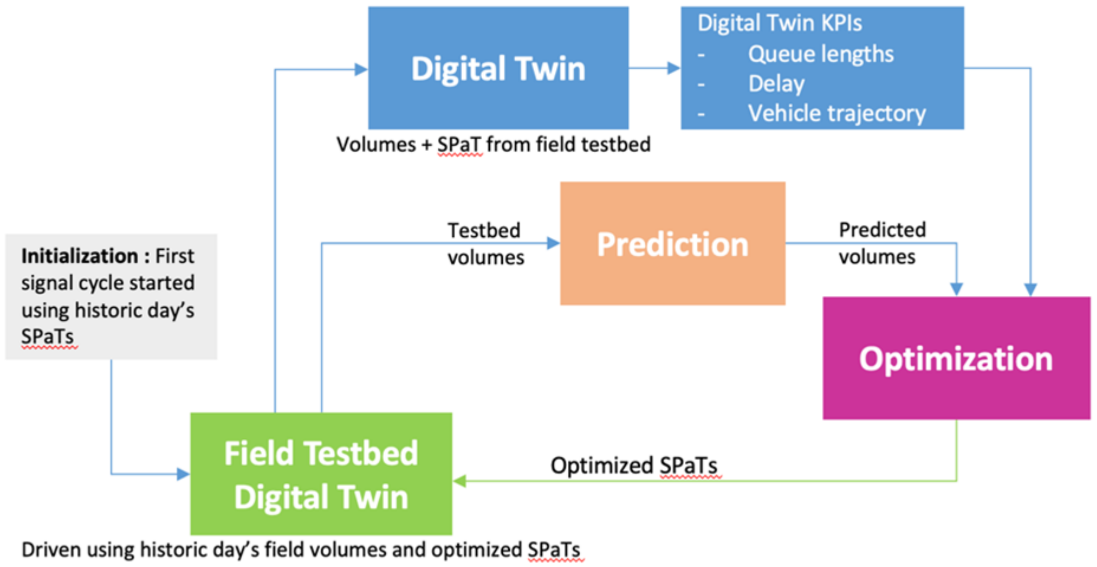
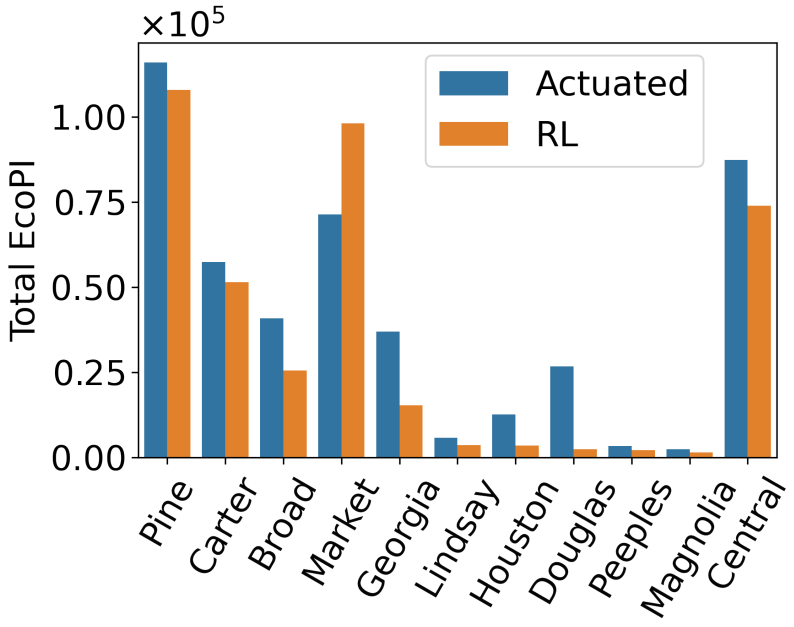

Digital Twin
Digital Twin KPIs
- Queue lengths
- Delay
- Vehicle trajectory
Volumes + SPaT from field testbed
Testbed volumes
Prediction
Predicted volumes
Initialization : First signal cycle started using historic day's SPaTs
Optimization
Field Testbed Digital Twin
Optimized SPaTs
Driven using historic day's field volumes and optimized SPaTs

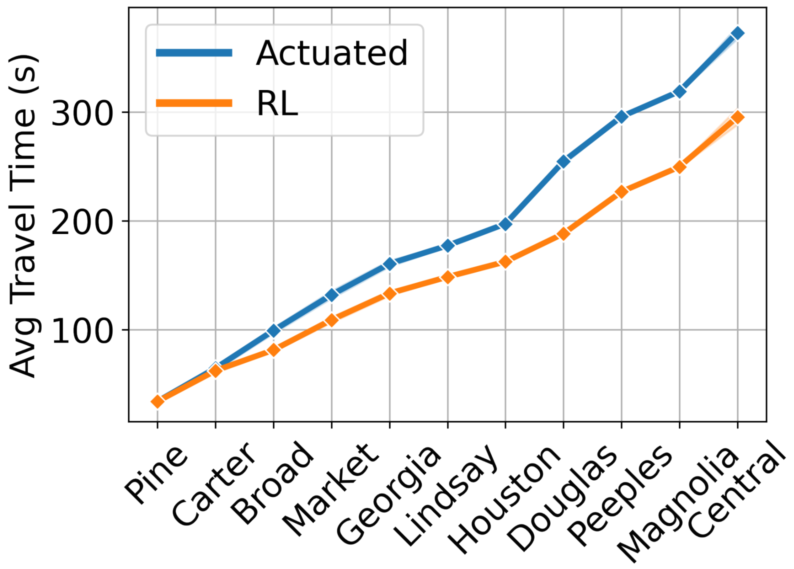
Figure 4.3 Architecture of the proposed framework

4.2 Demonstration

In a real-world testbed, reinforcement learning (RL) controllers were found to significantly improve fuel consumption and average travel time. The overall EcoPI [31] value of the entire corridor was improved by 15.78% (Figure 4.4), and average travel time was reduced by 20.81% (Figure 4.4). This was achieved by reducing the number of stops and stop delays at traffic lights. To visualize the results, vehicle trajectories were analyzed at an intersection called Houston and MLK. The trajectories (Figure 4.5) showed that RL-based TSCs (traffic signal controllers) resulted in smoother and more efficient traffic flow. This led to a reduction in fuel consumption and average travel time. These findings suggest that RL controllers can be used to improve the efficiency of traffic signals, which can lead to environmental and economic benefits.

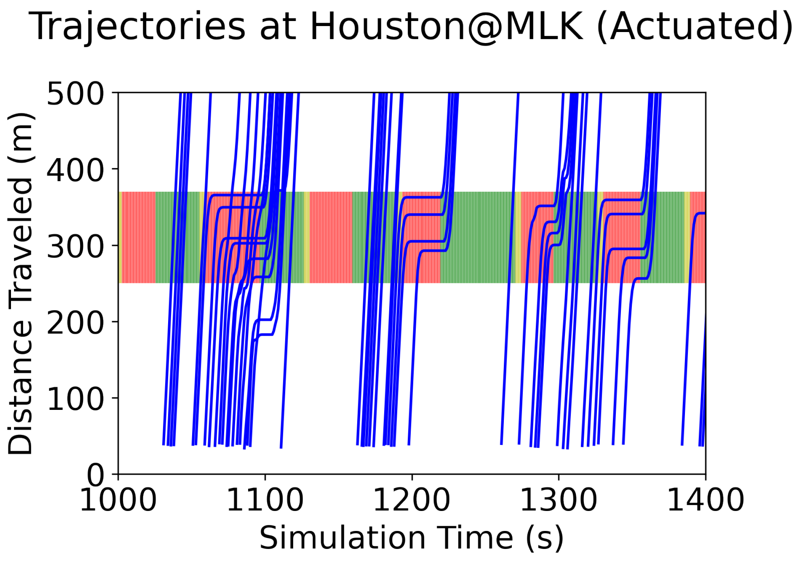


(a) Overall EcoPI of intersections



(b) Average travel time on the main street

Figure 4.4 Performance of Actuated and RL-based controls



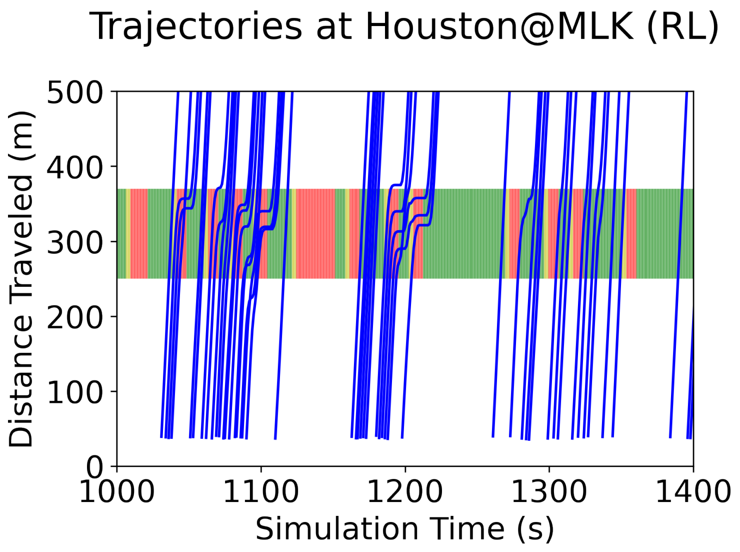


Figure 4.5 Trajectories of vehicles at Houston&MLK when using Actuated (left) and RL (right)

CHAPTER 5

Single Camera-enabled RL-based TSC System supporting Life-long Assessment

Chapter 4 has addressed various problems of reliable settings and baselines by using a digital twin. However, there is still a far path from digital twins to the field. It is because the simulation environment is different from the real world, which is a diverse, non-stationary and open-ended environment. This leads to issues related to unseen data, making it difficult to have a RL control that is robust to all scenarios. Additionally, traffic control is a problem that requires safety guarantees. Therefore, deploying RL-based TSCs necessitates a comprehensive life-long monitoring and evaluation.

However, existing reinforcement learning-based traffic signal control systems are evaluated on static benchmarks that have limitations [11, 21]. These limitations include: lack of abnormal scenarios, inability to capture long-term distribution shifts, and lack of real-time monitoring capability. Therefore, we propose a new RL-based TSC system that can be evaluated using an online dynamic benchmark [32]. Our system can constantly observe and assess performance in near real time, allowing transportation operations to respond rapidly to irregular situations. Our system's near real-time capabilities ensure that it can respond to unexpected changes in traffic patterns caused by special events or infrastructure changes. Overall, our proposed system provides a reliable and efficient solution in uncertain transportation environments.

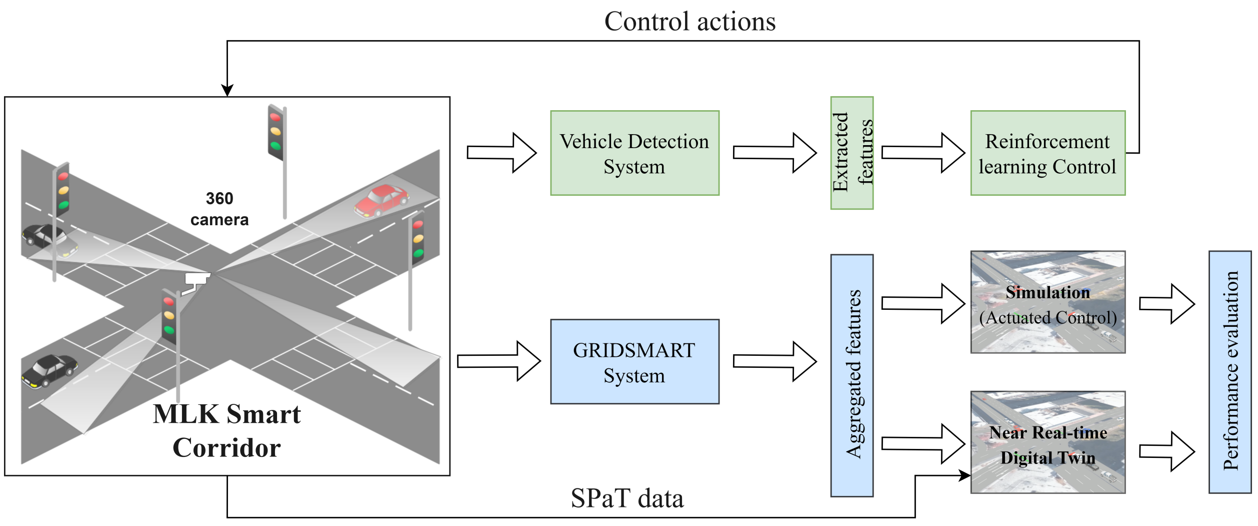


Figure 5.1 The architecture of the proposed system. The green components are conducted in real time while the blue ones come with one-minute delay.

5.1 Proposed framework

The proposed system is illustrated in Figure 5.1. It consists of five main components: Vehicle Detection System, Reinforcement Learning Control, GRIDSMART System, Simulation (Actuated Control), and Near Real-time Digital Twin. The Vehicle Detection System and Reinforcement Learning Control are conducted in real time. This means that they can process data and generate outputs immediately. The other three components, GRIDSMART System, Simulation (Actuated Control), and Near Real-time Digital Twin, have a one-minute delay. This means that they need to wait for one minute before they can process data and generate outputs.

• ***Vehicle Detection System***. Figure 5.2 presents the operation of our vehicle detection system. The vehicle detection system works by first capturing frames every second. These frames are then rotated and cropped to create four corresponding frames for each approach. Next, YOLOv8 [33] is used to detect vehicles in three classes: cars, trucks, and buses. Finally, the system determines the lane of each vehicle by finding the intersection of the vehicle's bounding box and the lane zones.

• ***Reinforcement Learning Control***. Reinforcement learning is a type of machine learning that allows an agent to learn how to behave in an environment by trial and

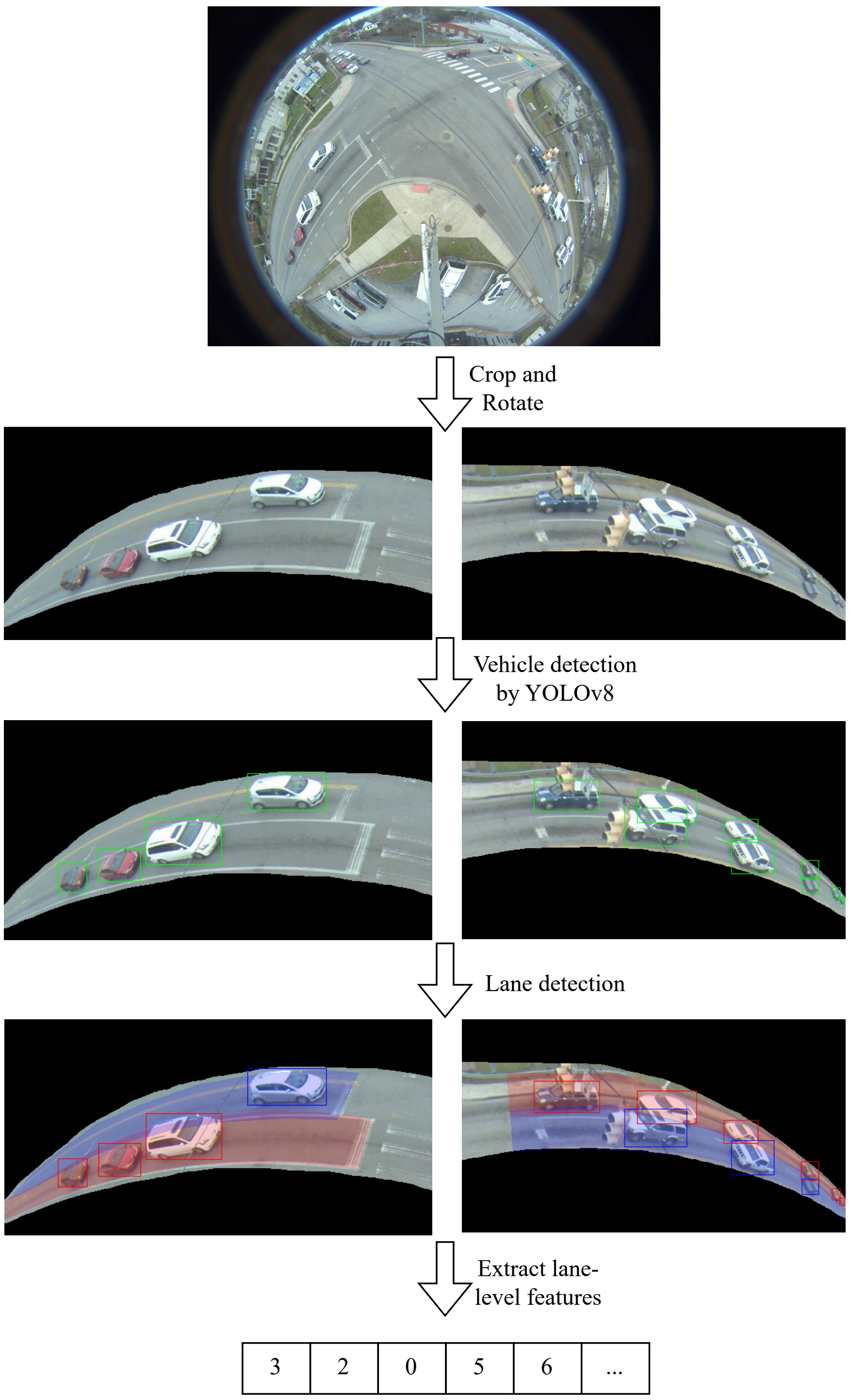


Figure 5.2 Illustration of the operation of our vehicle detection system which processes 360-camera frames to lane-level features

error. RL agents learn by receiving rewards for taking actions that lead to desired outcomes. In our system, the RL agent is responsible for controlling traffic lights at an intersection. The agent's state is defined by the number of vehicles in each 30-meter segment of each lane, same to MPLight [7]. The agent's action is to either keep the current traffic light phase or switch to the next phase. The agent's reward is calculated based on the traffic pressure [13], which is the difference between the number of vehicles on outgoing lanes and the number of approaching vehicles.

- ***GridSmart System***. GridSmart is a computer vision system that monitors traffic flows at intersections. It collects data on traffic volume, speed, and direction, and aggregates this information into features at zone, lane, and movement levels. However, this aggregation process introduces a one-minute delay, which can be problematic for real-time RL-based TSCs. Therefore, RL-based TSCs need to be able to adapt to traffic flows in real time, and the one-minute delay from GridSmart can make this difficult.

- ***Simulation and Near Real-time Digital Twin*** . Simulation and Near Real-time Digital Twin (NRDT) are using two traffic signal control strategies. NRDT receives SPaT data from the field, allowing it to synchronize its traffic lights with those in the field. Actuated Control does not receive SPaT data, but it is a simple logic method that has been extensively studied and shown reliable results in previous simulation research. Therefore, even if the field does not implement Actuated Control, we are still able to measure its performance through advanced simulation. To evaluate the performance of the two algorithms, we calculate the outputs of Simulation and NRDT. Actuated Control is used as the baseline because it is widely implemented in real-world transportation systems.

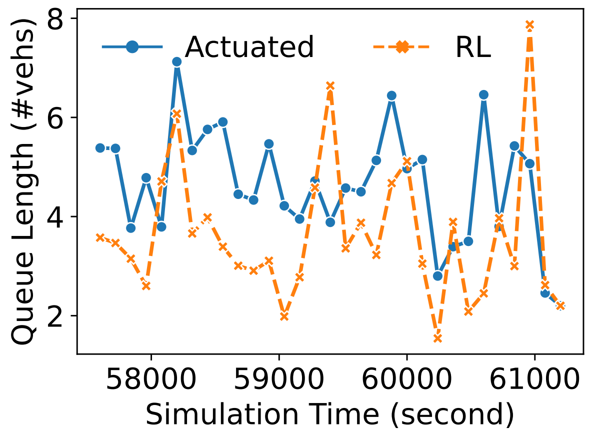
5.2 Demonstration

5.2.1 Vehicle Detection System

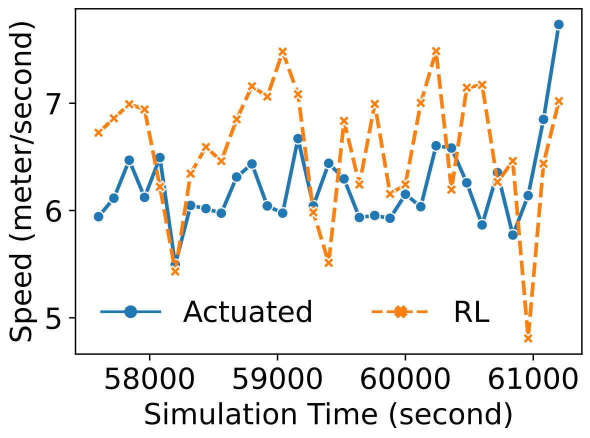
The vehicle detection system is based on YOLOv8. We collected and annotated 500 frames of video footage, which included 3 vehicle classes: cars, trucks, and buses. Cars made up the majority of the vehicles, accounting for 96.91% of the total. We fine-tuned YOLOv8 on our dataset for 100 epochs and achieved a mean average precision of 0.964, 0.724, and 0.655 for cars, trucks, and buses, respectively.

5.2.2 RL-based Traffic Signal Control

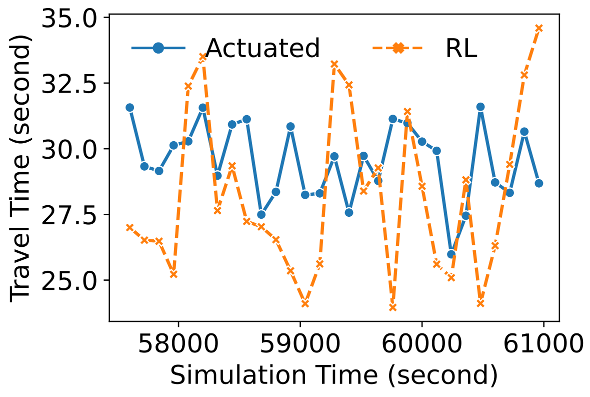
we develop a digital twin of the MLK Smart Corridor, which is a 11-intersection stretch of road in downtown Chattanooga, TN for a day of May 15th 2021 [30]. In this paper, we compare the performance of two traffic signal control (TSC) algorithms: RL and actuated control. We simulate the intersection of MLK and Market Street during the one-hour peak period from 4pm to 5pm. Our RL-based TSC algorithm demonstrates significant improvements in average queue length, speed, travel time, and total fuel consumption, with improvements of 22.23%, 6.34%, 4.29%, and 3.69%, respectively. The details of improvements are shown in Figure 5.3.



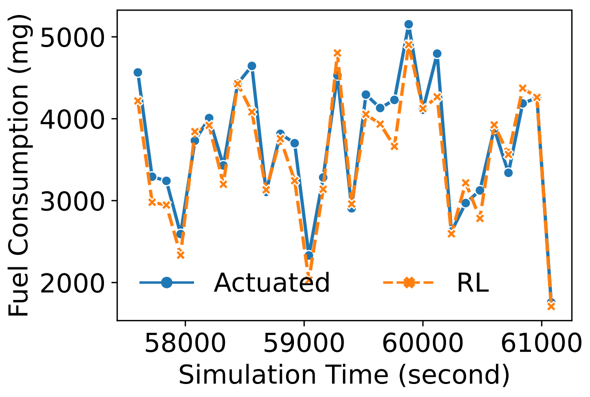
(a) Queue length



(b) Average speed



(c) Average travel time



(d) Total fuel consumption

Figure 5.3 Comparison of our RL-based TSC and the actuated control (which is currently running in the real-world testbed) at the intersection of Market & MLK from 4pm to 5pm

Part II

Citywide Public Transit Simulation

CHAPTER 6
SIMCal: A High-Performance Toolkit For Calibrating Traffic Simulation

As already stated in Chapter 1, public transit simulation is a citywide problem. Such large-scale traffic simulations require careful calibration due to the complex nature of traffic patterns and the multitude of variables involved. Calibrating these simulations involves accurately representing real-world conditions, such as road layouts, traffic signals, vehicle behavior, and driver characteristics. Ensuring the simulation matches actual traffic flow is crucial for improving the simulation accuracy and obtaining realistic results.

In this work, we introduce SIMCal [34] – a novel toolkit for calibrating traffic simulations. It supports multiple state-of-the-art algorithms, parallelization capabilities, flexibility, and open source nature. These features make SIMCal a powerful and versatile tool that can be used to improve the accuracy of traffic simulations in a variety of settings.

6.1 Brief Description of SIMCal

6.1.1 SIMCal's framework

SIMCal has a three-part framework: Simulation Processing, Optimization, and Output&Visualization, illustrated in Figure 6.1. Simulation Processing executes simulations and collects output data via APIs. Optimization finds a parameter set that can accurately represent the real traffic by performing a trial-and-error process, i.e., an iteration of a loop adjusting parameter sets and evaluating these sets via Simulation Processing. Output&Visualization represents the optimized parameter.

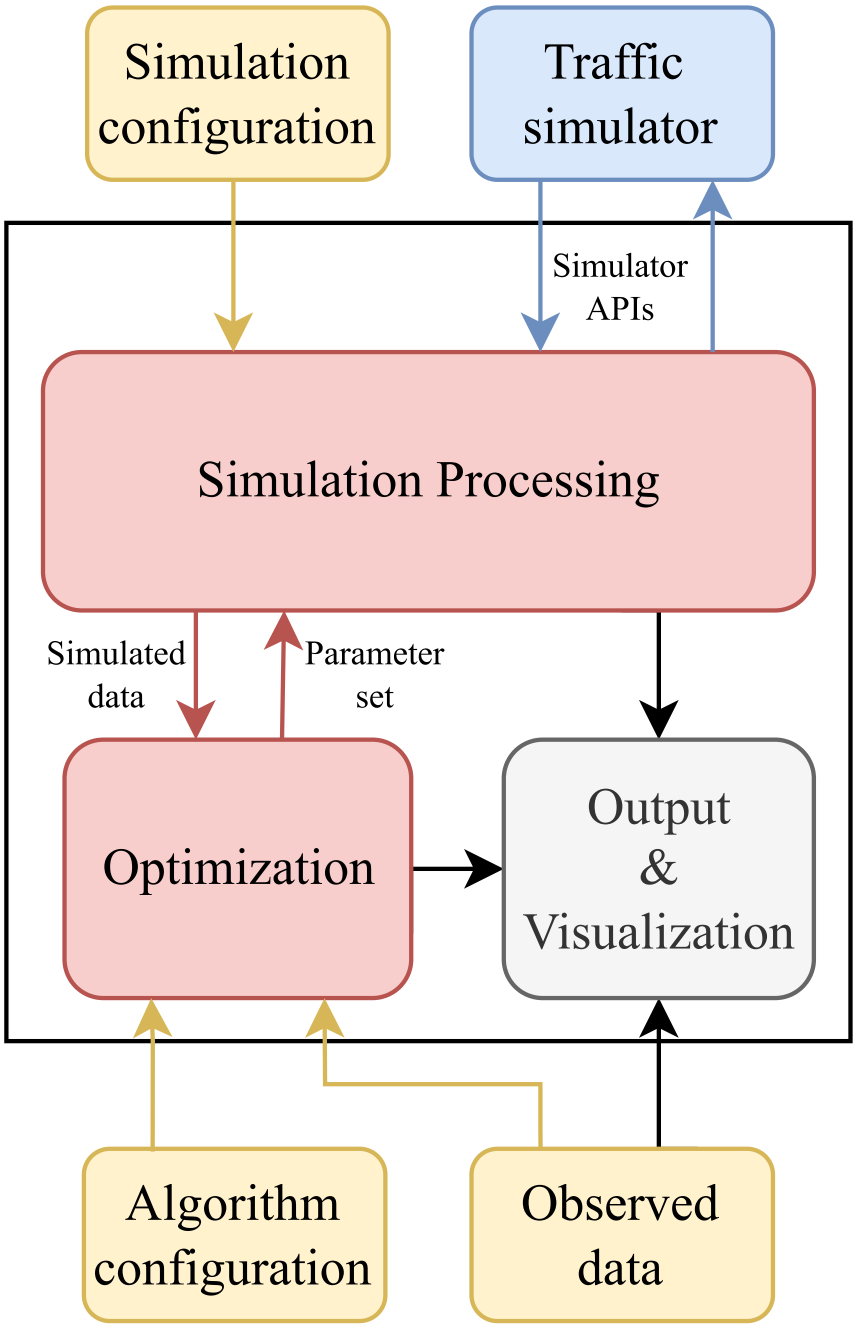


Figure 6.1 SIMCal's framework

6.1.2 Inputs

Simulation configuration. Our toolkit can be used to calibration traffic simulation using SUMO, a popular traffic simulator. The simulation configuration is made up of files that describe the road network, routes, and detectors.

Observed speed data. This csv file contains the average real-world observed speed of traffic during intervals. Our toolkit allows users to choose the interval length. Table 6.1 shows an example of 5-minute observed speed data.

Table 6.1 Five-minute observed speed data format

	DetectorID #1	**DetectorID #2**	**...**
2021-05-11 00:00:00	10.2	15.6	..
2021-05-11 00:05:00	12.5	16.4	..
...	...	...	...

Configuration. The YAML file below shows an example of how to set up SIMCal. SIMCal allows users to choose how to allocate computing resources and to select parameters of the simulation that need to be tuned, such as the range.

```
algorithm:
    name: "PSO"
    hyperparameter: {
        "w": 0.5,
        "number_particles": 15
    }
    objective_function: "MAPE"
    number_iterations: 50
    number_processors: 15

simulation:
    sumo_cfg: corridor.sumocfg
    obs_data: observed_speed_data.csv


passenger:
    MinGap: [1, 3]
    Tau: [0, 2]
    Accel: [2, 4]
    ...

initial: "default"
```

6.1.3 Optimization

Our toolkit uses evolutionary algorithms to optimize traffic simulation parameters. Figure 6.2 presents the flowchart of the optimization process. The optimization process starts by initializing a population of parameter sets. Then, there is a loop of evaluation and evolution. The evaluation step measures the differences between speed data from detectors in the real world and in the simulation. SIMCal offers some popular objective functions for evaluation, including MSE, MAE, and MAPE. The evolution step is about how the population evolves, which is done using evolutionary algorithms. There are four available evolutionary algorithms implemented in SIMCal:

1. Genetic Algorithm (GA), which is inspired by the natural evolution, implements biological operators such as selection, mutation, and crossover [35].
2. Particle Swarm Optimization (PSO) simulates movements and intelligence of swarms [36].
3. Firefly Algorithm (FA) follows the flashing behavior of fireflies [37].
4. Simulated Annealing (SA) is inspired by the annealing technique in metallurgy [38].

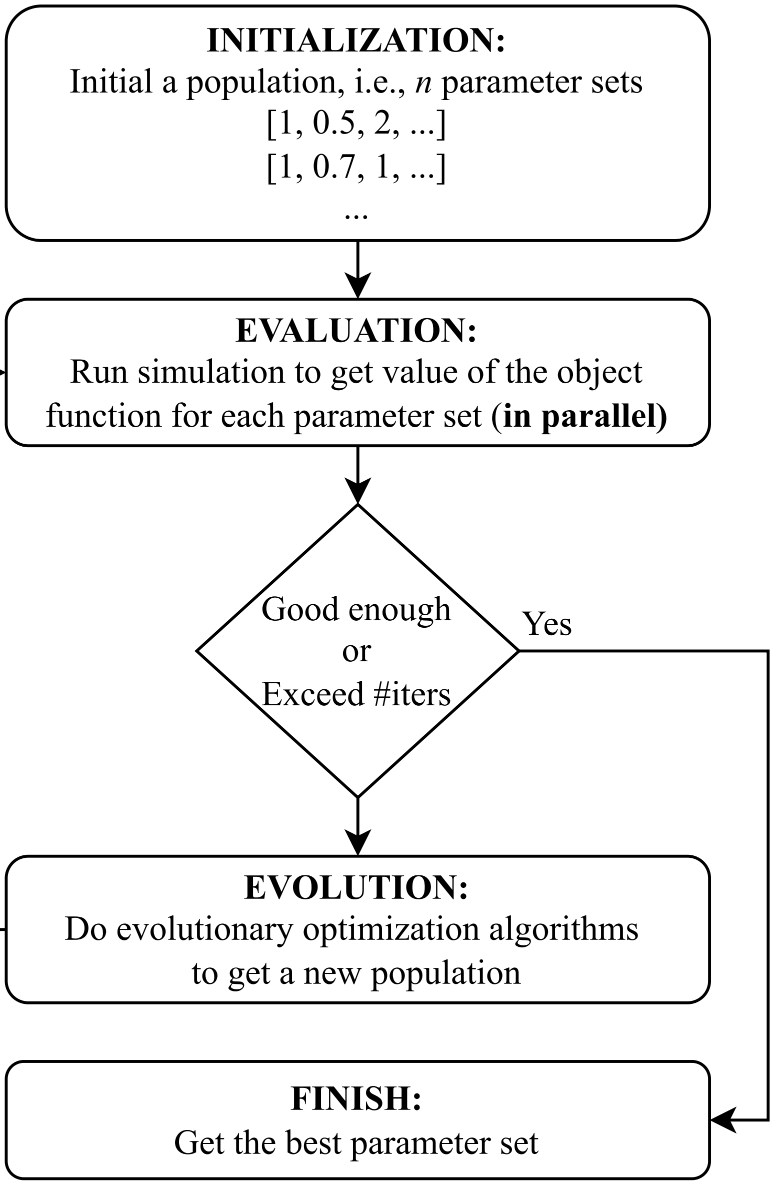

INITIALIZATION:
Initial a population, i.e., n parameter sets
[1, 0.5, 2, ...]
[1, 0.7, 1, ...]
...
EVALUATION:
Run simulation to get value of the object function for each parameter set (in parallel)
Good enough or Exceed #iters
Yes
EVOLUTION:
Do evolutionary optimization algorithms to get a new population
FINISH:
Get the best parameter set

Figure 6.2 Flowchart of the optimization process

6.1.4 Output and Visualization

The optimization process finds a parameter set that minimizes the objective function. The parameter set can include parameters for car-following models, such as MinGap and Tau, and for vehicle characteristics, such as speed deviation, acceleration, and deceleration. The number of parameters is usually large, so it is difficult to visualize the parameter sets found by the optimization process using coordinate systems. SIMCal offers using PCA to reduce the dimensionality of the parameter space so that we can visualize the solutions.

6.2 Experiments & Results

6.2.1 Experiment setting

All our experiments are conducted using SUMO on a computer with 80 processors and 188 GB of memory. In these experiments, we calibrate for six parameters:

- MinGap: the minimum distance between a vehicle and its leader in meter.
- Tau: the minimum time headway in seconds.
- Sigma: the driver imperfection – a value between between 0 and 1, which represent the perfect driving and the maximum driving imperfection.
- Speed deviation
- Acceleration
- Deceleration

6.2.2 Datasets

• ***MLK Corridor***. Traffic data was collected from the MLK Smart Corridor including 10 signalized intersections in downtown Chattanooga, TN, USA on May 11, 2021. The data includes information on vehicle routes, speeds, and traffic signal timing. This data was collected using a variety of methods, including detectors, cameras, and LiDAR devices. The data is accurate and represents real-world traffic conditions.

• ***City of Chattanooga***. We get traffic data from the whole city from INRIX – a comprehensive database of historical traffic time and speed. INRIX is a company that collects traffic data from a variety of sources, including sensors, fleet vehicles, taxis, users of the INRIX Traffic App, and local transportation authorities. The Chattanooga simulation, developed in [8], uses this data to create a realistic model of traffic flow in the city. The simulation includes a road network from Open Street Map and traffic demands based on an Origin-Destination matrix.

6.2.3 Algorithm effects

• ***Setting***. We simulate the corridor from 5 a.m. to 10 a.m. and conduct calibration using four algorithms: simulated annealing (SA), tabu search (FA), genetic algorithm (GA), and particle swarm optimization (PSO).

• ***Result***. Figure 6.3 presents performance of the algorithms. All four algorithms significantly improve the mean absolute percentage error (MAPE), detailed in Table 6.2. PSO and GA show the best performance, with MAPE values of 0.0838 and 0.0848, respectively. Both algorithms improve MAPE by almost 50% compared to the simulation using the default parameter set. The traffic calibration problem may have many local minima, which can cause SA to become stuck. PSO and GA have more versatile update strategies than SA, which allows them to escape local minima more easily. Therefore, SA is not as effective as PSO and GA in this case.

6.2.4 Population size effects

• ***Setting***. We simulate traffic in a corridor from 5 a.m. to 10 a.m. Due to the high computational cost, we only analyzed PSO. We conduct experiments with different

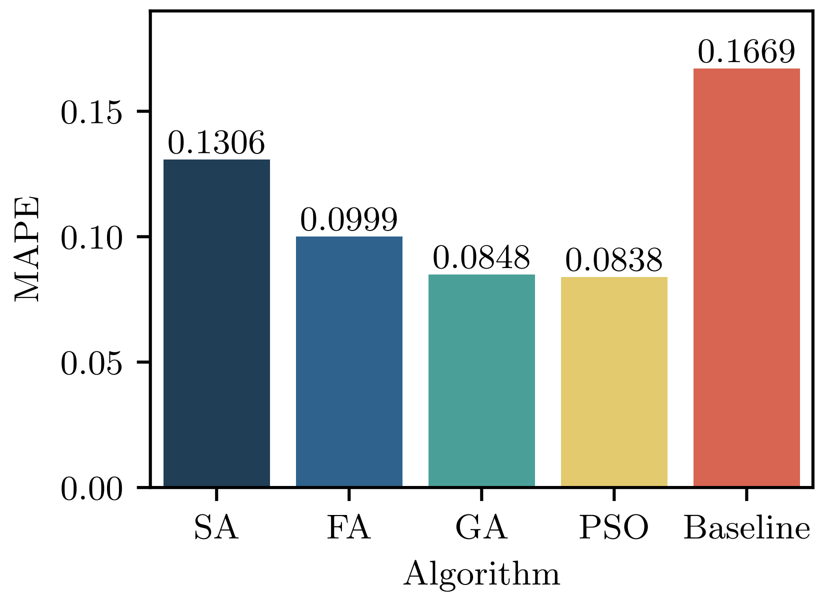


Figure 6.3 SIMCal's performance for calibrating the MLK corridor with different algorithms. The baseline is the simulation using default parameters

population sizes of 5, 15, 30, and 60 particles to measure the effects of population size.

• ***Result***. All population sizes significantly improved the mean absolute percentage error (MAPE) by at least 44.63%. The larger the population size, the better the parameter set found. The smallest population size of 5 particles achieved a MAPE of 9.24%, while the largest population size of 60 particles achieved a MAPE of 8.05%, which is the best performance. Figure 6.4 shows the MAPE over iterations of the population sizes. To visualize the results, we applied principal component analysis (PCA) to reduce the dimensions of the parameter sets. Figure 6.5 presents a visualization of the parameter sets found by PSO. The population sizes of 5 and 15 particles achieved equivalent results of 9.24% and 9.20% MAPE, respectively. However, their solutions are not close in the PCA space. This means that it is possible to have multiple solutions that provide equivalent performance for traffic calibration.

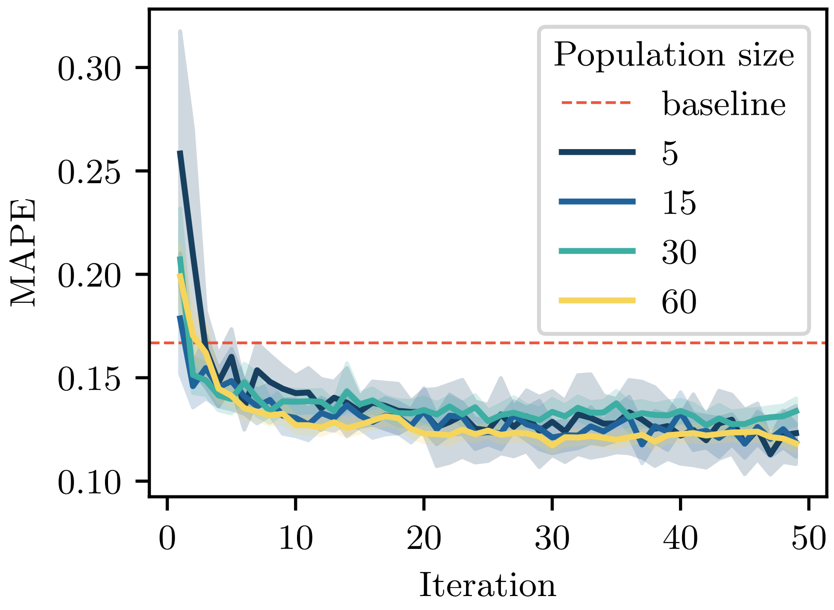


Figure 6.4 MAPE over iterations in terms of different population sizes

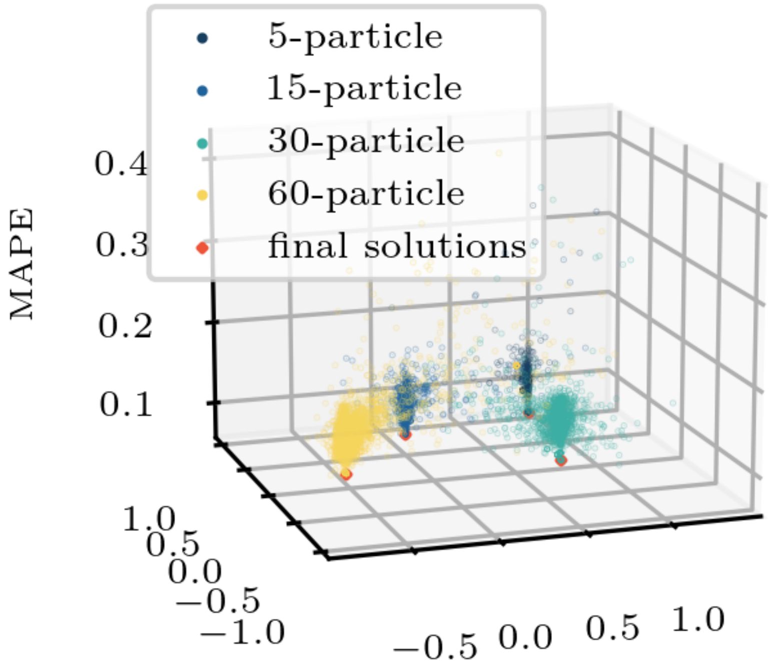


Figure 6.5 Visualization of parameter sets during 50 iterations by using PCA

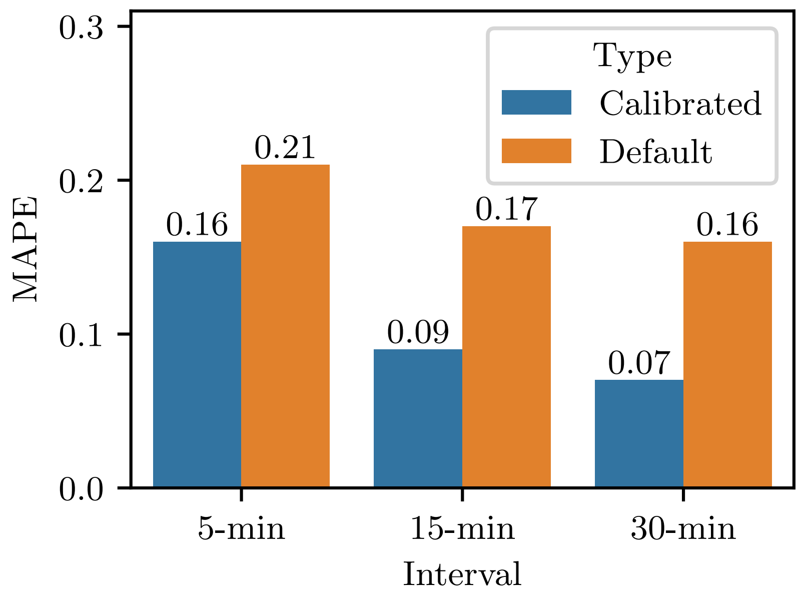


Figure 6.6 MAPE in terms of different intervals

6.2.5 Interval effects

• ***Setting***. We calibrate the traffic in a corridor from 5 a.m. to 10 a.m. To measure the effects of the interval size, we calculate the objective function with different intervals of 5, 15, and 30 minutes.

• ***Result***. The calibration process improves MAPE in all cases. The improvement was most significant for the 30-minute interval (57.70%), illustrated in Figure 6.6. The impact of vehicle individuality on MAPE is higher for shorter intervals, leading to a negative correlation between MAPE and interval size. For example, the average error of average speeds for each 30-minute interval is only 6.63%, while the error for each 5-minute interval is 15.96%. Figure 6.7 shows a visualization of the parameter sets found in the PCA space. This space suggests that the final solutions will be different for different interval sizes.

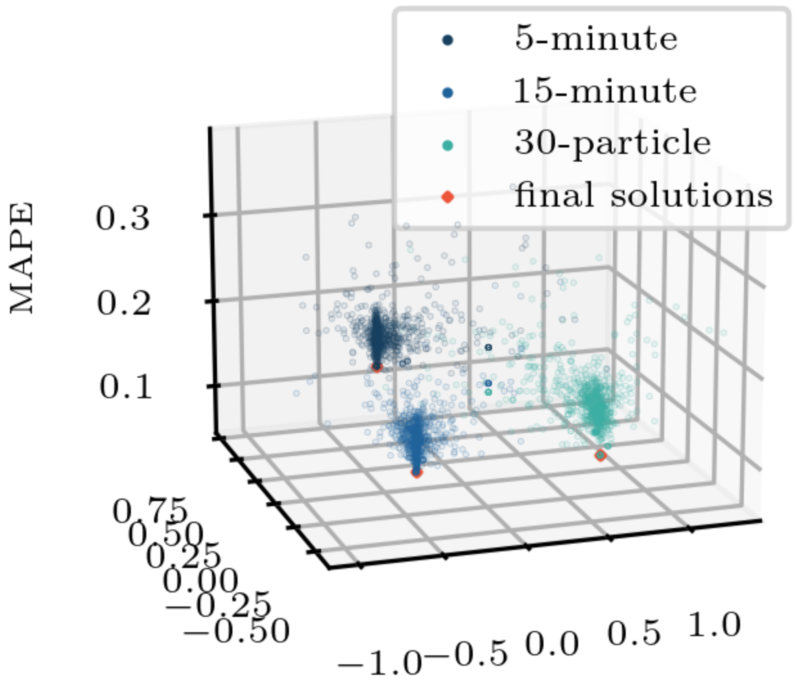


Figure 6.7 PCA-space visualization of parameter sets for different intervals

6.2.6 Traffic demand effects

• ***Setting***. In this experiment, we calibrate the traffic model for two different traffic scenarios: low-demand and high-demand. The low-demand scenario simulates traffic at night, while the high-demand scenario simulates traffic during peak hours, illustrated in Figure 6.8. Figure 1 shows the number of vehicles per hour on May 11, 2021 at the MLK corridor. The low-demand setting has an average of 306 vehicles per hour, while the high-demand setting has an average of 4146 vehicles per hour, which is about 13.5 times higher.

• ***Result***. Figure 1 shows the MAPE results before and after calibration for low- and high-demand traffic scenarios. SIMCal reduced the MAPE by 43.90% and 62.70%, respectively. Table 6.2 shows that the MinGap values are not much different. However, the found parameter sets show that the minimum headway time tends to decrease when the traffic is crowded. This trend has also been observed in other studies [39]. Sigma represents the slow-to-start behavior. The results show that there are more slow-to-start behaviors in low-demand traffic than in high-demand traffic. Drivers also tend to accelerate faster in crowded traffic than in sparse traffic.

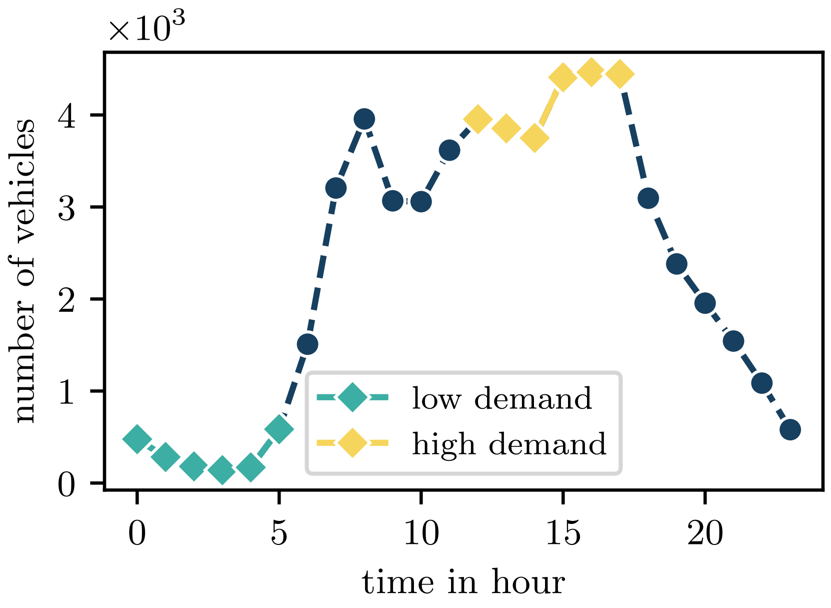


Figure 6.8 Traffic demands per hour on May 11, 2021 at the MLK corridor

6.2.7 Network size effects

• ***Setting***. In this experiment, we calibrate a traffic model for two different test sites. The first test site is a corridor that includes 4.34 km of roads and 8,251 vehicles. The second test site is the entire Chattanooga transportation network, which consists of 6176.69 km of roads and around 250,000 vehicles.

• ***Result***. SIMCal reduced the MAPE by 22.94% and 44.42% for the corridor and city simulations, respectively. However, the city-level simulation still has a significant error of 36.21%. This may be because using the same parameters for the whole city is not efficient, as different regions have different driving behaviors. For example, downtown and suburban areas have different traffic patterns. Figure 1 shows the MAPE during the calibration process for the simulation of Chattanooga. The algorithm took around 2.5 days to converge (Figure 6.9), due to the heavy demand on simulation computing. Calibrating the corridor only took several hours.

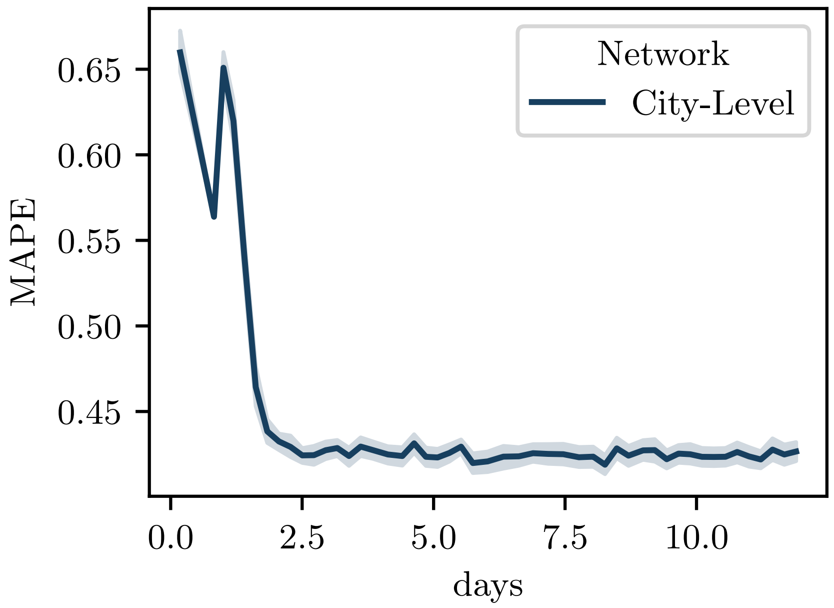


Figure 6.9 Running time to calibrate the Chattanooga simulation

Table 6.2 Detailed results of all experiments

Experiments		**Parameters**						**Results**	
Factor	**Type**	**MinGap**	**Tau**	**Sigma**	**Speed -deviation**	**Accel**	**Decel**	**MAPE (%)**	**Improvement (%)**
Algorithm	SA	4.000	0.919	0.513	0.200	0.600	4.018	13.06	21.75
	FA	1.000	1.473	0.399	0.010	1.293	5.682	9.90	40.65
	GA	1.562	2.000	0.000	0.188	2.000	6.000	8.48	49.20
	PSO	2.090	1.345	0.383	0.200	2.426	4.468	**8.38**	**49.75**
Population size	5	2.484	0.990	0.842	0.100	2.796	4.172	9.24	44.63
	15	2.940	1.392	0.701	0.100	2.407	4.726	9.20	44.85
	30	2.090	1.345	0.383	0.200	2.426	4.468	8.38	49.75
	60	3.000	2.000	0.551	0.100	2.219	4.915	**8.05**	**51.78**
Interval	5-min	2.484	1.671	0.692	0.100	2.438	4.641	15.96	22.94
	15-min	2.940	1.392	0.701	0.100	2.407	4.726	9.20	44.86
	30-min	2.219	0.744	0.956	0.200	3.239	4.671	6.63	57.72
Demand	Low	2.525	1.504	1.000	0.100	2.941	4.899	17.44	43.90
	High	2.585	1.050	0.683	0.100	2.662	3.854	6.40	62.70
Network size	Corridor	2.484	1.671	0.692	0.100	2.438	4.641	15.96	22.94
	City Level	1.734	2.000	1.000	0.200	4.000	3.000	36.21	44.42
Default of SUMO		2.5	1	0.5	0.1	2.6	4.5		
Range		[1 - 3]	[0 - 2]	[0 - 1]	[0.1 - 0.2]	[2 - 4]	[3 - 6]		

CHAPTER 7
BTE-Sim: Fast Simulation Environment For Public Transportation

Clearly, transit simulation is a powerful tool that can be used to improve transportation systems and make cities more livable. It can help transportation planners and engineers to understand how people move around a city, test new transportation ideas, and plan for the future. In addition, transit simulation can be used to educate the public about transportation issues and promote innovation in transportation. Transit simulation works by creating a computer model of a transportation system. This model can be used to simulate the movement of people and vehicles, as well as the impact of different transportation policies and projects. For example, a transit simulation can be used to see how a new bus route or train line would impact traffic patterns, or how a change in parking rates would affect the number of people who drive to work. Therefore, transit simulation is a valuable tool for transportation planners and engineers.

Transit simulation is a citywide problem. For example, Chattanooga's transportation system has the total road length of 13455 km, 28311 junctions, and around 250,000 vehicles per day. Simulating simulation such a huge system can take a round 8 hours for one-day traffic [8]. A transit simulation is a computer model that can be used to study how different factors affect the performance of a transportation system. Figure 7.1 illustrates three main components of a transit simulation: the transit system, background traffic, and transportation infrastructure. The transit system is made up of buses, bus stops, and commuters. Background traffic includes other modes of transportation, such as private vehicles, taxis, freight vehicles, and pedestrians. The transit system and background traffic share the use of transportation infrastructure, such as roads, bridges, and tunnels. This means that they can

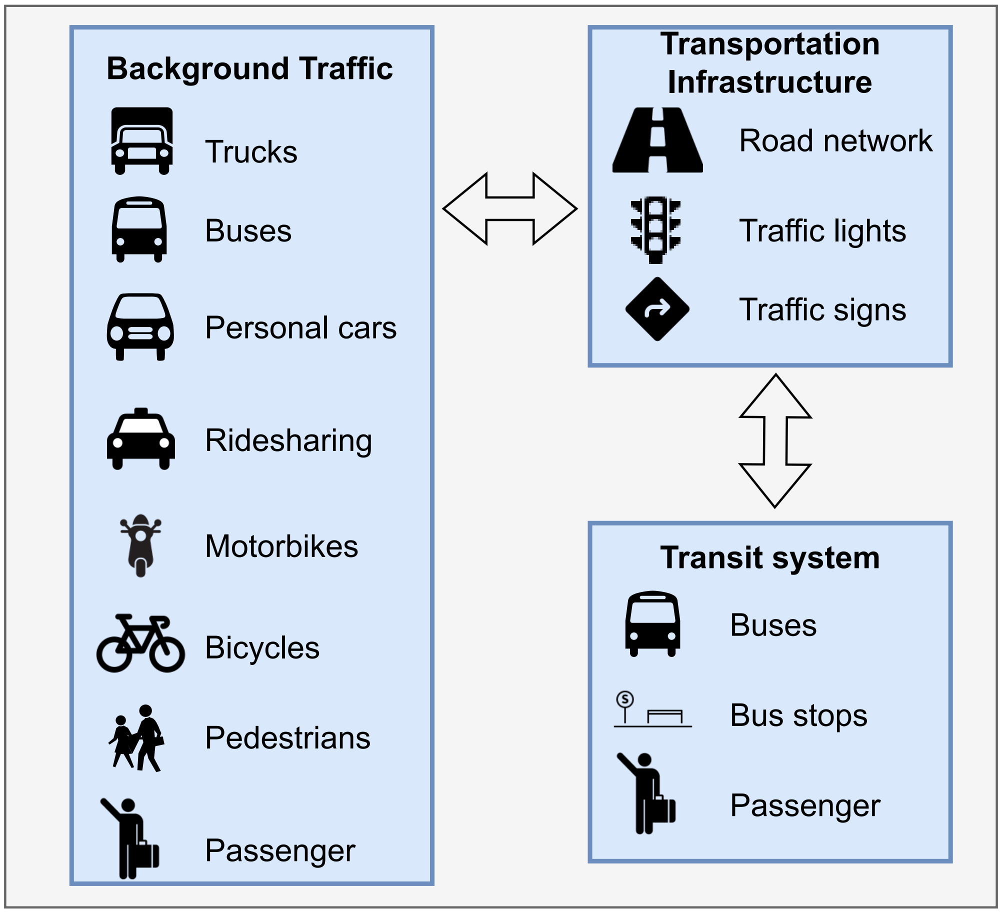


Figure 7.1 Components of a transit simulation

affect each other. For example, if there is a lot of background traffic, it can make it more difficult for buses to get around. This can lead to longer travel times for commuters.

In this work, we address a novel problem – how to conduct a new simulation where only the transit system's setting (e.g., number of buses) changes while the rest remains stable. This scenario happens in many transit route optimization and planning tasks. For example, greedy or evolutionary approaches are usually used to solve these tasks. This involves repeatedly simulating the process while adjusting only the bus routes.

One naive way is to update a transit system's setting and start from scratch each time. However, this can be time-consuming and inefficient because of the expensive computing cost of citywide simulations. To improve the computing-time efficiency, we propose a module called Background Traffic Elimination (BTE) [40]. BTE simulates the effects of background traffic on transit systems. We call this method of transit simulation BTE-Sim. The new module can simulate transit systems 13 times faster than previous methods, while still producing similar results in terms of trip duration, bus delay, bus speeds, total distance traveled, and virtual passenger alightings.

7.1 Brief Description of BTE-Sim

Figure 7.2 illustrates the key idea of BTE-Sim which replaces the background traffic by a historical edge-speed database. The database can be real-world observed as well as achieved from full simulations. When removing the background traffic, the movement behavior of buses will be changed. More specifically, the buses are implemented a car-following model that accounts the leader's speed and acceleration, shown in Figure 7.3. When the background traffic are eliminated, the buses will be free moved and easy to reach the speed limit of the edge. Therefore, our BTE-Sim dynamically adjusts the speed limit of all edges based on the historical database to force the buses to move as in a full simulation. The operation of BTE-Sim is illustrated in Figure 7.5.

7.2 Experiments and Results

7.2.1 Exp 1: BTE-Sim using different sources for the historical edge-speed database

• ***Setting***. To simulate the transit system on January 11, 2022, we use BTE-Sim from three sources: a previous run of Transit-Gym, INRIX, and Automated Passenger Count (APC).

• ***Result***. In Figure 7.6, the error in the time of arrival (ToA) is larger for data from APC (Automatic Passenger Counter) than for data from INRIX (a traffic data company). We could have used other data sources, but BTE-Sim (a bus travel time estimation model) performed best when using background traffic times generated using Transit-Gym (a transit

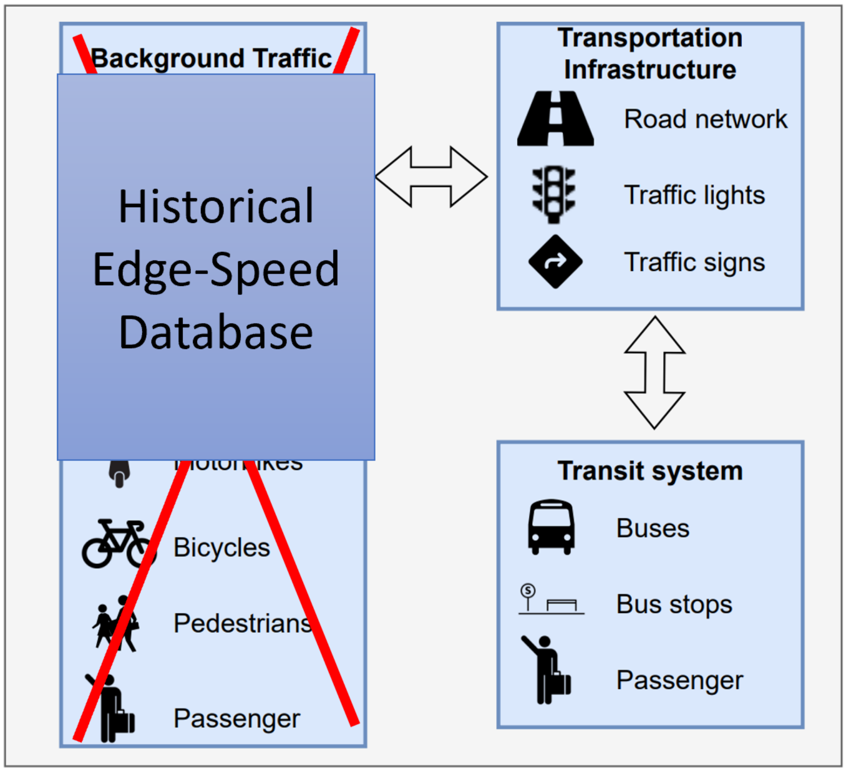


Figure 7.2 BTE replacing the background traffic by a historical edge-speed database

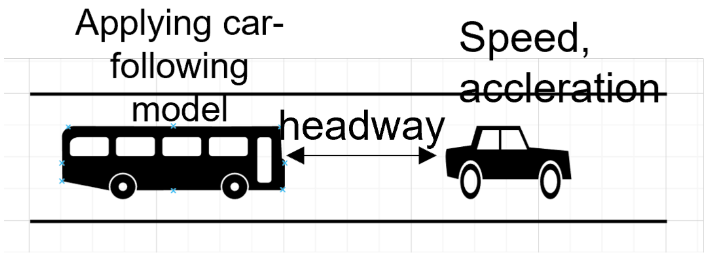


Figure 7.3 Bus movement with background traffic

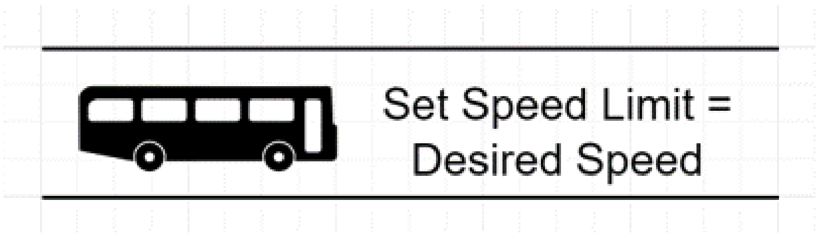


Figure 7.4 Bus movement without background traffic

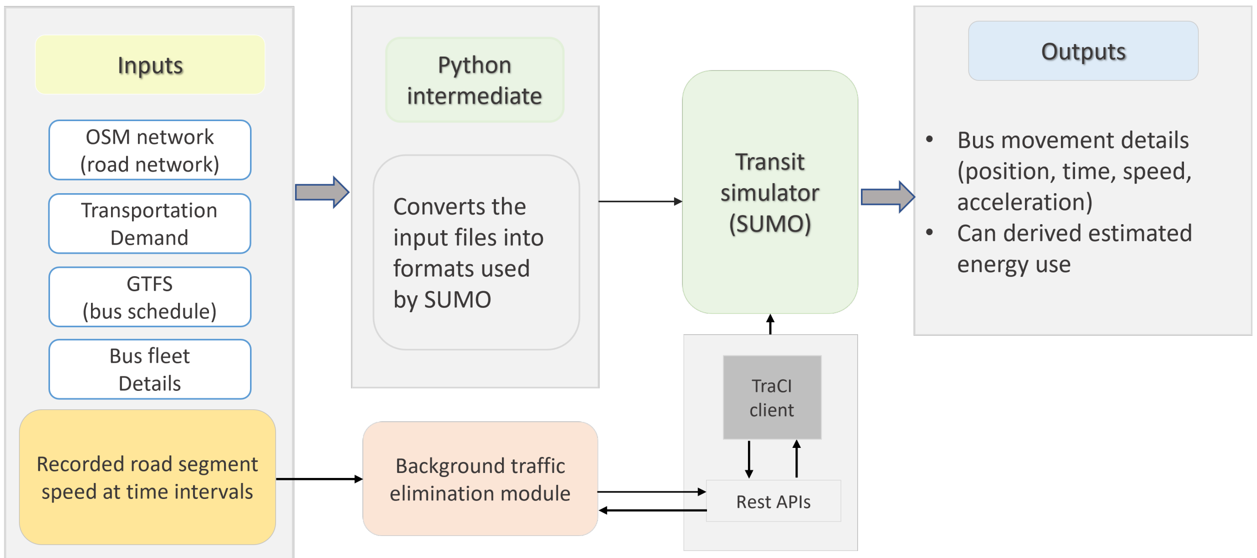

Inputs
OSM network (road network)
Transportation Demand
GTFS (bus schedule)
Bus fleet Details
Recorded road segment speed at time intervals
Python intermediate
Converts the input files into formats used by SUMO
Background traffic elimination module
Transit simulator (SUMO)
TraCI client
Rest APIs
Outputs
Bus movement details (position, time, speed, acceleration)
Can derived estimated energy use

Figure 7.5 BTE-Sim: Transit simulation without background traffic

simulation platform). Therefore, we used the background traffic speeds from Transit-Gym for the rest of the experiments. Moreover, it is possible to use any other data sources, such as google map, that include edge speeds.

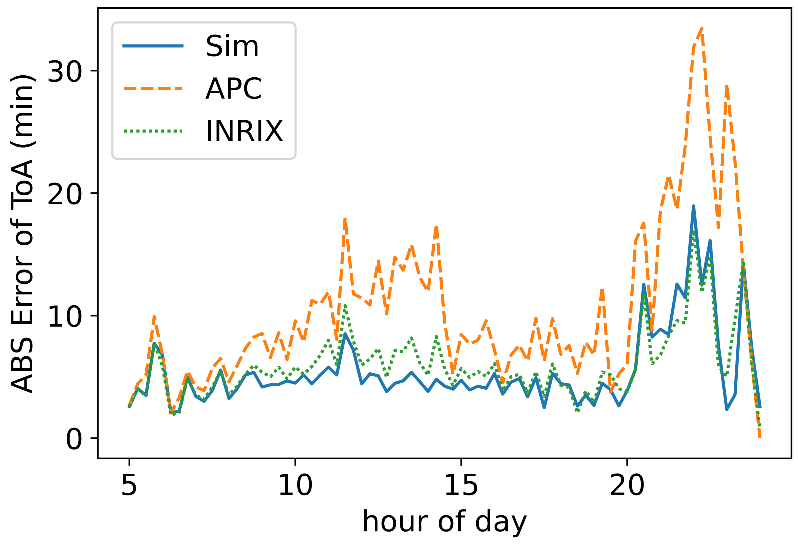


Figure 7.6 Comparing background traffic sources for BTE-Sim

7.2.2 Exp 2: BTE-Sim achieves competitive results compared to Transit-Gym

• ***Setting***. We conduct simulations of public transit in Chattanooga on January 11, 2022. We use real data from that day's transit operations. The city's planning agency provided us with an origin-destination (OD) travel matrix which are used to generate background traffic for Transit-Gym.

• ***Result***. To evaluate the simulation, we compared the Time of Arrival (ToA) at bus stops between the simulation and real world times. We measured the difference in minutes. Figure 7.7 shows the absolute error of ToA for Transit-gym and BTE-Sim. BTE-Sim uses historical edge speeds and is able to mimic the effects of background traffic. BTE-Sim has a consistently lower ToA than Transit-gym for an entire day's operation. Moreover, BTE-Sim can provide an extensive analysis of transit performance as Transit-gym does. Figure 7.8 depicts the performance of the Chattanooga's transit system on January 11, 2022 by BTE-Sim.

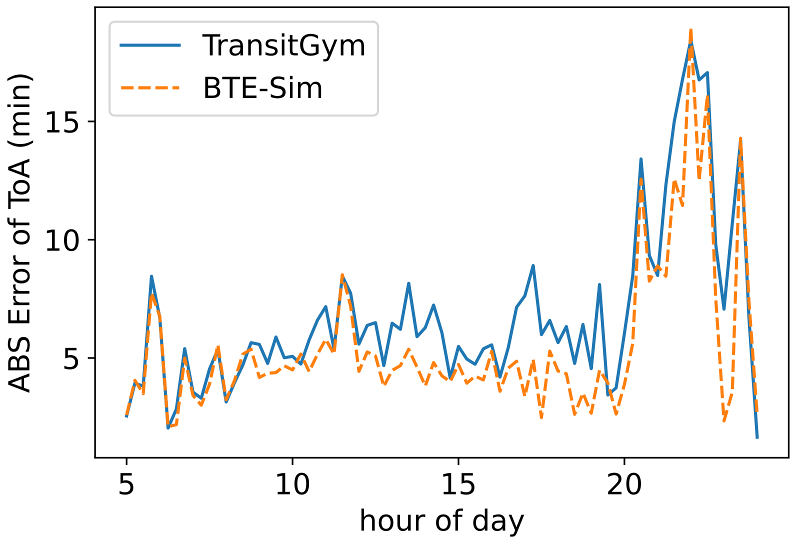
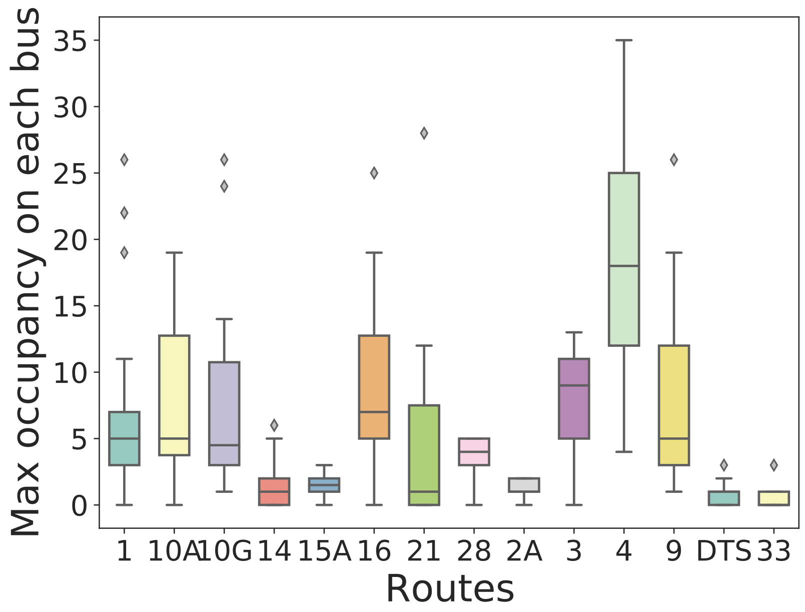
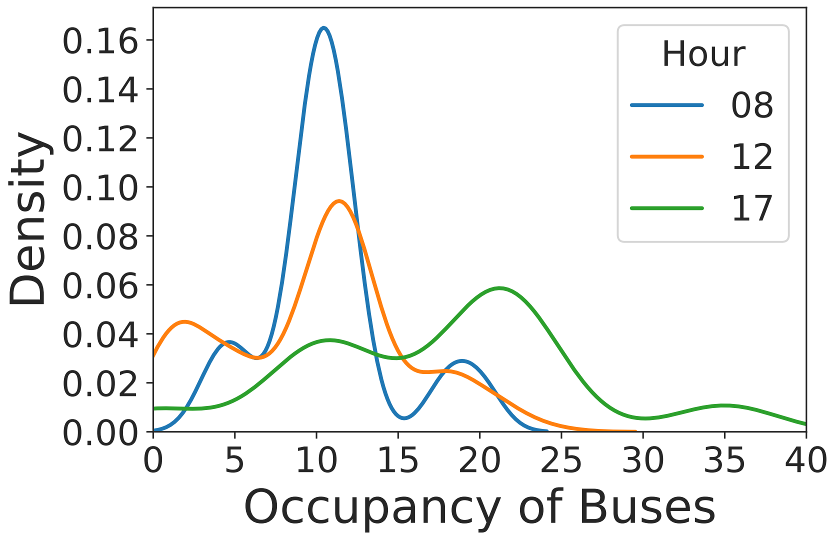


Figure 7.7 Comparing Transit-gym and BTE-Sim on absolute error of Time of Arrival



(a) Maximum occupancy



(b) Density of Route 4

Figure 7.8 Analysis examples for the transit system of Chattanooga on January 11, 2022, using BTE-Sim

7.2.3 Exp 3: BTE-Sim improves the simulation time

• ***Setting***. We aim to measure the improvement of BTE-Sim in different traffic demand scenarios. We vary the number of background vehicles from 100K to 1.4M.

• ***Result***. Table 7.1 shows that BTE-Sim is significantly faster than Transit-Gym. For a baseline of 100,000 vehicles, BTE-Sim run 12 times faster than Transit-Gym. As the number of vehicles increased, the difference in speed between BTE-Sim and Transit-Gym become even more pronounced. When the number of vehicles is quadrupled, Transit-Gym execution time increases by 8 times, while BTE-Sim computation time only grows by 2 times. This shows that BTE-Sim is highly adaptable to traffic volume changes and can be re-run for increased traffic scenarios without much time penalty.

#Vehicles	**Transit-Gym**	**BTE-Sim**
100K	27.7 minutes	2.21 minutes
400K	4 hours 4 minutes	5.11 minutes
800K	16 hours 51 minutes	7.81 minutes
1400K	41 hours 18 minutes	8.27 minutes

Table 7.1 Simulation time of Transit-Gym and BTE-Sim for scenarios with different number of vehicles

7.2.4 Exp 4: BTE-Sim simulates different dates

• ***Setting***. We use BTE-Sim to simulation a week, from January 10, 2002 to January 16, 2022. It is important to note that the transit settings for different dates are not the same. For example, a trip may be offered on Monday but not on other days.

• ***Result***. As shown in Figure 7.9, the simulated values are generally tightly clustered, with short inter-quartile ranges for each day. The exception is January 12, which has a wider dispersion. The mean and maximum absolute errors in the time of arrival (ToA) are consistently below 10 minutes. These results confirm that BTE-Sim has a low error margin when simulating regular traffic and transit operations on different days.

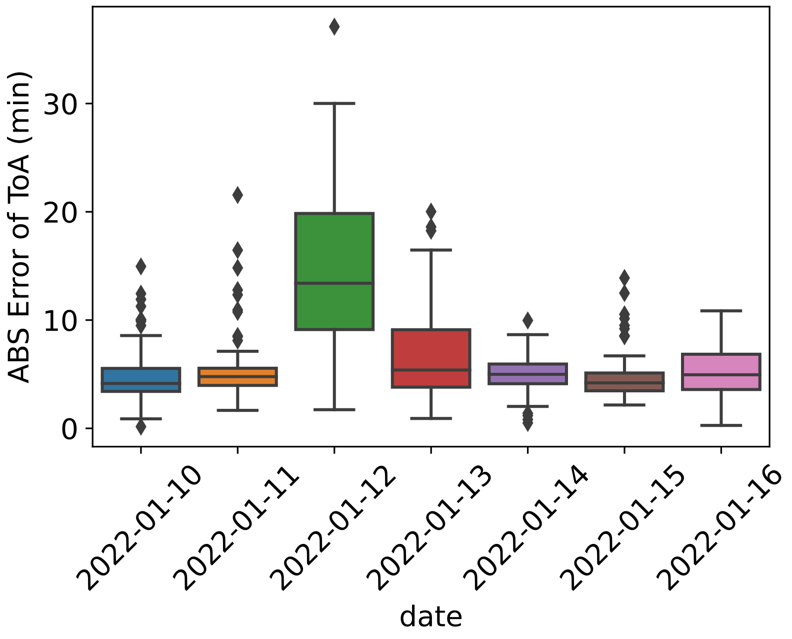


Figure 7.9 BTE-Sim over different dates

Part III

Neuroevolution for Training Neural Networks

CHAPTER 8
Neuroevolution for transportation applications

Neural networks and deep learning have been successfully applied to many transportation applications [41]. There are three main approaches to deep learning in transportation: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is used for prediction tasks, such as traffic speed prediction and travel time prediction. Unsupervised learning is used for anomaly detection and feature representation. Reinforcement learning is the state-of-the-art method for control tasks, such as traffic signal control (Part I) and autonomous driving. In general, deep learning is extremely popular in ITS, so even a small improvement in fundamental of neural networks, e.g., the way that neural networks are trained or designed, can have a big impact to the field.

8.1 Training neural networks

Training neural networks can be considered as an optimization process which find values of trainable parameters to minimize the loss function applied on the entire training dataset.

$$W = \underset{W}{\operatorname{argmin}} \left(\mathcal{L}(\operatorname{net}(W, X), y) \right), \tag{8.1}$$

where $\mathcal{L}$ is the lost function, W is the trainable parameters, X and y are the training dataset – inputs and labels respectively.

8.2 Training by Gradient Descent

Most neural networks have been trained by Gradient Descent (GD). The idea of GD is to repeat of taking small steps in the opposite direction of (approximate) the gradient.

$$w = w - \eta \cdot g\left(\frac{\partial \mathcal{L}(\text{net}(W, X_{batch}), y_{batch})}{\partial w}\right), \tag{8.2}$$

where $\frac{\partial \mathcal{L}}{\partial w}$ is gradient calculated by the backpropagation algorithm; W is trainable parameters; η is the learning rate; and g is the optimizer's function , e.g., Adam, SGD, and RMSprop.

8.3 Neuroevolution approaches

Unlike gradient descent (GD), which updates a single model iteratively, neuroevolution (NE) conducts an evolutionary process on a population of models. The following section provides more details on this training process.

Figure 8.1 shows the steps involved in training a neural network using neuroevolution. The first step is to initialize a population of models. Each model in the population is a randomly-initialized neural network. The next step is to evaluate the performance of each model. This is done by running the model on a training dataset and measuring its accuracy. The best models are then selected to be parents for the next generation. The parents are recombined to form new models. The new models are then mutated, which means that some of their weights are changed randomly. This process is repeated until the desired performance is achieved.

In this work, we investigate five NE algorithms including Genetic Algorithm, OpenAI Evolution Strategies, Augmented Random Search, Covariance Matrix Adaptation Evolution Strategy, and Policy Gradients with Parameter-based Exploration.

• **GA** – Genetic Algorithm [42] is inspired by the process of natural selection. It works by mimicking the biological operators of selection, mutation, and crossover.

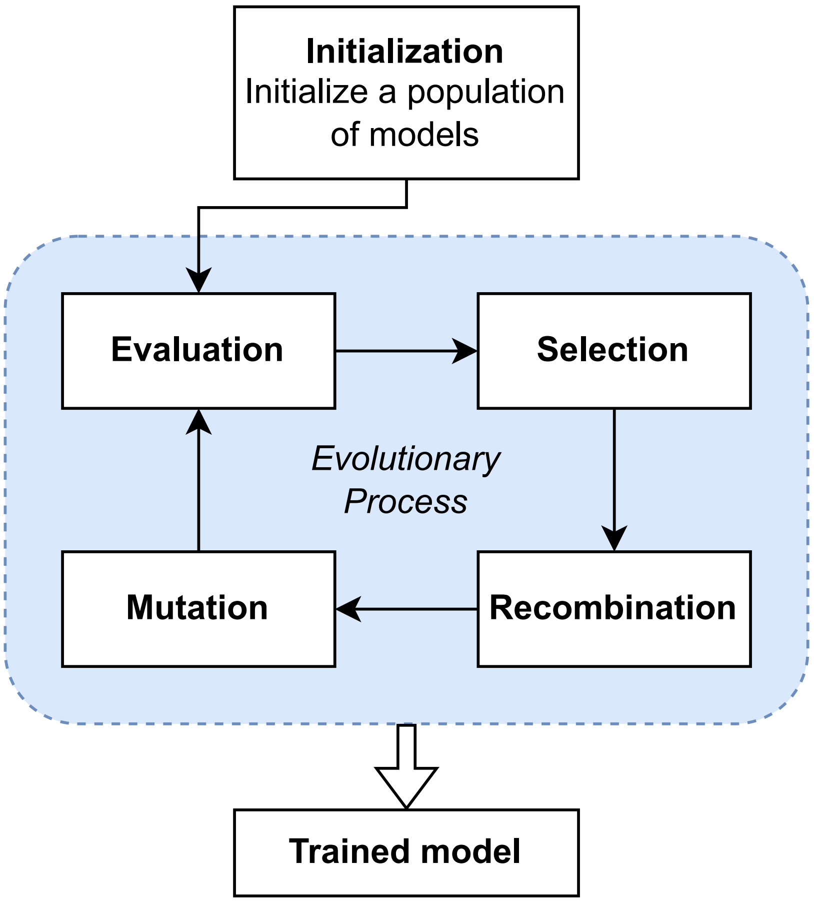


Figure 8.1 Flow chart of training neural network by NE

• **OpenES** – OpenAI Evolution Strategies [43], presented in Algorithm 1, is a way to find the best solution to a problem by iteratively mutating a population of individuals and then selecting the best individuals to continue the process. The mutation process adds a random amount of noise to the weight values of the individuals, which helps to prevent the algorithm from getting stuck in a local optimum. The selection process chooses the individuals with the highest fitness values to continue the process, which helps to ensure that the algorithm is finding the best possible solutions.

• **ARS** – Augmented Random Search [44] is an evolutionary algorithm that is considered an improvement over OpenES. It does this by considering both directions of samples, instead of just one, and by conducting selection, instead of only calculating the average of all solutions. This is beneficial because the goal of ARS is to maximize the collected fitness

Algorithm 1 OpenES algorithm

```
**Input**: learning rate $\eta$, noise standard deviation $\sigma$, initial weight $W_0$, population size $P$
for $t$ in 1 to #iters do
    Sample $\epsilon_1, ..., \epsilon_P \sim \mathcal{N}(0, I)$,
    $f_i \leftarrow$ Evaluate($W_t + \sigma\epsilon_i$) for $i = 1, ..., P$          ▷ Evaluation
    $W_{t+1} \leftarrow W_t + \eta \frac{1}{\sigma \cdot P} \sum_{j=1}^{P} f_j \epsilon_j$          ▷ Update

end for
```

values, and removing some bad solutions maybe a judicious strategy.

Algorithm 2 ARS algorithm

```
Input: learning rate $\eta$, noise standard deviation $\sigma$, initial weight $W_0$, population size $P$, number of top-performing solutions to use $b$
for $t$ in 1 to #iters do
    Sample $\epsilon_1, ..., \epsilon_P$with i.i.d. standard normal entries
    $f_i^{\pm} \leftarrow$ Evaluate($W_t \pm \sigma\epsilon_i$) for $i = 1, ..., P$          ▷ Evaluation
    Sort solutions by max$\left(f_i^+, f_i^-\right)$
    Select $b$ top solutions $\epsilon_{1..b}$          ▷ Selection
    $W_{t+1} \leftarrow W_t + \eta \frac{1}{\sigma \cdot b} \sum_{j=1}^{b} \epsilon_j \left[f_j^+ - f_j^-\right]$          ▷ Update

end for
```

• **CMA-ES** – Covariance Matrix Adaptation Evolution Strategy [45] CMA-ES is a stochastic optimization algorithm that changes the distribution parameters during searching, unlike OpenES and ARS, which sample from a static distribution. CMA-ES is designed for non-linear, non-convex, black-box optimization problems in continuous domains. Algorithm 3 provides a workflow of CMA-ES. The key idea of CMA-ES is the maximum-likelihood principle, which states that the distribution parameters are updated at every generation to maximize the likelihood of previously successful candidates.

Algorithm 3 CMA-ES algorithm

```
Input: population size $P$
Initialize: distribution parameters: $C = \mathbb{I}$, $\mu, \sigma$
for $t$ in 1 to #iters do
    for $i$ in 1 to $P$ do
        $W^i \sim \mathcal{N}(\mu, \sigma^2 C)$,          ▷ Multivariate normal distribution
        $f_i \leftarrow$ Evaluate($W_i$)
    end for
    $W^{\{1...P\}} \leftarrow W^{s(1)...s(P)}$          ▷ Sort solutions based on $f_i$
    $\mu \leftarrow$ Update_mean ($W_1, ...W_P$)
    $C \leftarrow$ Update covariance matrix
    $\sigma \leftarrow$ Update standard deviation
end for
```

• **PGPE** – Policy Gradients with Parameter-based Exploration [46] PGPE, described in Algorithm 4, is a method for estimating the gradient of the likelihood function by sampling directly from the parameter space. The parameters of the distribution are updated during the search, which helps to improve the accuracy of the estimates.

Algorithm 4 PGPE algorithm

```
Input: population size $P$
Initialize: distribution parameters: $\mu, \sigma$
for $t$ in 1 to #iters do
    for $i$ in 1 to $P$ do
        Sample $W^i \sim \mathcal{N}(\mu, \sigma^2 \mathbb{I})$
        $f_i \leftarrow$ Evaluate($W^i$)
    end for
    $\boldsymbol{T} = [t_{ij}]_{ij}$ with $t_{ij} := \left(W_i^j - \mu_i\right)$
    $\boldsymbol{S} = [s_{ij}]_{ij}$ with $s_{ij} := \frac{t_{ij}^2 - \sigma_i^2}{\sigma_i}$
    $\boldsymbol{r} = [(f_1 - b), ..., (f_P - b)]^T$
    Update $\mu = \mu + \eta \boldsymbol{T} \boldsymbol{r}$
    Update $\sigma = \sigma + \eta \boldsymbol{S} \boldsymbol{r}$
    Update baseline $b$ accordingly
end for
```

8.4 Experiments for Traffic prediction

We investigate the working of NE in the traffic prediction task which is the most popular supervised-learning job in transportation.

• **Dataset**. We experiment on the PeMS dataset which includes data from 11,160 detectors in California [47]. To reduce the computing requirements, 50 sensors within four weeks are randomly selected. The first three weeks of data are used for training while the last week is for testing. Finally, we aggregate the 30-second data frequency into 5-minute by getting average values.

• **Baselines**. To compare the two approaches, we use the same neural network architecture, proposed by [48], for both neuroevolution and gradient-based approaches. The network consists of a LSTM layer with 32 hidden units and a fully connected layer with 32 units. In total, the model has 6,081 trainable parameters. The activation function is ReLU and the optimizer is RMSprop with the default learning rate of 0.001. The training and testing data are the same, the only difference is the training process. Since the patterns in time-series data of various sensors are different, we train a separate model for each sensor. Each model is trained with 500 epochs and the best model is saved during training.

- **Algorithm effects**. Figure 8.2 depicts the average MSE over iterations. The baseline is models trained by GD. In general, most NE methods outperform GD on the training loss. Among NE algorithms, ARS and PGPE converge fastest after 40K iterations and also reach the lowest training loss. Table 1 shows the number of time series where each training algorithm achieved the best performance on the validation set. PGPE is the best method, achieving the lowest testing MSE on 18 time series. GD outperforms NE methods on only 1 time series, which is 2% of the total dataset. Although the average training loss of ARS is better than CMA-ES, CMA-ES obtained the best testing MSE for 16 time series, compared to 8 time series by ARS. This suggests that ARS is not good at generalization and is more likely to overfit the training data. Furthermore, Table 8.2 provides the average training time. Generally, GD's training time is shorter than all NE's.

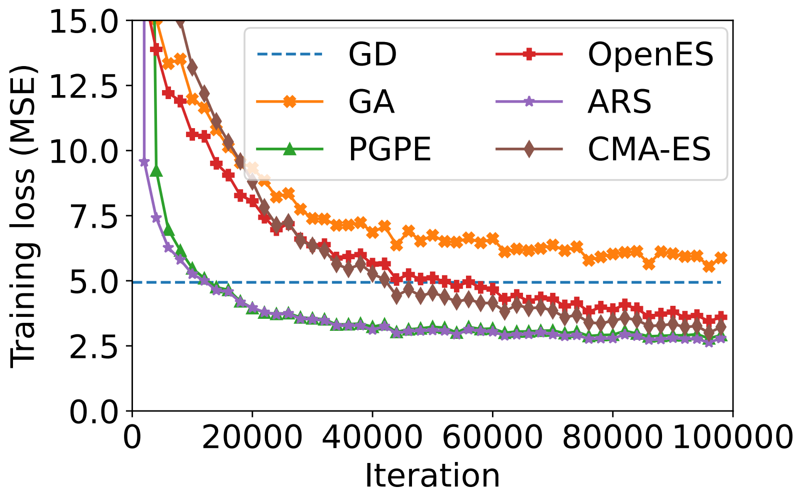


Figure 8.2 Average MSE of 50 time series during training

Method	GD	GA	**PGPE**	OpenES	ARS	CMA-ES
#TimeSeries	1	5	**18**	2	8	16

Table 8.1 Number of time series that methods achieve the best performance on the testing data

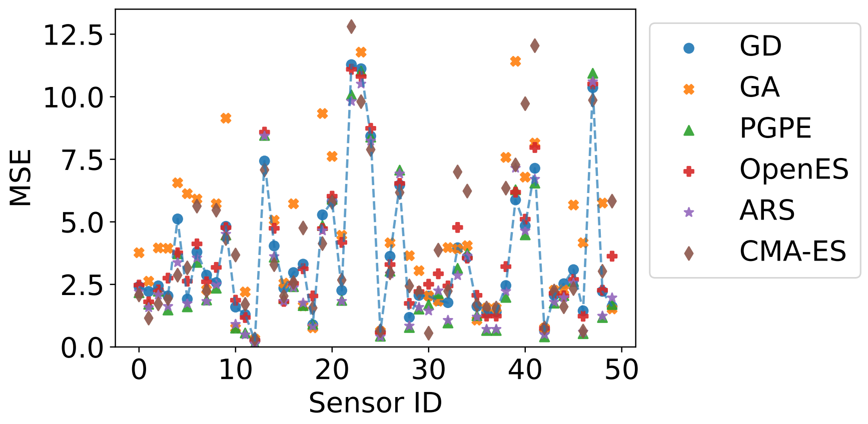


Figure 8.3 MSE on the testing data of 50 selected time series

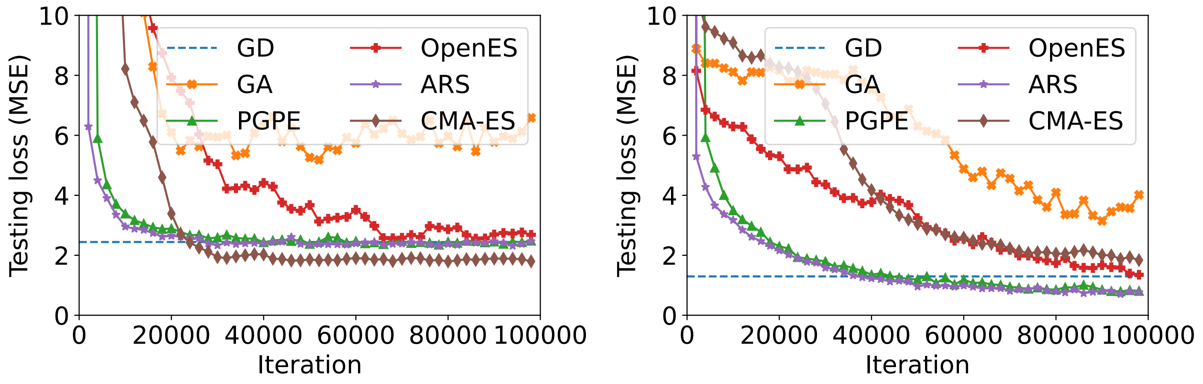


Figure 8.4 Testing loss curves of Sensor 2 and 11 over iterations

• **Population size effects**. To understand effects of the population size, we train models with different settings for the population size, but keep all other hyperparameters the same. We only focus on PGPE, as it was the algorithm that produced the best results in the previous experiment.

All population sizes outperform GD, but that the best performance is achieved with a population size of 512. This is surprising because a larger population size is generally expected to perform better. However, note that our population sizes are considered quite small for the search space of $\mathbb{R}^{6081}$. For example, a population size of 200 is used for the search space of $\mathbb{R}^3$ in [49]. This suggests that there is a trade-off between population size and computing capacity. A larger population size can provide a better global ability to avoid local minima, but it is also more computationally expensive.

Method	Total training time (s)	One-iteration training time (ms)
GD	**89.24**	**1.89**
GA	332.08	3.32
PGPE	228.47	2.28
OpenES	272.75	2.73
ARS	265.16	2.65
CMA-ES	244.76	2.45

Table 8.2 Average training time for each time series

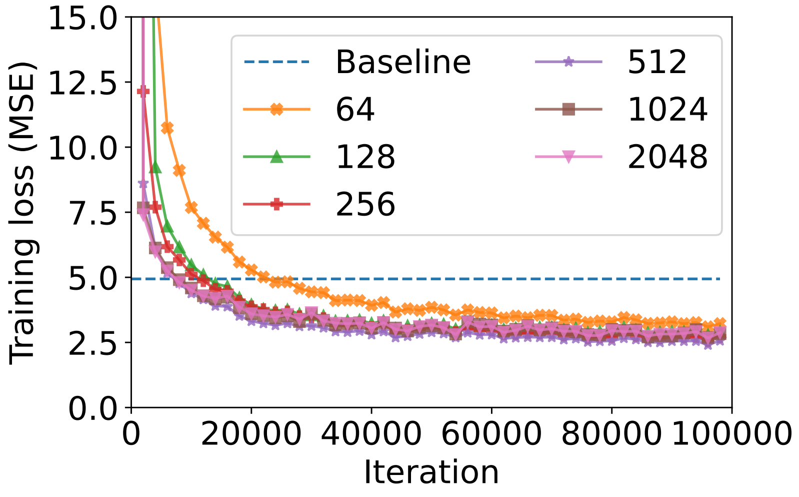


Figure 8.5 PGPE's performance with various population sizes

• **Learning rate effects**. We study the effects by varying the learning rate while keeping other hyperparameters.

Figure 8.6 shows the performance of PGPE with different learning rates. The loss value is a measure of how well the model fits the training data, and convergence time is the time it takes for the model to reach a stable state. The baseline is the performance of GD. This suggests that a lower learning rate allows the model to converge more slowly and avoid overfitting. The learning rate of 0.1 obtained an unstable result because it was too high, causing the model to oscillate between different minima. The learning rates of 0.01 and 0.001 have better performance than the baseline because they are lower and allow the model to converge more slowly. The setting with the learning rate of 0.001 outperforms the baseline by around 35% on the training loss, indicating that it is the best performing setting.

• **An interesting visualization**. We use a new technique called filter-wise nor-

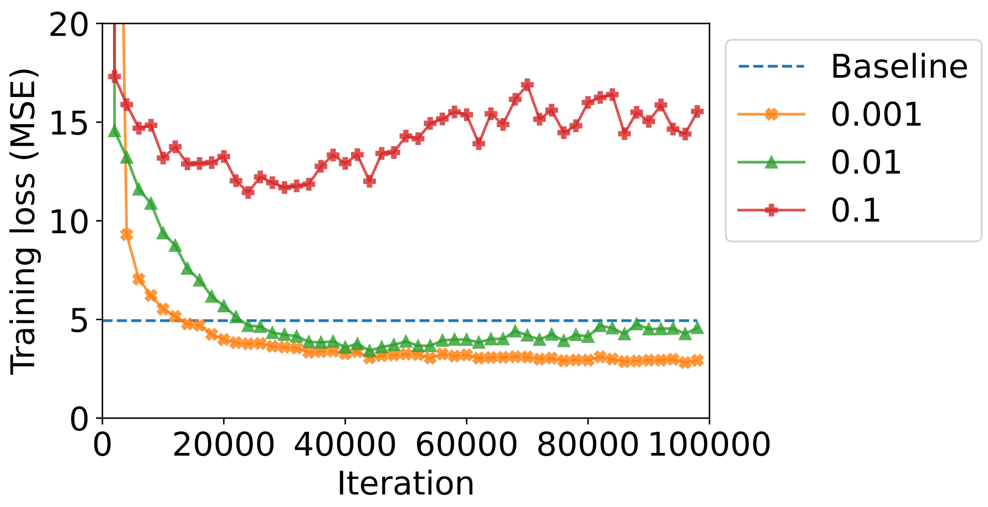


Figure 8.6 PGPE's performance with various learning rates

malization [50] to visualize the loss surfaces of LSTM networks for traffic prediction. This technique helps us to see more detail in the loss surfaces than previous techniques, which were limited by the complexity of the loss function. We can use this technique to compare loss surfaces for different network architectures and training datasets. In this paper, we focus on comparing loss surfaces for the same network architecture but different time series.

The loss surfaces of the time series of Sensors 04 and 39 are shown in Figures 8.7 and 8.8, respectively. As shown in Figure 8.3, NE outperforms GD on Sensor 04, while GD is better than NE for Sensor 39. Figures 8.9 and 8.10 show the contour lines when the loss value is less than 50. In general, the loss function of Sensor 04 is non-convex, with many local minimums. The loss function of Sensor 39 is much more convex. GD is not guaranteed to converge to the global minimum [51], so it is challenging for GD to find the global minimum of the non-convex loss function of Sensor 04. In contrast, NE is able to escape local minimums, so it is better than GD for Sensor 04. For Sensor 39, which has a convex loss function, GD is better than NE because GD can easily find the global minimum.

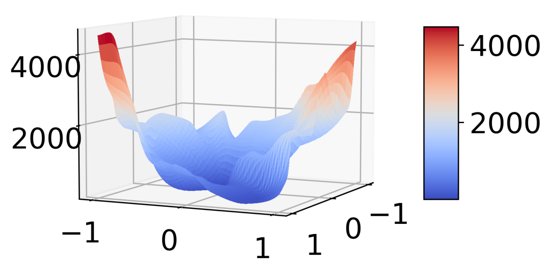


Figure 8.7 The loss surface of Sensor 04

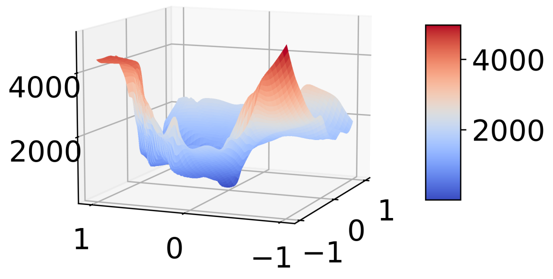


Figure 8.8 The loss surface of Sensor 39

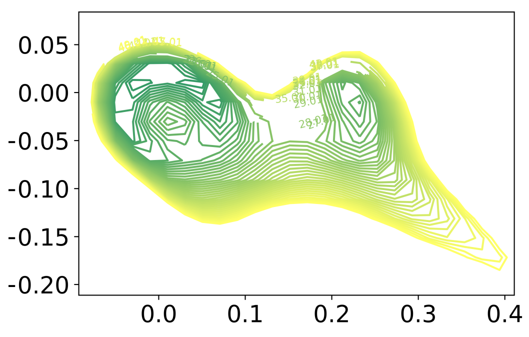


Figure 8.9 The contour-line visualization for the loss surface of Sensor 04, in which NE outperforms GD

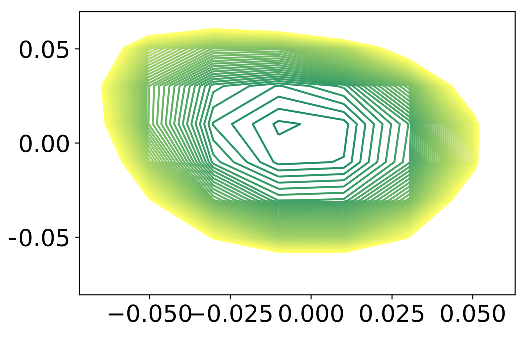


Figure 8.10 The contour-line visualization for Sensor 39's loss surface, where GD is better than NE

8.5 Discussion

We have introduced a new method for training neural networks using neuroevolution. Our method outperforms gradient descent-based learning methods by up to 50% for the traffic prediction task. We have also provided visualizations of the loss surfaces, which helps us to understand why neuroevolution is better than gradient descent in some cases and vice versa. Although our neuroevolution framework is highly scalable, our current settings for the population size are still relatively small for the vast search spaces of training neural networks. In the future, we plan to investigate the functionality of larger populations when we have sufficient computing resources. Additionally, we are excited about the potential of neuroevolution for training graph neural networks, which is an area of research that is still under development.

CHAPTER 9
Discussion

In this thesis, we investigated various problems in reinforcement learning for traffic signal control and public transit simulation. For TSC, we introduced TSLib, which is designed to be modular and reusable, so that researchers can quickly implement and evaluate new ideas in TSC. We also demonstrated the operation of TSLib by a benchmark on multiple network scales. Subsequently, we proposed a novel method named WorldLight using world models for automated feature engineering. WorldLight slightly outperformed the previous methods in certain cases and showed its stability on various reward functions. Additionally, we implemented RL-based TSCs on a digital twin of the smart corridor at Chattanooga, TN, USA and proposeed a TSC system supporting lifelong assessment. The RL-based TSCs improved average queue length, speed, travel time, and total fuel consumption by 22.23%, 6.34%, 4.29%, and 3.69%, compared to Actuated Controls. However, there is still a path from digital twins to the real-world field that is need to be investigated. For example, there are a few real-to-sim issues such as generalization, robustness of RL models, noisy features extracted from real-time computer vision systems, and latency of communication.

Regarding to public transit simulation, we introduced SIMCal which is a high-performance toolkit for calibrating traffic simulation. It is designed to help researchers and practitioners to quickly and easily calibrate traffic simulations. Additionally, we addressed a novel simulation problem – how to efficiently conduct a new simulation where only the transit system's setting changes while the rest of the transportation system remains stable. However, all experiments were conduct for Chattanooga only. The more extensive experiment across multiple cities is necessary to improve the simulation procedure.

Finally, for training neural networks, we proposed a novel approach using Neuroevolution, which outperforms Gradient Descent methods by up to 50% on training loss. Although our neuroevolution framework is highly scalable, our current settings for the population size are still relatively small for the vast search spaces of training neural networks. In the future, we plan to investigate the functionality of larger populations when we have sufficient computing resources. Additionally, we are excited about the potential of neuroevolution for training graph neural networks, which is an area of research that is still under development.

REFERENCES

[1] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flotterod, R. Hilbrich, L. Lucken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic traffic simulation using sumo," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2575–2582.

[2] H. Zhang, S. Feng, C. Liu, Y. Ding, Y. Zhu, Z. Zhou, W. Zhang, Y. Yu, H. Jin, and Z. Li, "Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 3620–3624. [Online]. Available: https://doi.org/10.1145/3308558.3314139

[3] D. Schrank, L. Albert, B. Eisele, and T. Lomax, *2021 Urban Mobility Report.* Texas A&M Transportation Institute, 2021.

[4] S. A. A. Shah, E. Ahmed, M. Imran, and S. Zeadally, "5g for vehicular communications," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 111–117, 2018.

[5] E. van der Pol and F. A. Oliehoek, "Coordinated deep reinforcement learners for traffic light control," 2016.

[6] H. Wei, G. Zheng, H. Yao, and Z. 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*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2496–2505. [Online]. Available: https://doi.org/10.1145/3219819.3220096

[7] C. Chen, H. Wei, N. Xu, G. Zheng, M. Yang, Y. Xiong, K. Xu, and Zhenhui, "Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control," in *AAAI*, 2020.

[8] R. Sun, R. Gui, H. Neema, Y. Chen, J. Ugirumurera, J. Severino, P. Pugliese, A. Laszka, and A. Dubey, "Transit-gym: A simulation and evaluation engine for analysis of bus transit systems," in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2021, pp. 69–76.

[9] R. Sen, A. K. Bharati, S. Khaleghian, M. Ghosal, M. Wilbur, T. Tran, P. Pugliese, M. Sartipi, H. Neema, and A. Dubey, "E-transit-bench: Simulation platform for analyzing electric public transit bus fleet operations," in *Proceedings of the Thirteenth ACM International Conference on Future Energy Systems*, ser. e-Energy '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 532–541. [Online]. Available: https://doi.org/10.1145/3538637.3539586

[10] K. Seyedmehdi, N. Himanshu, S. Mina, T. Toan, S. Rishav, and D. Abhishek, "Calibrating real-world city traffic simulation model based on vehicle speed data," *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2023.

[11] T. V. Tran, T.-N. Doan, and M. Sartipi, "Tslib: A unified traffic signal control framework using deep reinforcement learning and benchmarking," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 1739–1747.

[12] C. Gershenson, "Self-organizing traffic lights," *Complex Systems*, vol. 16, no. 1, 2004.

[13] P. Varaiya, "Max pressure control of a network of signalized intersections," *Transportation Research Part C: Emerging Technologies*, vol. 36, pp. 177–195, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X13001782

[14] S. Mousavi, M. Schukat, P. Corcoran, and E. Howley, "Traffic light control using deep policy-gradient and value-function based reinforcement learning," *IET Intelligent Transport Systems*, vol. 11, 04 2017.

[15] X. Liang, X. Du, G. Wang, and Z. Han, "A deep reinforcement learning network for traffic light cycle control," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1243–1253, 2019.

[16] U. D. of Transportation Federal Highway Administration. (2016) Next generation simulation (ngsim) vehicle trajectories and supporting data. [Online]. Available: http://doi.org/10.21949/1504477

[17] Z. J. Li. (2023) Open datasets on cityflow. [Online]. Available: https://traffic-signal-control.github.io/dataset.html

[18] L. Codecà and J. Härri, "Monaco sumo traffic (most) scenario: A 3d mobility scenario for cooperative its," *SUMO User Conference, Simulating Autonomous and Intermodal Transport Systems*, 2018.

[19] G. Zheng, X. Zang, N. Xu, H. Wei, Z. Yu, V. Gayah, K. Xu, and Z. Li, "Diagnosing reinforcement learning for traffic signal control," 2019.

[20] H. Wei, C. Chen, G. Zheng, K. Wu, V. Gayah, K. Xu, and Z. 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*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1290–1298. [Online]. Available: https://doi.org/10.1145/3292500.3330949

[21] J. Ault and G. Sharon, "Reinforcement learning benchmarks for traffic signal control," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. [Online]. Available: https://openreview.net/forum?id=LqRSh6V0vR

[22] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf

[23] D. R. Ha and J. Schmidhuber, "World models," *ArXiv*, vol. abs/1803.10122, 2018.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[25] A. Harris, J. Stovall, and M. Sartipi, "Mlk smart corridor: An urban testbed for smart city applications," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 3506–3511.

[26] H. Wei, G. Zheng, V. Gayah, and Z. Li, "Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation," *SIGKDD Explor. Newsl.*, vol. 22, no. 2, p. 12–18, jan 2021. [Online]. Available: https://doi.org/10.1145/3447556.3447565

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 1928–1937.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[29] Z. Fang, F. Zhang, T. Wang, X. Lian, and M. Chen, "Monitorlight: Reinforcement learning-based traffic signal control using mixed pressure monitoring," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM)*, 2022.

[30] A. Saroj, T. V. Trant, A. Guin, M. Hunter, and M. Sartipi, "Optimizing traffic controllers along the mlk smart corridor using reinforcement learning and digital twin," in *2022 IEEE 2nd International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 2022, pp. 1–2.

[31] A. Stevanovic, S. Alshayeb, and S. Patra, "Fuel consumption intersection control performance index," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2675, 04 2021.

[32] T. V. Tran and M. Sartipi, "Single camera-enabled reinforcement learning traffic signal control system supporting life-long assessment," *2023 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2023.

[33] G. Jocher, A. Chaurasia, and J. Qiu, "Yolo by ultralytics," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[34] T. V. Tran, S. Khaleghian, J. Zhao, and M. Sartipi, "Simcal: A high-performance toolkit for calibrating traffic simulation," in *2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 2895–2902.

[35] K. Sastry, D. Goldberg, and G. Kendall, *Genetic Algorithms*. Boston, MA: Springer US, 2005, pp. 97–125.

[36] J. Kennedy, *Particle Swarm Optimization.* Boston, MA: Springer US, 2010, pp. 760–766.

[37] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Stochastic Algorithms: Foundations and Applications*, O. Watanabe and T. Zeugmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 169–178.

[38] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[39] L. Li and X. M. Chen, "Vehicle headway modeling and its inferences in macroscopic/microscopic traffic flow theory: A survey," *Transportation Research Part C: Emerging Technologies*, vol. 76, pp. 170–188, 2017.

[40] R. Sen, T. Tran, S. Khaleghian, P. Pugliese, M. Sartipi, H. Neema, and A. Dubey, "Bte-sim: Fast simulation environment for public transportation," in *2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 2886–2894.

[41] H. Nguyen, M. Kieu, T. Wen, and C. Cai, "Deep learning methods in transportation domain: A review," *IET Intelligent Transport Systems*, vol. 12, 07 2018.

[42] M. Mitchell, *An Introduction to Genetic Algorithms.* Cambridge, MA, USA: MIT Press, 1998.

[43] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *ArXiv*, vol. abs/1703.03864, 2017.

[44] H. Mania, A. Guy, and B. Recht, "Simple random search of static linear policies is competitive for reinforcement learning," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS' 18)*, 2018.

[45] N. Hansen, "The cma evolution strategy: A comparing review," in *Towards a New Evolutionary Computation*, 2006.

[46] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Policy gradients with parameter-based exploration for control," in *Proceedings of the 18th International Conference on Artificial Neural Networks (ICANN)*, 2008.

[47] T. Choe, A. Skabardonis, and P. Varaiya, "Freeway performance measurement system: Operational analysis tool," *Transportation Research Record*, vol. 1811, no. 1, pp. 67–75, 2002.

[48] R. Fu, Z. Zhang, and L. Li, "Using lstm and gru neural network methods for traffic flow prediction," in *Proceedings of 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, 2016.

[49] O. Roeva, S. Fidanova, and M. Paprzycki, "Influence of the population size on the genetic algorithm performance in case of cultivation process modelling," in *2013 Federated Conference on Computer Science and Information Systems*, 2013, pp. 371–376.

[50] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Proceedings of Neural Information Processing Systems (NeurIPS' 18)*, 2018.

[51] S. Ruder, “An overview of gradient descent optimization algorithms,” 2016. [Online]. Available: https://arxiv.org/abs/1609.04747

VITA

Toan Tran was born and grew up in Vietnam. He obtained a bachelor degree in Computer Engineering at Ho Chi Minh City University of Technology. Currently, he is a master student in Computer Science at University of Tennessee at Chattanooga. He has a broad interest in data mining and its applications. After UTC, he continues his study at Emory University.