

DEEP REINFORCEMENT LEARNING APPROACH FOR SMART TRAFFIC SIGNAL CONTROL

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

POLAMREDDY KARTHIK REDDY (Reg. No.40110950)

SANDHYA D.K (Reg. No.40111129)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

CATEGORY-1 UNIVERSITY BY UGC

**Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600119**

APRIL - 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category - I University by UGC

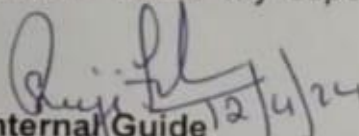
Accredited "A++" by NAAC | Approved by AICTE

www.sathyabama.ac.in

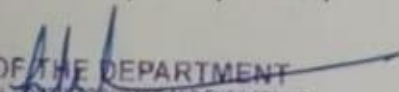
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of POLAMAREDDY KARTHIK REDDY (Reg. No 40110950) and SANDHYA D.K (Reg. No 40111129) who carried out the Project entitled "DEEP REINFORCEMENT LEARNING FOR SMART TRAFFIC SIGNAL CONTROL" under my supervision from November 2023 to April 2024.

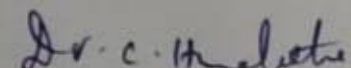

Internal Guide

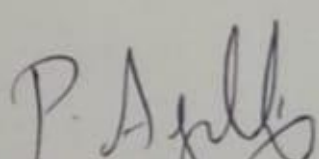
Dr. L. SUJHELEN, M.E., Ph.D.,


HEAD OF THE DEPARTMENT
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY

Dr. L. LAKSHMANAN, M.E., Ph.D.,
Jepplaar Nagar, Rajiv Gandhi Salai,
Chennai - 600 119.

Submitted for Viva voce Examination held on 13-April-2024


Internal Examiner

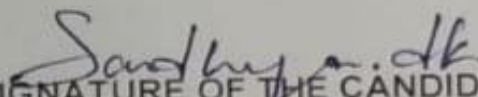

External Examiner

DECLARATION

I, SANDHYA D.K (Reg. No.40111129) hereby declare that the Project Report entitled "DEEP REINFORCEMENT LEARNING APPROACH FOR SMART TRAFFIC SIGNAL CONTROL" done by me under the guidance of Dr. L. SUJHELEN, M.E., Ph.D., is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 13-04-2024

PLACE: Chennai


SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **The Board of Management of Sathyabama Institute of Science and Technology** for their encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA M.E., Ph. D, Dean, School of Computing, Dr. L. LAKSHMANAN M.E., Ph.D., Head of The Department,** Department of Computer Science and Engineering for providing me the necessary support the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. L. SUJHELEN, M.E., Ph. D,** for her valuable guidance, suggestions, and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Traffic congestion is becoming a critical issue with the increasing usage of automobiles. The rapid increase in automobiles has naturally led to traffic congestion all over the world, forcing people to sit idle in their vehicles, consuming more fuel, and increase of air pollution. Traffic light control policies are not optimized, leading vehicles to wait pointlessly for non-existent traffic to pass on the crossing road. Currently, most traffic signal systems are using fixed timing plans, which do not depend on the vehicle count. Designing efficient transportation systems is crucial to save time and reduce the impact on the environment by minimizing carbon emissions. Traffic light control policies can be greatly improved by implementing Deep learning and Reinforcement learning concepts. In recent years, Reinforcement Learning (RL) has emerged as a promising technique for traffic signal control, because of its ability to learn optimal control policies directly from experience. In this project traffic is efficiently controlled by using Reinforcement Learning, this will reduce unwanted delays, congestion, fuel consumption, and pollution.

TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	v
	LIST OF FIGURES	viii
1	INTRODUCTION	1
	1.1 Overview	1
	1.2 Purpose of Deep Learning	1
	1.3 Problem Statement	4
	1.4 Objective	5
2	LITERATURE SURVEY	6
	2.1 Critic on Literature Review	8
	2.2 Existing system of Traffic Signal Control System	10
3	REQUIREMENT ANALYSIS	12
	3.1 Limitations of TLCS using Fixed timings	12
	3.2 Software Requirements	12
	3.3 Hardware Requirements	14
4	DESCRIPTION OF PROPOSED SYSTEM	15
	4.1 Proposed System	15
	4.2 Architecture of Traffic Light Control System using RL	16
	4.3 Description of modules in System Architecture	16
5	IMPLEMENTATION DETAILS	20
	5.1 Implementation of the proposed model/Algorithm	20
	5.2 Workflow involved in Traffic Signal Control using RL	34
	5.3 Methodology Involved in Workflow	35
6	RESULTS AND DISCUSSIONS	38
	6.1 Simulation Results	38

6.2	Comparing existing System and Reinforcement Model	39
7	CONCLUSION AND FUTURE SCOPE	45
7.1	Conclusion	45
7.2	Future Scope	45
	REFERENCES	47
	APPENDIX	49
	A. Source code	49

LIST OF FIGURES

Figure No.	Figure Name	Page No.
1.1	A Venn-diagram of AI, ML and Deep Learning	2
1.2	Basic Structure of Deep Learning	3
4.1	Architecture for Deep Reinforcement Learning	16
5.1	Environment Creation using SUMO	21
5.2	Weibull Distribution graph with a slope of 2	22
5.3	Diagram representing Deep Q-Learning	32
5.4	Workflow of Traffic Signal Control using DRL	35
6.1	Simulation Results for TLCS using RL	38
6.2	Simulation Results for TLCS using RL	39
6.3	Delay vs Episodes Plot for existing system	40
6.4	Delay vs Episodes Plot for Reinforcement Model	41
6.5	Q length vs Episodes Plot for existing system	42
6.6	Q length vs Episodes Plot for Reinforcement Model	42
6.7	Reward vs Episodes plot for Existing system	43
6.8	Reward vs Episodes Plot for Reinforcement model	44

CHAPTER 1

INTRODUCTION

1.1 Overview

Traffic Light Control System is the process of managing the flow of vehicular and pedestrian traffic at intersections or junctions, with the goal of ensuring safe and efficient movement of traffic. Traffic signals are typically used to regulate the right-of-way for different traffic streams and to manage conflicts between them. The operation of traffic signals is typically based on a fixed-time or actuated control strategy. Fixed-time control involves setting the signal timings in advance, based on historical traffic patterns or design standards. Actuated control, on the other hand, involves using sensors or detectors to detect the presence of vehicles or pedestrians, and adjusting the signal timings in real-time to respond to changing traffic conditions. The choice of control strategy depends on the specific traffic situation and the goals of the traffic management agency. Fixed-time control is often used in areas with relatively stable traffic patterns, such as suburban or rural areas. Traffic signal control is a complex task that involves balancing competing objectives, such as minimizing delays, reducing congestion, and improving safety.

In recent years, there has been growing interest in using Machine Learning techniques, such as Reinforcement Learning, to optimize traffic signal control. This approach has the potential to improve the efficiency and effectiveness of traffic signal control, as well as to adapt to changing traffic conditions in real-time. The context of the project is traffic signal control at one intersection. Q-Learning algorithm with Deep Neural Networks is used to optimize traffic flow at a traffic signal intersection.

1.2 Purpose of Deep Learning

Deep Learning is a subfield of Machine Learning that involves training Artificial Neural Networks with multiple layers to perform tasks such as Image Classification, Speech Recognition, Natural Language Processing, and more. The term "Deep"

refers to the fact that these neural networks have many layers, which allow them to learn complex representations of data.

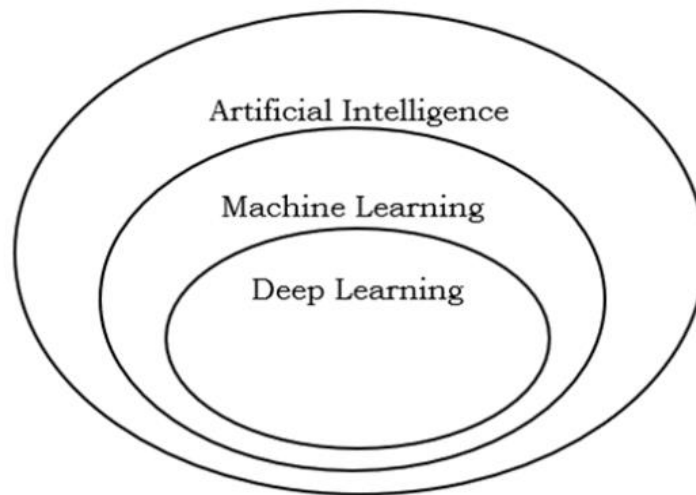


Fig:1.1 A Venn-diagram of AI, ML and Deep Learning

Fig:1.1 shows the Venn diagram for Artificial Intelligence, Machine Learning and Deep Learning. Deep Learning has become increasingly popular in recent years due to its ability to achieve state-of-the-art performance on a wide range of tasks. This is in part due to the availability of large amounts of data and the increase in computing power that allows for the training of these complex models. Overall, Deep Learning has become an important tool for solving a wide range of Machine Learning problems, and its ability to automatically learn complex features from data has led to significant advances in fields such as Computer Vision, Natural Language Processing, and Speech Recognition.

- **The Basic Structure of Deep Learning**

Deep Learning includes various Neural Networks that possess different layers, such as input layers, hidden layers, and output layers. The input layer accepts input data; hidden layers are used to find any hidden pattern and feature from the data, and output layers show the expected results. Figure-1.2 shows the Basic structure of Deep Learning with Input layer, Output layer and Hidden layers.

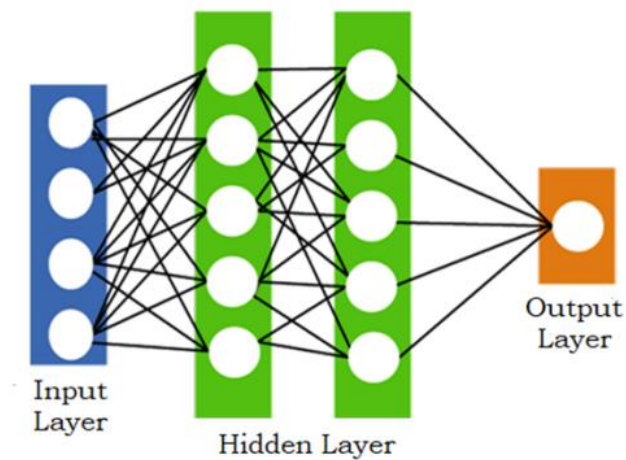


Fig:1.2 Basic Structure of Deep Learning

- **Advantages**

1. **Superior Performance:** Deep Learning algorithms have shown superior performance in a wide range of applications such as Image and Speech Recognition, Natural Language Processing, and even playing games such as Chess and Go.
2. **Adaptability:** Deep Learning algorithms are highly adaptable and can learn from new data, making them ideal for applications that require continuous learning.
3. **Automation:** Deep Learning algorithms can automate complex tasks, reducing the need for human intervention. **Feature Extraction:** Deep Learning algorithms can automatically extract relevant features from data, which can be time-consuming and difficult to do manually.
4. **Scalability:** Deep Learning algorithms can scale to handle large amounts of data, making them ideal for big data applications.
5. **Improved accuracy:** Deep Learning algorithms have been shown to achieve state-of-the-art performance on a variety of tasks, such as Image Classification, Speech Recognition, Natural Language Processing, and Game Playing. This is due to their ability to learn complex and nonlinear relationships between inputs and outputs from large datasets and to generalize well to new examples.

6. Flexibility: Deep Learning algorithms can be applied to a wide range of data modalities, including images, videos, text, speech, and sensor data, and can be adapted to different tasks, such as classification, regression, clustering, and Reinforcement Learning.

- **Disadvantages**

- 1. Data Requirement:** Deep Learning algorithms require large amounts of data to learn effectively. This can be a challenge for smaller organizations or those with limited resources.
- 2. Time-Consuming:** Training Deep Learning models can be time consuming, especially for larger datasets. It can take hours, days, or even weeks to train a model, depending on the complexity of the problem and the amount of data available.
- 3. Lack of Transparency:** Deep Learning algorithms can be difficult to interpret, making it challenging to understand how they arrive at their conclusions. This can be a concern in applications where transparency is important, such as in legal or medical fields.
- 4. Overfitting:** Deep Learning algorithms are prone to overfitting, where the model performs well on the training data but poorly on new data. This can be a challenge to overcome, requiring additional data or techniques such as Regularization.
- 5. Resource-Intensive:** Deep Learning algorithms require significant computing power, including powerful GPUs and large amounts of memory.

1.3 Problem Statement

Traffic congestion is a common issue at intersections, leading to increased waiting times for vehicles and decreased overall traffic efficiency. Traditional traffic light control systems often operate on fixed timers or simple rule-based algorithms, which may not adapt well to changing traffic conditions. To address this problem, a structural model utilizing deep Q-learning is proposed. This model aims to empower a deep reinforcement learning agent to dynamically choose the optimal

traffic light phase at an intersection, with the objective of maximizing traffic flow effectiveness and minimizing vehicle waiting times.

1.4 Objective

The primary goal of this project is to develop a computer vision-based traffic light controller capable of dynamically responding to fluctuating traffic conditions. The current traffic flow lacks a predictable pattern, and conventional static signal timers exacerbate congestion issues. By creating a system that autonomously determines optimal green signal durations based on real-time traffic data, we aim to prioritize green signals for directions with higher average vehicle waiting times, thereby mitigating congestion, and reducing overall delays. This novel approach replaces outdated hardcoded signal systems, leading to decreased waiting times for vehicles, reduced fuel consumption, and consequently, improved air quality. The traffic signal controller acts as the reinforcement agent, with the reward system incentivizing traffic flow optimization and congestion reduction. Through simulation and data analysis, the controller learns from traffic patterns to make more effective decisions regarding signal changes. A key advantage lies in the system's adaptability, as it can continuously adjust its strategies based on real-time data, particularly beneficial in urban settings characterized by rapidly changing traffic conditions. Ultimately, employing reinforcement learning for traffic signal control holds promise for enhancing traffic flow efficiency and alleviating congestion, contributing positively to urban environments and residents' quality of life.

CHAPTER 2

LITERATURE SURVEY

(1) Li, Li, Yisheng Lv, and Fei-Yue Wang. “Traffic signal timing via deep reinforcement learning.”

In this paper, it is proven that Deep Q-Learning is more efficient when compared to Q-Learning. The Q-Learning algorithm uses a Table to store Q-values. Deep Q-Learning uses Convolution Neural Networks to store Q-values. Traditional Q-Learning uses a lookup table of all possible Q-values of state-action pairs and iteratively updates the Q-values. A two-lane traffic intersection is considered here. There are 32 neurons in the input layer, just 2 neurons in the output layer. Hence there are only two possible actions. There are 16 and 4 neurons in the hidden two layers, respectively. The activation function for hidden layers is the sigmoid function. They further compare the mean values of the obtained delay and the number of fully stopped vehicles of Reinforcement Learning and Deep Reinforcement Learning traffic controllers during simulations 20 times. They found that the average delay can be reduced by about 14% if the Deep Reinforcement Learning method is used instead of the conventional Reinforcement Learning method. In morning peak hours, a vehicle may spend about 13 s less to pass the studied intersection. Moreover, the number of fully stopped vehicles is reduced by 20% when the Deep Reinforcement Learning method is applied. Particularly, the number of fully stopped vehicles is reduced by 15% vehicles in the morning peak hours.

(2) Salaymeh, Areej, Loren Schwiebert, and Stephen Remias. “MultiAgent Reinforcement Learning for Optimizing Traffic Signal Timing.”

In this paper, SUMO (Simulation of Urban Mobility) is used for traffic generation. To test the model against traffic, two vehicle-generating modes are used. (1) steady vehicle flow (2) variable flow. The steady vehicle flow generates a steady number of cars during episodes. The variable flow tries to mimic reality

by using a Rayleigh distribution with $\gamma = 2$. In the Rayleigh distribution they used, the car flow rises until a certain point, then starts to decline. Within each episode, they considered different ratios of how many cars go in each direction. The Simulation of the Urban Mobility (SUMO) engine is used for the generation of traffic. SUMO enables the simulation of transport networks, including vehicles, and public transportation. SUMO provides a range of support resources that manage tasks such as route discovery, analysis, network import, and emission estimation. Simulation of Urban Mobility is an open-source, highly portable, microscopic, and continuous traffic simulation package designed to handle larger networks. SUMO is developed by the German Aerospace Center and community users in the year 2001. SUMO allows the users to create a road Intersection model.

(3) Bhawe, Neel, Aniket Dhagavkar, Kalpesh Dhande, Monis Bana, and Jyoti Joshi. "Smart signal–adaptive traffic signal control using Reinforcement Learning and object detection."

In This paper Smart traffic control using a Q-Learning algorithm. The model developed using the Q-Learning algorithm has two phases, training or Learning and testing. In the training phase model is trained for a certain number of episodes, and at the end of each episode average waiting time of vehicles is calculated. In the testing phase, the model is tested for a traffic scenario. In this paper, the SARSA algorithm is applied to a simulated traffic control problem. The use of this algorithm lowered the average vehicle waiting time by 29%. When the Q-Learning algorithm is applied to a simulated traffic control problem, it lowered the average vehicle waiting time by 44%. Using the Q-Learning algorithm, there was a 38.4% decrease in the Mean number of cars waiting at any point in time and a 44% decrease in the Mean waiting time for cars.

- (4) Li, Congcong, Fei Yan, Yiduo Zhou, Jia Wu, and Xiaomin Wang. "A regional traffic signal control strategy with Deep Reinforcement Learning".**

In this paper, a Deep Reinforcement Learning algorithm for regional intersection traffic control is used to control traffic efficiently. And it uses Deep Learning to reduce the dimension of the huge traffic flow data generated by the road network. Simulation results suggest that the Deep Reinforcement Learning method significantly reduces the average delay in the traffic network when traffic conditions changed rapidly. The combination of Reinforcement Learning and Deep Learning enables the agent to interact with the environment of the intersection with prior knowledge. During the initial phases of training, the results of the Deep Q-Learning algorithm are not good, after a certain number of training episodes the efficiency of the model gets improved. The control of traffic with a Deep Reinforcement Learning algorithm is better than the Reinforcement Learning algorithm and traditional method of traffic control. Deep Reinforcement Learning is Learning from the existing dataset and applying the prior knowledge to the new dataset.

- (5) Li, Wei, Fan Zhou, Kaushik Roy Chowdhury, and Waleed Meleis. "QTCP: Adaptive congestion control with Reinforcement Learning."**

In this paper, Environment specifications are discussed for the Deep Q-Learning model. In Environmental Specification, network topologies are discussed. Many different types of network topologies can be chosen for the traffic model. In general, it is possible to study either isolated intersections or multiple intersections. Furthermore, it is possible to study either synthetic topologies or real-world topologies. Synthetic layouts allow for better control of the environment, but real-world intersections allow better adapt the Traffic Signal Control to the specific requirements for that intersection. Synthetic intersection layouts of bidirectional 4-way intersections are used in this paper. Researchers chose different numbers of lanes for the legs: between 1 and 4. The four-way intersection scenario of traffic is close to the real-world.

2.1 Critic on Literature Review

Based on a literature survey of existing research in the field of traffic signal control using Reinforcement Learning, several key observations can be made.

- **Observations**

Based on a literature survey of existing research in the field of traffic signal control using Reinforcement Learning several key observations can be made.

1. Reinforcement Learning has shown promise as a technique for optimizing traffic signal control. The ability of Reinforcement Learning algorithms to learn and adapt to changing traffic conditions makes them well-suited for this application.
2. The most commonly used Reinforcement Learning algorithms for traffic signal control are Q-Learning, SARSA, and Deep Q-Networks (DQNs). Each of these algorithms has its own strengths and weaknesses and may be more suitable for different types of traffic control scenarios.
3. Many studies have focused on using Reinforcement Learning to optimize traffic signal control at intersections. However, some studies have also explored the use of Reinforcement Learning for network-wide traffic control.
4. The performance of Reinforcement Learning algorithms for traffic signal control depends on a number of factors, including the specific algorithm used, the type of traffic scenario, and the performance metrics used to evaluate the effectiveness of the approach.
5. Several studies have compared the performance of Reinforcement Learning-based traffic signal control to traditional rule-based methods and have found that Reinforcement Learning can often achieve better results.
6. While Reinforcement Learning has shown promise for traffic signal control, there are still several challenges and limitations that need to be addressed. These include issues related to scalability, generalization, and interpretability of the learned policies.
7. Overall, the existing research suggests that Reinforcement Learning has the potential to be a powerful technique for optimizing traffic signal control, but further research is needed to address the remaining challenges and limitations.

- **Limitations from Literature Survey**

1. Q-Learning algorithm suffers from computational complexity.
2. Q-Learning algorithm has a problem with the curse of dimensionality
3. The traffic generation for variable rates is not used, it decreases the efficiency of the model, as it does not adapt to real-world scenarios.
4. Using real-world intersections in the model may result in better adaptation to the specific requirements of the intersection, but can lack control compared to synthetic layouts.
5. Limiting the movements of vehicles in the model to only straight traffic or forbidding turns may not accurately represent real-world traffic scenarios

2.2 Existing system of Traffic Signal Control System

The existing system of Traffic Signal Control involves using traffic lights to manage the flow of traffic at intersections. These traffic lights are controlled by a central system, which can be programmed to operate on a fixed schedule or adjusted manually by traffic controllers. The system typically includes a series of sensors, cameras, and other equipment to monitor traffic flow and adjust signal timings accordingly.

The most common type of Traffic Signal Control System is the fixed-time system, which operates on a pre-determined schedule. The timing of the signals is based on historical data and is designed to accommodate peak traffic periods. However, this approach can lead to inefficiencies during non-peak periods, as vehicles may be forced to wait at red lights even when there is no cross-traffic. More advanced Traffic Signal Control Systems, such as adaptive systems, use real-time data to adjust signal timings based on current traffic conditions. These systems can improve traffic flow and reduce congestion by allocating green lights based on the number of vehicles waiting at an intersection. In addition to

Traditional Traffic Signal Control Systems, there are also emerging technologies such as Connected and Autonomous Vehicles (CAVs) that have the potential to transform how traffic signals are controlled. CAVs can communicate with traffic signal systems to provide real-time data on their location, speed, and direction, allowing for more precise and efficient control of traffic signals.

Overall, the existing system of Traffic Signal Control is an important tool for managing traffic flow and ensuring safety at intersections, but it has limitations that can lead to inefficiencies and safety concerns. As technology continues to advance, new approaches to Traffic Signal Control are emerging that have the potential to address these limitations and improve the overall efficiency and safety of the transportation system.

CHAPTER 3

REQUIREMENTS ANALYSIS

3.1 Limitations of TLCS using Fixed timings:

1. **Inefficiency:** Fixed-time traffic signal systems often waste resources by allocating green lights to empty lanes, leading to unnecessary idling and increased pollution.
2. **Lack of Adaptability:** Most existing Traffic Signal Control Systems operate on a fixed schedule, which means that the timing of traffic signals is based on pre-determined patterns that do not take into account changes in traffic flow. This can result in unnecessary delays, increased congestion, and longer travel times for motorists.
3. **Limited Communication:** Traditional traffic signal systems have limited communication capabilities, which can make it difficult for transportation authorities to collect data and make informed decisions about traffic patterns and trends.

3.2 Software Requirements

Software Requirements for implementing the project Deep Reinforcement Approach for Smart Traffic Signal control is described as follows:

1. **Python:** version 3.10 2.
2. **SUMO:** Simulation of Urban Mobility (SUMO) is an open-source software package for simulating urban traffic systems. It is widely used in research and development to test and evaluate traffic management strategies and technologies.
SUMO simulates the movement of vehicles, pedestrians, and other traffic related entities in a virtual city. It models the interactions between different

modes of transportation and can simulate a wide range of scenarios, such as varying traffic volumes, signal timing, and road infrastructure.

Overall, SUMO provides a cost-effective and efficient way to evaluate traffic management strategies and technologies in a safe and controlled environment before implementing them in the real world.

- 3. TensorFlow:** TensorFlow is a powerful tool for building and training neural networks, which are a type of Machine Learning algorithm modeled after the structure of the human brain. Neural networks are widely used in various fields such as computer vision, natural language processing, and speech recognition.

In TensorFlow, neural networks are built using a high-level API called Keras, which provides an easy-to-use interface for constructing complex models. Keras allows users to build neural networks by stacking layers, where each layer performs a specific operation on the input data. For example, a convolutional layer can be used to extract features from an image, while a recurrent layer can be used to process sequences of data such as text or time series.

In addition to building and training neural networks, TensorFlow provides tools for optimizing and deploying them in production environments. For example, the TensorFlow Lite framework can be used to optimize models for deployment on mobile devices with limited computational resources, while the TensorFlow Serving framework can be used to serve models in production environments.

Overall, TensorFlow provides a powerful and flexible platform for building and training neural networks, making it a popular choice among researchers and developers in the Machine Learning community.

- 4. Visual Studio Code**

- 5. Operating System:** Windows 10,11

3.3 Hardware Requirements

Hardware Requirements for implementing the project Deep Reinforcement Approach for Smart Traffic Signal control is described as follows:

- 1. Processor:** i7
- 2. Hard Disk:** 10 GB
- 3. RAM:** 8GB and above
- 4. System type:** 64-bit operating system

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

4.1 Proposed System

The main objective of this project is to design a traffic light controller based on Computer Vision that can adapt to the current traffic situation. The traffic flow has no specific pattern that is followed, and the static signal timers pose a huge problem to the already critical problem of congestion.

Therefore, implementing a system which aims to reduce chances of such scenarios by automatically computing the optimal green signal time based on the current traffic at the signal will ensure that the direction with vehicles having more average waiting time has allotted a green signal first as compared to the direction with vehicles having less average waiting time.

This system can override the older system of hard coded lights which cause unwanted delays, reducing congestion and waiting time which will reduce the waiting time of vehicles and fuel consumption which in turn will help in controlling the air pollution. The reinforcement agent would be the Traffic Signal Controller, and the reward system would be based on optimizing traffic flow and reducing congestion.

The basic idea behind this approach is to use simulation and other data sources to gather information about traffic patterns, and then use that data to train the Traffic Signal Controller to make more efficient decisions about when to change the traffic lights. One of the key advantages of this approach is its adaptability. The system can learn from real-time data and adjust its decisions accordingly, which means it can respond to changes in traffic patterns and optimize traffic flow in real time. This is particularly useful in urban areas where traffic conditions can change rapidly and unpredictably.

Overall, Traffic Signal Control using Reinforcement Learning has the potential to significantly improve traffic flow and reduce congestion, which can have a positive impact on both the environment and quality of life in urban areas.

4.2 Architecture of Traffic Light Control System using RL

The Architecture of Traffic Signal control using Deep Reinforcement Learning is shown in Figure.

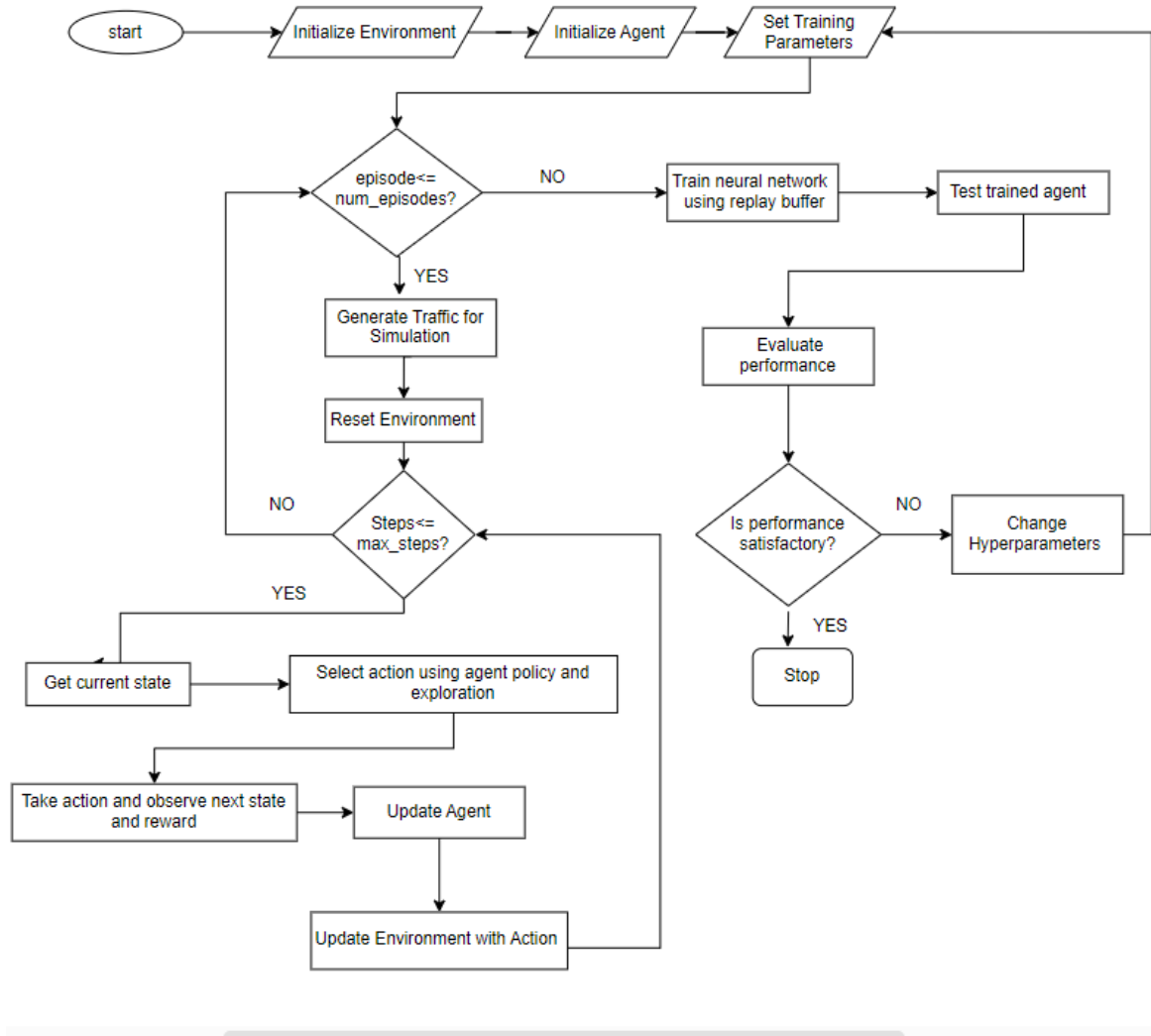


Fig:4.1 Architecture for Deep Reinforcement Learning

4.3 Description of modules in System Architecture

The architecture of Deep Reinforcement Learning (DRL) typically consists of several modules that work together to learn an optimal policy for an agent to perform a specific task. The following are some of the modules typically found in DRL architecture:

1. **Environment:** The environment is the external system or world in which the agent operates. It provides the agent with observations and rewards, and the agent takes actions in the environment to maximize the rewards.
2. **Agent:** The agent is the learning agent that interacts with the environment to learn an optimal policy. It receives observations from the environment and selects actions to perform based on the policy it has learned.
3. **State:** The state represents the current situation of the environment at a given time. It is a function of the history of observations and actions taken by the agent. discretization of oncoming lanes into presence cells, which identify the presence or absence of at least 1 vehicle inside them. There are 20 cells per arm. 10 of them are placed along the left most lane while the other 10 are placed in the other three lanes. 80 cells in the whole intersection. Action: The action is the output of the agent that represents the action to be taken in the current state of the environment. choice of the traffic light phase from 4 possible predetermined phases, described below. Every phase has a duration of 10 seconds. When the phase changes, a yellow phase of 4 seconds is activated.
 - **North-South Advance:** green for lanes in the north and south arm dedicated to turning right or going straight.
 - **North-South Left Advance:** green for lanes in the north and south arm dedicated to turning left.
 - **East-West Advance:** green for lanes in the east and west arm dedicated to turning right or going straight.
 - **East-West Left Advance:** green for lanes in the east and west arm dedicated to turning left.
4. **Reward:** The reward is a scalar feedback signal that the agent receives from the environment after performing an action. It indicates how well the agent performed the task, the change in cumulative waiting time between actions, where the waiting time of a car is the number of seconds spent with speed=0 since the spawn. When a car leaves an oncoming lane (i.e. crossed the intersection), its waiting time is no longer counted. Therefore, this translates to a positive reward for the agent.

5. **Policy:** The policy is the strategy or rule that the agent uses to select actions given the current state of the environment. It maps states to actions and is learned by the agent through interaction with the environment.
6. **Replay memory:** Replay memory is a key component in neural networks that are used in Reinforcement Learning, particularly in Deep Reinforcement Learning (DRL). It is a buffer that stores the experiences of an agent in the form of tuples of observations, actions, rewards, and next-state observations. During the training of a DRL agent, the replay memory is used to store and recall past experiences in order to improve the training efficiency and stability of the neural network. The memory buffer is initially filled with random experiences collected by the agent while it interacts with the environment. During the training process, the agent randomly samples a batch of experiences from the replay memory to update the weights of the neural network using backpropagation and gradient descent. Replay memory helps to break the correlation between consecutive samples in the training data, which can cause the training to become unstable and lead to poor performance. By randomly sampling experiences from the replay memory, the agent is able to learn from a diverse set of experiences and avoid getting stuck in local optima. Another advantage of replay memory is that it allows for experience replay, which is the process of reusing past experiences for multiple updates to the neural network. This can improve the learning efficiency of the network by allowing it to learn from rare events that may not occur frequently during the training process. Overall, replay memory is a key component in neural networks used in Reinforcement Learning that enables the agent to learn from a diverse set of experiences and improve the training efficiency and stability of the network.
7. **Value Function:** The value function is a function that estimates the expected cumulative reward that the agent will receive from a given state. It is used to evaluate the goodness of a state or an action and is learned through iterative updates.
8. **Neural Network:** The neural network is a function approximator that is used to approximate the policy, value function, or both. It is typically implemented

using Deep Neural Networks and is trained using backpropagation and gradient descent.

9. **Learning mechanism:** The agent makes use of the Q-learning equation $Q(s, a) = \text{reward} + \gamma \cdot \max_{a'} Q'(s', a')$ to update the action values and a Deep Neural Network to learn the state-action function. The neural network is fully connected with 80 neurons as input (the state), 5 hidden layers of 400 neurons each, and the output layers with 4 neurons representing the 4 possible actions. Also, an experience replay mechanism is implemented: the experience of the agent is stored in a memory and, at the end of each episode, multiple batches of randomized samples are extracted from the memory and used to train the neural network, once the action values have been updated with the Q-learning equation. These modules work together to enable the agent to learn an optimal policy for the task at hand through trial and error. The agent interacts with the environment, receives rewards, updates its policy and value function, and repeats this process until it learns an optimal policy.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 Description and implementation of the proposed model/Algorithm

- **Implementation**

The implementation of Traffic Signal Control using Reinforcement Learning can be divided into the following system modules.

1. Environment Creation
2. Traffic Generation
3. Model Training
4. System Testin
5. Model Evaluation
6. Optimization and Hyperparameter tuning.

The implementation steps are discussed as follows:

1. Environment Creation

For the agents to perform its actions, we set up the environment for the agent, which basically is the agent's world. The rules and dynamics of an environment cannot be influenced by the actions of the agent which interacts. We create the environment using the tool SUMO. Simulation of Urban Mobility (SUMO) is an open-source, highly portable, and continuous traffic simulation package designed to generate and handle large networks of traffic. SUMO allows the modeling of intermodal traffic systems including road vehicles, public transport, and pedestrians. It is purely microscopic, i.e., each vehicle is modeled explicitly having its own route and moves individually through the network. The 4-way intersection can be generated using SUMO, which gives the base foundation for the environment creation. Each lane consists of 4 incoming lanes and 4 outgoing lanes. Each lane consists of a unique ID. Discretization of oncoming lanes into cells is done so that the presence or absence of any vehicles on a lane will be known. These cells are not fixed for each lane and each cell will have a unique ID, with which a vehicle's presence on a lane will be inspected. Figure-5.1 represents the Environment Created using SUMO for the project Deep Reinforcement Learning approach for Smart Traffic Signal Control.

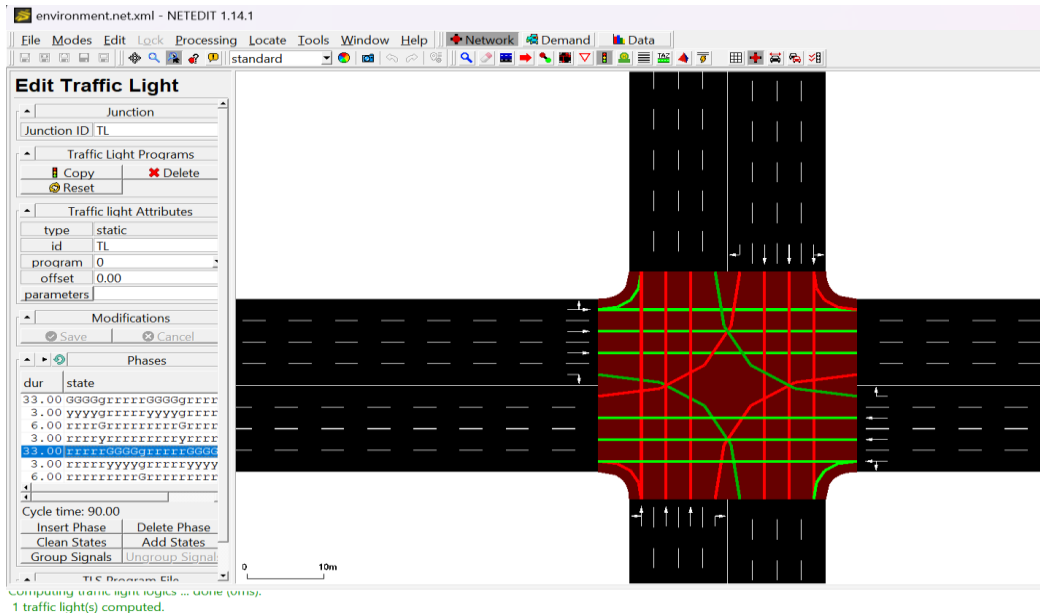


Fig:5.1 Environment creation using SUMO

2. Traffic Generation

Traffic generation is a critical aspect of simulation models that aim to predict traffic flows and evaluate traffic scenarios. In this simulation model, 100 cars are generated for the episode as part of the traffic generation process. To determine the timing of car arrivals, a Weibull distribution with shape 2 is used. This distribution results in a rapid increase in the arrival of cars until the mid-episode, followed by a slow decrease. This timing is critical to ensuring that the simulation the model reflects real-life traffic situations accurately. In terms of the direction of travel, 75% of the vehicles spawned will go straight, while 25% will turn left or right. This split ensures that the simulation model takes into account the various directions that vehicles may take while on the road. It is worth noting that every vehicle has an equal chance of being spawned at the beginning of every arm. This approach ensures that the traffic generation process is fair and unbiased. Further-more, since the cars are randomly generated for the episode, having two identical episodes regarding the vehicle's arrival layout is impossible. This variability in the traffic generation process ensures that the simulation model accurately reflects the

unpredictability of real-life traffic situations. Traffic is generated using the Weibull Distribution graph with slope 2. Weibull Distribution graph with slope 2 covers all cases of variable rates of traffic flow as shown in figure-5.1. It covers the slow and steady rate of traffic flow as well as the heavy traffic flow rate. Figure-5.2 represents weibull Distribution with slope 2 used for traffic generation for the project Deep Reinforcement Learning approach for Smart Traffic Signal Control. Figure-5.1 represents the Traffic Generated using SUMO for the project Deep Reinforcement Learning approach for Smart Traffic Signal Control.

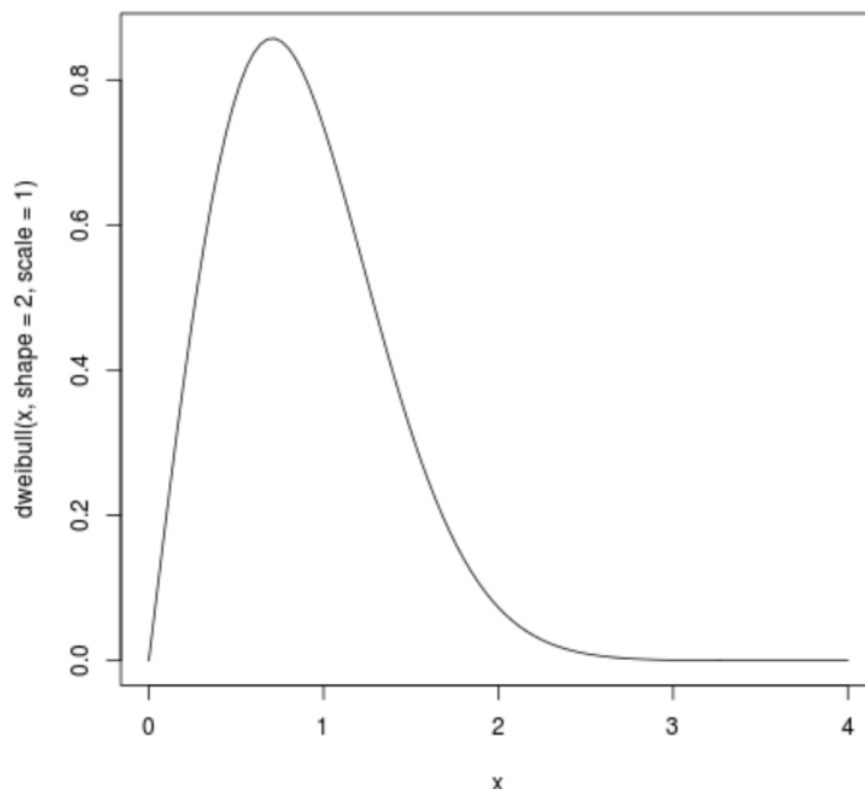


Fig:5.2 Weibull Distribution graph with a slope of 2

3. Model Training

- **Initialize traffic intersection:** This step sets up the traffic intersection within the simulation, including the location of traffic lanes, stop lights, and any other features necessary to simulate real-world traffic.
- **Initialize Reinforcement Learning algorithm with neural networks:** The RL algorithm is initialized with neural networks. These neural networks are used to predict the Q-value of a given state-action pair.

- **Initialize environment:** The environment is the simulation in which the RL algorithm will operate. It includes the traffic intersection as well as any other relevant features necessary for simulating traffic flow.

Input parameters for training the model:

[simulation]

gui = False

total episodes = 50

max steps = 2000

n cars generated = 100

green duration = 10

yellow duration = 4

[model]

num layers = 4

width layers = 400

batch size = 32

learning rate = 0.001

training epochs = 100

[memory]x

memory size min = 600

memory size max = 50000

[agent]

num states = 80

num actions = 4

gamma = 0.75

[dir]

models path name = models

sumocfg file name = sumo config .sumocfg

- **Repeat until simulation ends:** This step initiates a loop that will run until the simulation has ended. Within this loop, the RL algorithm will make decisions about when to let cars through the intersection.
- **Run simulation in environment:** At the beginning of each iteration of the loop, the simulation is run in the environment. This generates a new state, including the locations and velocities of all vehicles in the simulation.

- **Get current state from environment:** Once the simulation is run, the current state of the environment is obtained. This state includes information about the location and velocity of all vehicles, the state of the traffic lights, and any other relevant information about the environment.

Pseudo code for getting the current state from the environment:

```
def get state ( self ) :
    state = np. zeros ( self . num states )
    car list = traci . vehicle . getIDList ( )
    for car id in car list :
        lane pos = traci . vehicle . getLanePosition ( car id )
        lane id = traci . vehicle . getLaneID ( car id )
        lane pos = 750 - lane pos
        i f lane id == "W2TL 0" or lane id == "W2TL 1" :
            lane group = 0
        e l i f lane id == "W2TL 3" :
            lane group = 1
        e l i f lane id == "N2TL 0" or lane id == "N2TL 1 " :
            lane group = 2
        e l i f lane id == "N2TL 3 " :
            lane group = 3
        else :
            lane group = -1
        i f lane group >= 1 and lane group <= 7:
            car position = int ( str ( lane group ) + str ( lane cell ) )
            valid car = True
        e l i f lane group == 0:
            car position = lane cell
            valid car = True
        else :
            valid car = False
    return state
```

- **Choose action based on current state using Reinforcement**

Learning algorithm: Using the neural networks and the current state, the RL algorithm chooses the best action to take. The action might involve changing the traffic lights to let more cars through, for example.

pseudo code for choosing the action based on current state:

```
def choose action ( self , state , epsilon ) :
    if random.random ( ) < epsilon :
        return random. randint (0 , self . num actions - 1)
    else :
        return np.argmax( self . Model . predict one ( state ) )
```

- **Take action:** The action chosen by the RL algorithm is taken in the simulation. For example, the traffic lights might be changed to allow more cars through the intersection.

Pseudo code for choosing yellow phase:

```
def set yellow phase ( self , old action ) :
    yellow phase code = old action * 2 + \1
    traci . traffilight . setPhase ( " TL" , yellow phase)
```

Pseudo code for choosing Green phase:

```
def set green phase ( self , action number ) :
    if action number == 0:
        traci . t r a f f i c l i g h t . setPhase ( " TL" ,NS GREEN)
    e l i f action number == 1:
        traci . t r a f f i c l i g h t . setPhase ( " TL" ,NSL GREEN)
    e l i f action number == 2:
        traci . t r a f f i c l i g h t . setPhase ( " TL" , EWGREEN)
    e l i f action number == 3:
        traci . t r a f f i c l i g h t . setPhase ( " TL" , EWL GREEN)
```

- **Observe next state and reward:** After the action is taken, the simulation is run again to generate the next state. The reward is also calculated based on how well the RL algorithm performed. For example, if the algorithm increased the number of cars that were able to pass through the intersection, it would receive a positive reward.

Pseudo code for predicting the next state without batches of data:

```
def predict one ( self , state ) :  
    state = np. reshape ( state , [1 , self . input dim ] )  
    return self . model . predict ( state )
```

Pseudo code for predicting the next state with batches of data:

```
def predict batch ( self , states ) :  
    return self . model . predict ( states )
```

- **Store experience in experience replay buffer:** The experience replay buffer is a collection of previous state-action pairs and the resulting rewards. After each step of the RL algorithm, the current state-action pair and its resulting reward are stored in the experience replay buffer.

pseudo code for storing experiences into replay buffer:

```
def replay ( self ) :  
    batch = self . Memory. get samples ( self . Model . batch size )  
    i f len ( batch ) > 0:  
        states = np. array ( [ val [0] for val in batch ] )  
        next states = np. array ( [ val [3] for val in batch ] )  
        q s a = self . Model . predict batch ( states )  
        q s a d = self . Model . predict batch ( next states )  
        x = np. zeros ( ( len ( batch ) , self . num states ) )  
        y = np. zeros ( ( len ( batch ) , self . num actions ) )  
        for i , b in enumerate ( batch ) :  
            state , action , reward , = b[0] , b[1] , b[2]  
            current q = q s a [ i ]  
            current q [ action ] = reward + self . gamma  
            x [ i ] = state  
            y [ i ] = current q  
        overall accuracy = self . Model . train batch ( x , y )  
        self . accuracy .append( overall accuracy )
```

- **Train neural network with experience replay buffer:** After a certain number of steps, the neural network is trained on a batch of experiences randomly sampled from the experience replay buffer. The goal is to improve the accuracy of the Q-value predictions and make the RL algorithm more effective.

Pseudo code for building the model using Neural Networks:

```
def build_model ( self , num layers , width ) :
    inputs = keras . Input ( shape=( self . input dim , ) )
    x = layers .Dense( width , activation = ' relu ' ) ( inputs )
    for in range ( num layers ) :
        x = layers .Dense( width , activation = ' relu ' ) ( x )
    outputs = layers .Dense( self . output dim ,
        activation = ' linear ' ) ( x )
    model = keras .Model ( inputs=inputs , outputs=outputs ,
        name= 'my model ' )
    model. compile ( loss=losses . mean squared error , optimizer=
        Adam( lr=self . learning rate ) )
    return model
```

Pseudo code for training the batch and saving the model:

```
def train_batch ( self , states , q sa ) :
    self . accuracy history = [ ]
    batch size=self . batch size , epochs=1, verbose=0)
    return history . history [ ' accuracy ' ] [ 0 ]
def save_model ( self , path ) :
    self . model . save ( os . path . join ( path , ' trained model .h5
        ' ) )
```

- **Update Q-values:** The Q-values of the state-action pairs are updated based on the reward received in the previous step. This updates the neural networks and allows the RL algorithm to make better decisions in the future.

Adding samples to memory and retriving samples from memory:

```

class Memory:
def i n i t ( self , size max , size min ) :
self . samples = [ ]
self . size max = size max
self . size min = size min
def add sample ( self , sample ) :
self . samples .append( sample )
i f self . size now ( ) > self . size max :
self . samples .pop(0)
def get samples ( self , n ) :
i f self . size now ( ) < self . size min :
return [ ]
i f n > self . size now ( ) :
return random.sample ( self . samples , self .
size now ( ) )
else :
return random.sample ( self . samples , n)
def size now ( self ) :
return len ( self . samples )

```

- **End:** Once the simulation has ended, the loop is exited, and the RL algorithm is finished running. The resulting Q-values can be saved and used to control a real-world traffic intersection.

4. System Testing

- **Initialize traffic intersection:** This step sets up the traffic intersection within the simulation, including the location of traffic lanes, stop lights, and any other features necessary to simulate real-world traffic.
- **Initialize Reinforcement Learning algorithm with pre-trained neural networks:** The RL algorithm is initialized with pre-trained neural networks. These neural networks are used to predict the Q-value of a given state-action pair.

Load the best-trained model to test the environment:

```

def load my model ( self , path ) :
path = os . path . join ( path , ' trained model .h5 ' )

```

```

print ( path )
if os . path . i s f i l e ( model file path ) :
    loaded model = load model ( model file path )
    return loaded model
else :
    sys . exit ( " Model number not found " )

```

- **Initialize environment:** The environment is the simulation in which the RL algorithm will operate. It includes the traffic intersection as well as any other relevant features necessary for simulating traffic flow.

Initialize parameters for system testing:

```

[ simulation ]
gui = True
max steps = 4000
n cars generated = 1000
episode seed = 10000
yellow duration = 4
green duration = 10
[ agent ]
num states = 80
num actions = 4
[ dir ]
models path name = models
sumocfg file name = sumo config .sumocfg
model to test = 1

```

- **Repeat until simulation ends:** This step initiates a loop that will run until the simulation has ended. Within this loop, the RL algorithm will make decisions about when to let cars through the intersection.
- **Run simulation in environment:** At the beginning of each iteration of the loop, the simulation is run in the environment. This generates a new state, including the locations and velocities of all vehicles in the simulation.
- **Get current state from environment:** Once the simulation is run, the current state of the environment is obtained. This state includes information about the location and velocity of all vehicles, the state of

the traffic lights, and any other relevant information about the environment.

- **Region Proposal Network (RPN):** Once the simulation is run, the current state of the environment is obtained. This state includes information about the location and velocity of all vehicles, the state of the traffic lights, and any other relevant information about the environment.
- **Choose action based on current state using RL algorithm:** Using the pre-trained neural networks and the current state, the RL algorithm chooses the best action to take. The action might involve changing the traffic lights to let more cars through, for example.
- **Take action:** The action chosen by the RL algorithm is taken in the simulation. For example, the traffic lights might be changed to allow more cars through the intersection.
- **Observe next state and reward:** After the action is taken, the simulation is run again to generate the next state. The reward is also calculated based on how well the RL algorithm performed. For example, if the algorithm increased the number of cars that were able to pass through the intersection, it would receive a positive reward.
- **Update Q-values:** The Q-values of the state-action pairs are updated based on the reward received in the previous step. This updates the neural networks and allows the RL algorithm to make better decisions in the future.
- **End:** Once the simulation has ended, the loop is exited and the RL algorithm is finished running. The resulting Q-values can be saved and used to control a real-world traffic intersection.

Pseudo code for plotting visualization graphs:

```
class Visualization :  
    def init ( self , path , dpi ) :  
    def save data and plot ( self , data , filename , xlabel , ylabel ) :  
        min val = min( data )  
        max val = max( data )  
        plt.rcParams.update ( { ' font . size ' : 24 } )  
        plt . plot ( data )
```

```

plt . ylabel ( ylabel )
plt . xlabel ( xlabel )
plt . margins (0)
f i g = plt . gcf ( )
f i g . set size inches (20 , 11.25)
f i g . savefig ( os . path . join ( self . path , filename + '.png ' ) )
plt . close ( " all " )

with open( os . path . join ( ' data . txt ' ) , "w" ) as f i l e :

for value in data :
    f i l e . write("%s\n" % value)

```

5. Model Evaluation

Model evaluation is the process of assessing the performance of a Machine Learning model to determine its accuracy and reliability. The evaluation of a model is essential to determine whether the model is useful for making predictions and to identify areas where the model can be improved.

6. Optimization and Hyper Parameters Turing

The system's performance is evaluated regularly to identify any areas for improvement. The Reinforcement Learning model is retrained periodically to incorporate new traffic data and optimize signal timings further.

- **Reinforcement Learning**

Reinforcement Learning is a feedback-based Machine Learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or a penalty.

- **Deep Q-Learning**

In Deep Q-Learning, we use the Q-Learning idea and we use a neural network that takes a state and approximates the Q-values for each action

based on that state. We use a Deep Neural Network that takes the state as input and produces different Q-values for each action. We choose the action with the highest Q-value. All the past experience is stored by the user in memory. The next action is determined by the maximum output of the Q-network. Figure shows the Working of Deep Q-Learning algorithm with Agent, Environment and Policy.

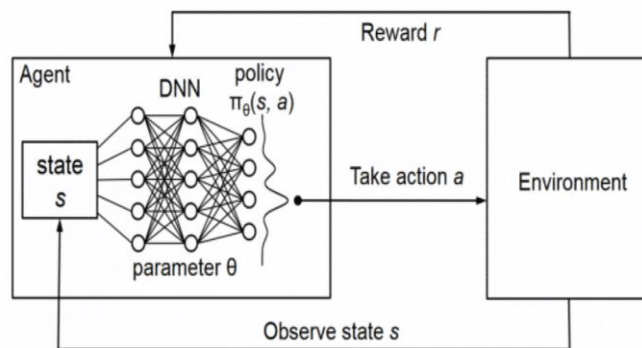


Fig:5.3 Diagram representing Deep Q-Learning

- **Q-Function**

We will use the Bellman equation as Q-function. According to the Bellman equation, the reward in a given action is equal to the reward from the current action combined with the expected rewards from future actions. The agent makes use of the Bellman equation for performing the best possible action on a given state. Bellman equation: $Q(s, a) = r(s, a) + \gamma \max_{a'} Q(s', a')$

State(s): Current state where the agent is in the Environment. Next State(s'): After taking action(a) at state(s) the agent reaches s' . Value(Q): Numeric representation of a state which helps the agent to find its path. $Q(s, a)$: means the value of the state s at an action a . Reward(R): treat which the agent gets after performing an action(a). $R(s)$: reward for being in the state s . $R(s, a)$: reward for being in the state and performing an action a . $R(s, a, s')$: reward for being in a state s , taking an action a and ending up in s' .

- **Q-Learning Algorithm**

1. Q-Learning Algorithm
2. Choose an action in the current states based on the current best Q-value.
3. Perform this action and observe the outcome (new state s').
4. Measure the reward R after this action.
5. Update Q with an updated formula that is called the Bellman Equation.

- **Environment**

The Environment is the agent's world in which it lives and interacts. The agent can interact with the environment by performing some action but cannot influence the rules or dynamics of the environment through those actions. A 4-way intersection is considered. Each arm has 4 incoming lanes and 4 outgoing lanes. Each arm has a specific length. Each arm is divided into a specific number of lanes.

- **States**

In Reinforcement Learning, states are the observations that the agent receives from the Environment. The state is a situation returned by the Environment after each Action taken by the Agent. Discretization of oncoming lanes into cells, which identify the presence or absence of at least 1 vehicle inside them. Each arm is divided into a certain number of cells. The number of cells in all lanes of an arm may or may not be the same. We keep track of vehicles by using a matrix of cells that each store a vehicle ID if one is present in that cell. Each vehicle is represented by an object that keeps track of the vehicle's ID, speed and position (cell indexes). An agent has access to the cell matrix and car information in the intersection for which it is responsible.

- **Action Space**

We use the traffic lights at intersections to control traffic flow. they represent the agents. The agents have the ability to show one of different colours,

green, or yellow, at different intervals, and these are considered the actions the agent will take. These actions will be assessed by the reward function for generating the reward for the agent. However, agents cannot just select any color they want at any time. Within an intersection, you cannot have two traffic lights open green if they are in conflict, so these traffic lights need to be synchronized to avoid any crashes. In order to avoid this, we coordinate things in time phases, where we have the intersection represented by agents that control these phases, when and for how long each phase should be opened or closed. Each road has four lanes. Left turns are allowed at any time, and here we consider only car traffic. a traffic light is assigned to each road, where all roads at an intersection are controlled by one agent.

5.2 Workflow involved in Traffic Signal Control using RL

The workflow of a system refers to the series of steps that are performed by the system to accomplish a particular goal or objective. The workflow of a system typically involves multiple phases, and each stage may have specific inputs, outputs, and dependencies. These phases are usually connected in a sequential or parallel fashion, with each stage passing its output to the next stage as input. The workflow of a system can be modeled using a flowchart or a diagram that shows the different stages of the process and the connections between them. This helps in understanding the system and identifying areas where improvements can be made to optimize the workflow and improve efficiency. Workflow of Deep Reinforcement Learning Approach for Smart Traffic Signal Control Signal is shown in Figure.

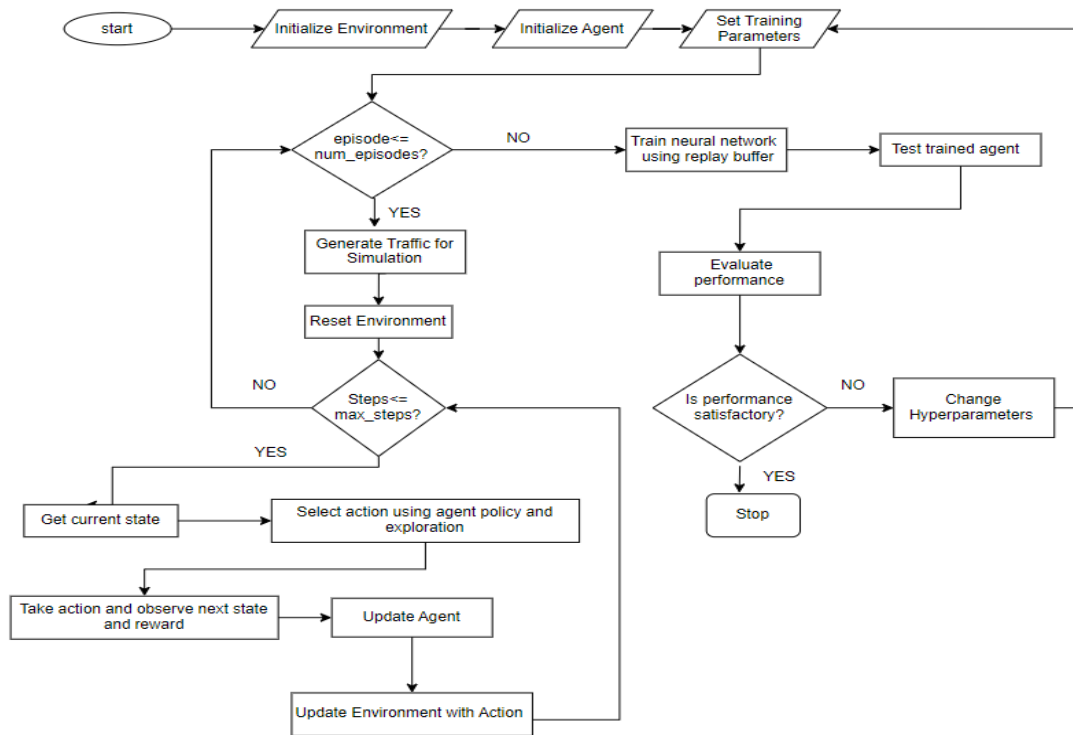


Fig:5.4 Workflow of Traffic Signal Control using Deep RL

5.3 Methodology Involved in Workflow:

1. **Start:** This is the starting point of the flowchart.
2. **Initialize environment:** In this step, the environment is initialized, which involves setting up the simulation and defining the traffic patterns.
3. **Initialize agent:** The agent is initialized, which involves setting up the Reinforcement Learning algorithm and defining the agent's policy.
4. **Set training parameters:** The training parameters are set, which includes defining the number of episodes and maximum number of steps per episode.
5. **Episode loop:** The episode loop starts and runs until the number of episodes is reached.

6. **Generate traffic for simulation:** Traffic is generated for the simulation based on the defined traffic patterns.
7. **Reset environment:** The environment is reset to its initial state.
8. **Step loop:** The step loop starts and runs until the maximum number of steps is reached.
9. **Get current state:** The agent gets the current state of the environment.
10. **Select action using agent policy and exploration:** The agent selects an action based on its policy, which can include exploration to try new actions.
11. **Take action and observe next state and reward:** The agent takes the selected action, and the environment returns the next state and a reward based on the action.
12. **Update agent:** The agent updates its policy based on the observed state, action, and reward.
13. **Update environment with action:** The environment updates its state based on the action taken by the agent.
14. **End of episode check:** If the end of the episode is reached, the loop is broken, and the next episode is started.
15. **Train neural network using replay buffer:** The agent's neural network is trained using the replay buffer, which is a data structure that stores past experiences to improve training efficiency.
16. **Test trained agent:** The trained agent is tested on a separate set of data to evaluate its performance.

17. Evaluate performance: The agent's performance is evaluated based on metrics such as traffic flow, congestion, and average waiting time.

18. Performance check: If the agent's performance is satisfactory, the training is stopped. If not, the hyperparameters are adjusted, and the training and testing are repeated.

In summary, this flowchart outlines the steps involved in training and testing a Reinforcement Learning agent for Traffic Signal Control. The agent learns by interacting with the environment, taking actions, and receiving rewards, and its performance is continuously evaluated and improved through iterative training and testing. The flowchart emphasizes the importance of setting appropriate training parameters and adjusting hyper parameters to optimize the agent's performance.

CHAPTER 6

RESULTS AND DISCUSSIONS

6.1 Simulation Results

The simulation results for Traffic Signal Control using Reinforcement Learning typically involve evaluating the performance of the Reinforcement Learning algorithm in optimizing the traffic signal timings. The results can be measured in terms of various metrics, including traffic flow, travel time, and fuel consumption. The figure-6.1 shows the simulation results for the Traffic Signal Control using Reinforcement Learning.

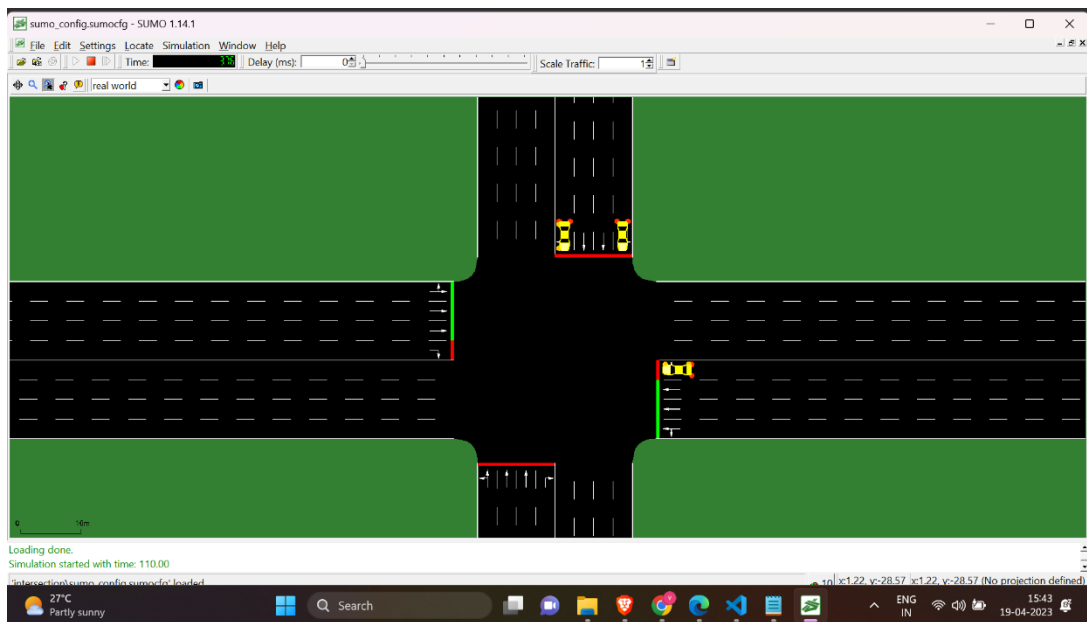


Fig:6.1 Simulation results for TLCS using RL

The traffic lights are under the control of a reinforcement agent. Reinforcement Learning is an effective approach for controlling traffic signal systems because it can adapt to changing traffic conditions and optimize the signal timings in real time. Figures 6.1 and 6.2 shows the simulation results for vehicles movements with the traffic lights being controlled by the reinforcement agent. The average waiting time of vehicles decreases when we control the traffic signal control using reinforcement learning.

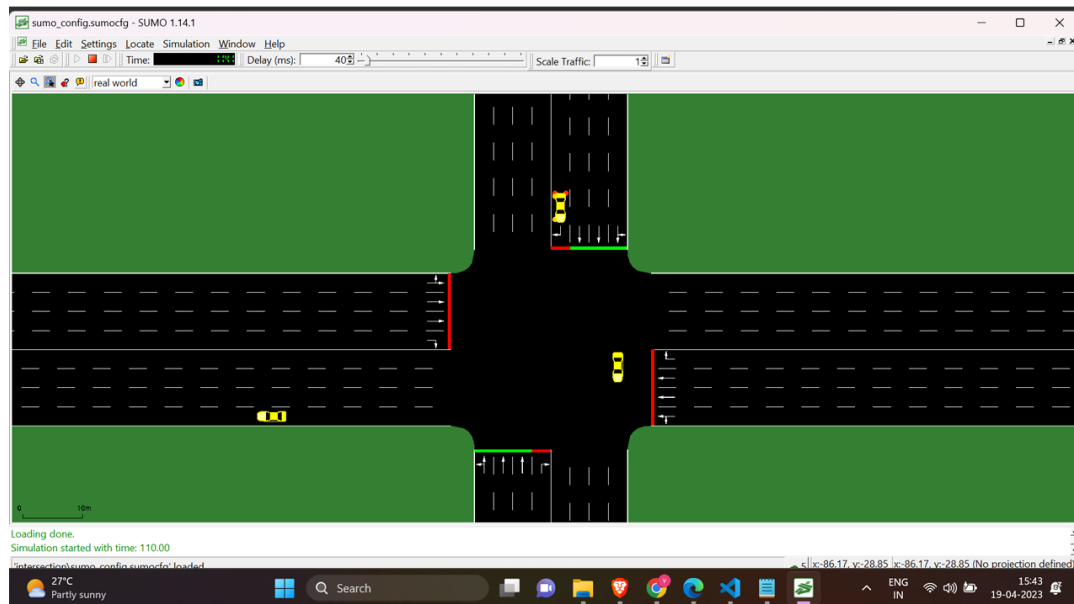


Fig:6.2 Simulation results for TLCS using RL

In the context of Traffic Signal Control using Reinforcement Learning, results may include measures such as the delay experienced, the average queue length, the reward gained by the agent, or the fuel consumption and emissions generated by vehicles. The following metrics are considered for comparing the performance of the existing system of Traffic Signal Control and the Traffic Signal Control using Reinforcement Learning:

- Episodes vs Delay plot
- Episodes vs Average Queue Length plot
- Episodes vs Cumulative Negative Reward plot

6.2 Comparing for existing system and Reinforcement model

• Episodes vs Delay plot

The plot in Figure-6.3 shows the relationship between the delay and the number of episodes in the context of Traffic Signal Control using Reinforcement Learning. The delay is plotted on the y-axis, which is a measure of how long vehicles have to wait at the intersection, while the number of episodes is plotted on the x-axis.

As the agent interacts with the environment over multiple episodes, it learns to control the traffic signal timings more effectively, which leads to a decrease in the delay experienced by vehicles. This learning is reflected in the plot as a downward trend in the delay over time.

The plot in the context of fixed-time Traffic Signal Control System may look like the one in Fig-6.3, with the y-axis representing the delay and the x axis representing the episodes. However, unlike the Fig-6.3 plot, the delay values in a fixed-time system will be fixed and not vary with the number of episodes. The episode vs waiting time plot will show that the waiting time for the Reinforcement Learning algorithm decreases more rapidly than that of the existing system. This suggests that the Reinforcement Learning algorithm is learning faster and performing better at optimizing traffic signal timing. The comparison between the two systems indicates that the Reinforcement Learning algorithm has the potential to significantly reduce traffic congestion and delays at intersections, resulting in a more efficient traffic flow.

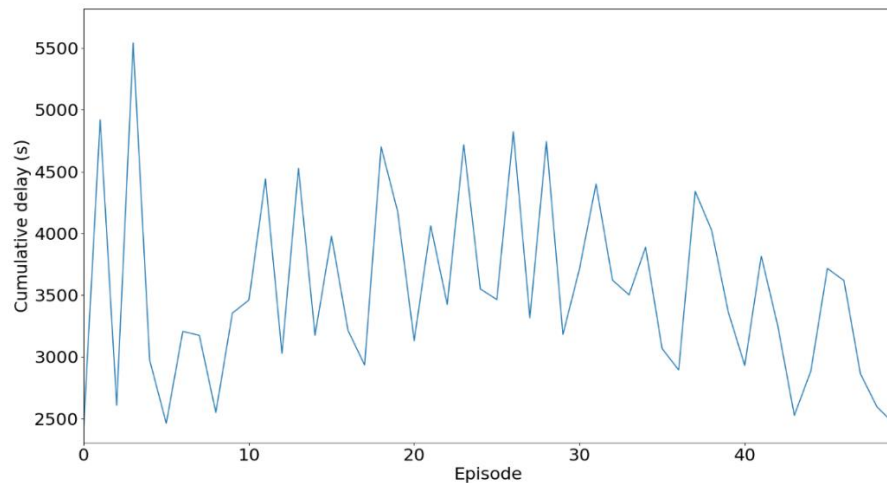


Fig:6.3 Delay vs Episodes plot for Existing System

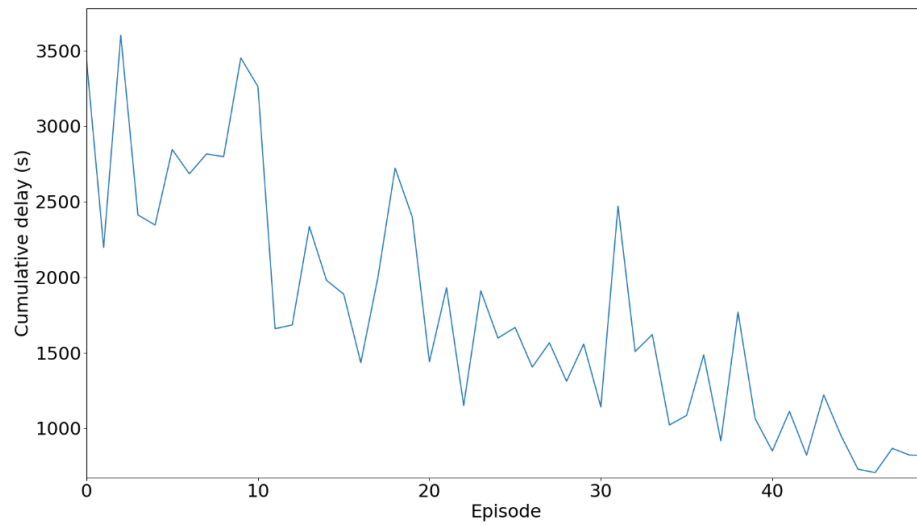


Fig:6.4 Delay vs Episodes plot for Reinforcement Model

- **Episodes vs Average Queue Length plot**

The average queue length refers to the average number of vehicles waiting in a queue at a given time period at an intersection. The queue length can be measured as the number of vehicles waiting at a particular point or along a stretch of road, and can be used as an indicator of traffic congestion and delay.

The plot in Figure-6.6 shows the relationship between the average queue length and the number of episodes in the context of Traffic Signal Control using Reinforcement Learning. The average queue length is plotted on the y-axis, while the number of episodes is plotted on the x-axis.

As the agent interacts with the environment over multiple episodes, it learns to control the traffic signal timings more effectively, which leads to a decrease in the average queue length of vehicles over the episodes. This learning is reflected in the plot as a downward trend in the delay over time.

The average queue length vs episodes plot in the context of a fixed-time Traffic Signal Control System may look similar to the one in Fig-6.6, with the y-axis representing the average queue length and the x-axis representing the

episodes. However, unlike the Fig-6.6 plot, the average queue length values in a fixed-time system will be fixed and not vary with the number of episodes. The Episode vs Average Queue Length plot will show that the Average Queue Length for the Reinforcement Learning algorithm decreases more rapidly than that of the existing system. This suggests that the Reinforcement Learning algorithm is learning faster and performing better at optimizing traffic signal timing. The comparison between the two systems indicates that the Reinforcement Learning algorithm has the potential to significantly reduce traffic congestion and delays at intersections, resulting in a more efficient traffic flow.

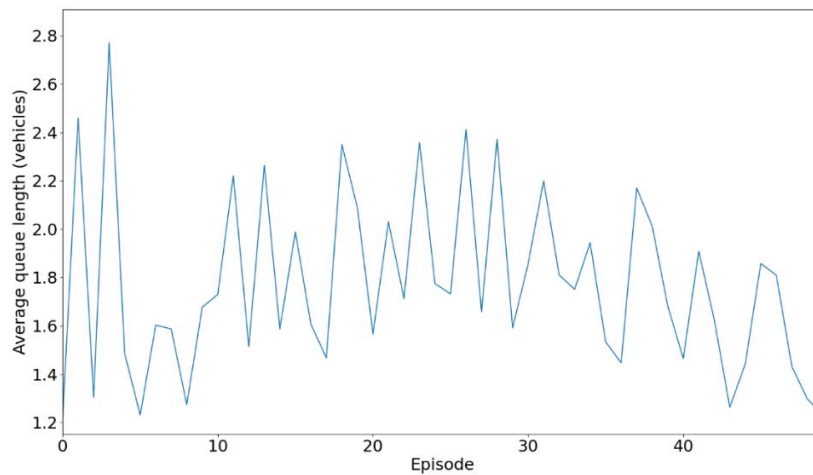


Fig:6.5 Queue Length vs Episodes plot for existing System

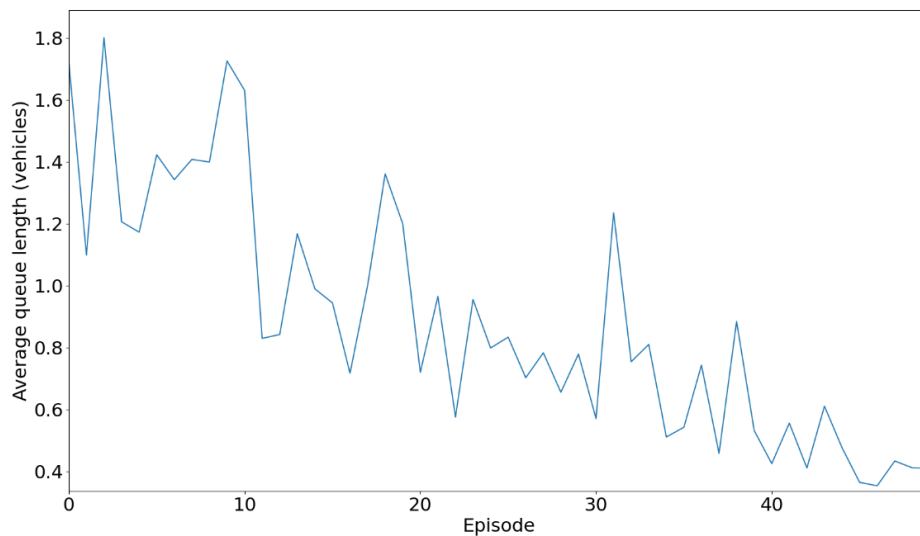


Fig:6.6 Queue Length vs Episodes plot for Reinforcement Model

- **Comparing Episodes vs Cumulative Negative Reward plot for existing system and Reinforcement model**

The Reward is the difference in waiting time for each vehicle between the current and previous time steps. The plot in Figure-6.8 shows the relationship between the cumulative reward and the number of episodes in the context of Traffic Signal Control using Reinforcement Learning. The cumulative is plotted on the y-axis, while the number of episodes is plotted on the x-axis. As the agent interacts with the environment over multiple episodes, it learns to control the traffic signal timings more effectively, which leads to an increase in the cumulative reward gained by an agent. This learning is reflected in the plot as a downward trend in the delay over time.

The episodes vs cumulative reward plot in the context of a fixed-time Traffic Signal Control System may look similar to the one in Fig-6.7, with the y-axis representing the cumulative reward and the x-axis representing the episodes. However, unlike the Fig-6.7 plot, the reward values in a fixed-time system will be fixed and not vary with the number of episodes.

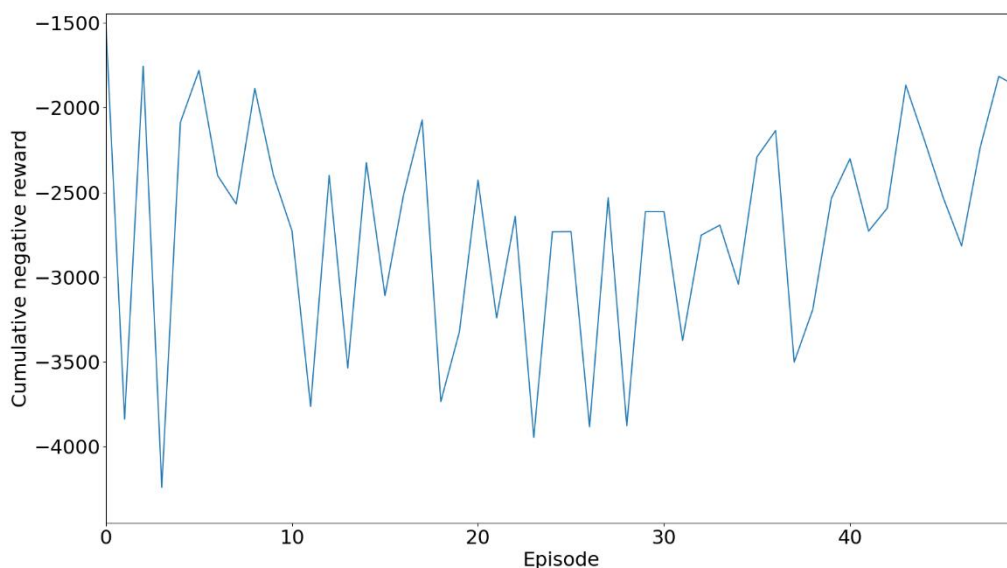


Fig: 6.7 Reward vs Episodes plot for existing System

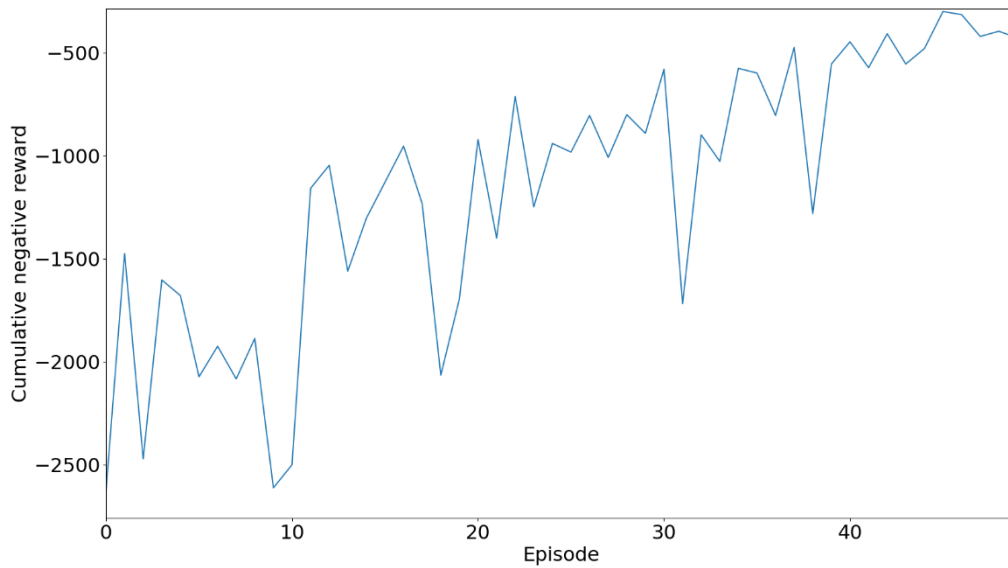


Fig:6.8 Reward vs Episodes plot for Reinforcement Model

The episode vs Cumulative Negative Reward plot will show that the Cumulative Negative Reward for the Reinforcement Learning algorithm decreases more rapidly than that of the existing system. This suggests that the Reinforcement Learning algorithm is learning faster and performing better at optimizing traffic signal timing. The comparison between the two systems indicates that the Reinforcement Learning algorithm has the potential to significantly reduce traffic congestion and delays at intersections, resulting in a more efficient traffic flow.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

The use of Reinforcement Learning in Traffic Signal Control has shown promising results in improving traffic flow and reducing congestion. The application of Deep Reinforcement Learning techniques has led to significant improvements in the optimization of traffic signal timing plans, with better performance compared to conventional approaches. By leveraging the power of Machine Learning algorithms, Traffic Signal Control Systems can become more adaptive and responsive to changing traffic conditions, resulting in a more efficient and sustainable transportation network. The results demonstrate that the Deep Reinforcement Learning approach outperforms traditional signal control methods in terms of reducing traffic delay and improving intersection efficiency.

Different from the traffic flow-based control strategy, the Deep Reinforcement Learning algorithm simultaneously learns the dynamics of traffic systems and the optimal control plan by implicitly modeling the control actions and the change of system states. Applying the Deep Reinforcement Learning method developed and shown that it notably outperformed conventional approaches in finding a better signal timing plan.

7.2 Future Scope

- **Multi Agent TLCS:** The Reinforcement Learning algorithm can be implemented for multiple junctions; such that multiple Reinforcement agents interacts with each other for better Traffic Signal Control.
- **Emergency Vehicles:** Traffic clearance for the emergency vehicles, allows them to bypass traffic and reach their destination faster.
- **Customized solutions for specific traffic scenarios:** Reinforcement Learning algorithms can be customized for specific traffic scenarios, such as rush hour traffic or special events. This customization can help optimize traffic flow in these scenarios and reduce congestion.

- **Use of real-time data:** The use of real-time data can help improve the accuracy and effectiveness of Reinforcement Learning algorithms. This data can include traffic volume, weather conditions, and incidents such as accidents or roads.

REFERENCES

- [1] A. L. Bazzan, "Opportunities for multiagent systems and multiagent reinforcement learning in traffic control," *Autonomous Agents and Multi-Agent Systems*, vol. 18, pp. 342–375, 2009.
- [2] A. Salaymeh, L. Schwiebert, and S. Remias, "Multi-agent reinforcement learning for optimizing traffic signal timing," in *CS & IT Conference Proceedings*, vol. 11, *CS & IT Conference Proceedings*, 2021.
- [3] C. Li, F. Yan, Y. Zhou, J. Wu, and X. Wang, "A regional traffic signal control strategy with deep reinforcement learning," in *2018 37th Chinese Control Conference (CCC)*, pp. 7690–7695, IEEE, 2018.
- [4] D. Krajzewicz, "Traffic simulation with sumo—simulation of urban mobility," *Fundamentals of traffic simulation*, pp. 269–293, 2010.
- [5] D. Leitner, P. Meleby, and L. Miao, "Recent advances in traffic signal performance evaluation," *Journal of traffic and transportation engineering (English edition)*, 2022.
- [6] L. Li, Y. Lv, and F.-Y. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 3, no. 3, pp. 247– 254, 2016.
- [7] M. Eom and B.-I. Kim, "The traffic signal control problem for intersections: a review," *European transport research review*, vol. 12, pp. 1–20, 2020.
- [8] N. Bhave, A. Dhagavkar, K. Dhande, M. Bana, and J. Joshi, "Smart signal–adaptive traffic signal control using reinforcement learning and object detection," in *2019 third international conference on I-SMAC (IoT in social, mobile, analytics and cloud) (I-SMAC)*, pp. 624–628, IEEE, 2019.

- [9] N. S. Jadhao and A. S. Jadhao, "Traffic signal control using reinforcement learning," in 2014 fourth international conference on communication systems and network technologies, pp. 1130–1135, IEEE, 2014.
- [10] P. A. Rao, B. N. Kumar, S. Cadabam, and T. Praveena, "Distributed deep reinforcement learning using tensorflow," in 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC), pp. 171–174, IEEE, 2017.
- [11] S. A. Celtek, A. Durdu, and M. E. M. Ali, "Evaluating action durations for adaptive traffic signal control based on deep q-learning," *International journal of intelligent transportation systems research*, vol. 19, no. 3, pp. 557–571, 2021.
- [12] S. Touhbi, M. A. Babram, T. Nguyen-Huu, N. Marilleau, M. L. Hbid, C. Cambier, and S. Stinckwich, "Adaptive traffic signal control: Exploring reward definition for reinforcement learning," *Procedia Computer Science*, vol. 109, pp. 513–520, 2017.
- [13] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "Qtcp: Adaptive congestion control with reinforcement learning," *IEEE Transactions on Network Science and Engineering*, vol. 6, no. 3, pp. 445–458, 2018.
- [14] W. Genders, *Deep reinforcement learning adaptive traffic signal control*. PhD thesis, 2018.
- [15] W. Genders and S. Razavi, "Evaluating reinforcement learning state representations for adaptive traffic signal control," *Procedia computer science*, vol. 130, pp. 26–33, 2018.

APPENDIX

A. Source Code

- **Traffic Generator**

```
import numpy as np
import math

class TrafficGenerator:
    def __init__(self, max_steps, n_cars_generated):
        self._n_cars_generated = n_cars_generated # how many cars per episode
        self._max_steps = max_steps

    def generate_routefile(self, seed):
        """
        Generation of the route of every car for one episode
        """
        np.random.seed(seed) # make tests reproducible

        # the generation of cars is distributed according to a weibull distribution
        timings = np.random.weibull(2, self._n_cars_generated)
        timings = np.sort(timings)

        # reshape the distribution to fit the interval 0:max_steps
        car_gen_steps = []
        min_old = math.floor(timings[1])
        max_old = math.ceil(timings[-1])
        min_new = 0
        max_new = self._max_steps
        for value in timings:
            car_gen_steps = np.append(car_gen_steps, ((max_new - min_new) /
(max_old - min_old)) * (value - max_old) + max_new)
```

```

car_gen_steps = np rint(car_gen_steps) # round every value to int ->
effective steps when a car will be generated

# produce the file for cars generation, one car per line
with open("intersection/episode_routes.rou.xml", "w") as routes:
    print("""<routes>
        <vType accel="1.0" decel="4.5" id="standard_car" length="5.0"
minGap="2.5" maxSpeed="25" sigma="0.5" />

        <route id="W_N" edges="W2TL TL2N"/>
        <route id="W_E" edges="W2TL TL2E"/>
        <route id="W_S" edges="W2TL TL2S"/>
        <route id="N_W" edges="N2TL TL2W"/>
        <route id="N_E" edges="N2TL TL2E"/>
        <route id="N_S" edges="N2TL TL2S"/>
        <route id="E_W" edges="E2TL TL2W"/>
        <route id="E_N" edges="E2TL TL2N"/>
        <route id="E_S" edges="E2TL TL2S"/>
        <route id="S_W" edges="S2TL TL2W"/>
        <route id="S_N" edges="S2TL TL2N"/>
        <route id="S_E" edges="S2TL TL2E"/>""", file=routes)

    for car_counter, step in enumerate(car_gen_steps):
        straight_or_turn = np.random.uniform()
        if straight_or_turn < 0.75: # choose direction: straight or turn - 75% of
times the car goes straight
            route_straight = np.random.randint(1, 5) # choose a random source &
destination
            if route_straight == 1:
                print('    <vehicle id="W_E_%i" type="standard_car" route="W_E"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)
            elif route_straight == 2:

```

```

        print('    <vehicle id="E_W_%i" type="standard_car" route="E_W"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

    elif route_straight == 3:
        print('    <vehicle id="N_S_%i" type="standard_car" route="N_S"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

    else:
        print('    <vehicle id="S_N_%i" type="standard_car" route="S_N"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

    else: # car that turn -25% of the time the car turns
        route_turn = np.random.randint(1, 9) # choose random source source
& destination

        if route_turn == 1:
            print('    <vehicle id="W_N_%i" type="standard_car" route="W_N"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

        elif route_turn == 2:
            print('    <vehicle id="W_S_%i" type="standard_car" route="W_S"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

        elif route_turn == 3:
            print('    <vehicle id="N_W_%i" type="standard_car" route="N_W"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

        elif route_turn == 4:
            print('    <vehicle id="N_E_%i" type="standard_car" route="N_E"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

        elif route_turn == 5:
            print('    <vehicle id="E_N_%i" type="standard_car" route="E_N"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

```

```

        elif route_turn == 6:
            print('    <vehicle id="E_S_%i" type="standard_car" route="E_S"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

        elif route_turn == 7:
            print('    <vehicle id="S_W_%i" type="standard_car" route="S_W"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

        elif route_turn == 8:
            print('    <vehicle id="S_E_%i" type="standard_car" route="S_E"
depart="%s" departLane="random" departSpeed="10" />' % (car_counter, step),
file=routes)

    print("</routes>", file=routes)

```

- **Model**

```

import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2' # kill warning about tensorflow
import tensorflow as tf
import numpy as np
import sys

```

```

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import losses
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import load_model

```

```

class TrainModel:
    def __init__(self, num_layers, width, batch_size, learning_rate, input_dim,
output_dim):
        self._input_dim = input_dim

```

```

self._output_dim = output_dim
self._batch_size = batch_size
self._learning_rate = learning_rate
self._model = self._build_model(num_layers, width)

```

```

def _build_model(self, num_layers, width):
    """
    Build and compile a fully connected deep neural network
    """
    inputs = keras.Input(shape=(self._input_dim,))
    x = layers.Dense(width, activation='relu')(inputs)
    for _ in range(num_layers):
        x = layers.Dense(width, activation='relu')(x)
    outputs = layers.Dense(self._output_dim, activation='linear')(x)

    model = keras.Model(inputs=inputs, outputs=outputs, name='my_model')
    model.compile(loss=losses.mean_squared_error,
optimizer=Adam(lr=self._learning_rate))
    return model

```

```

def predict_one(self, state):
    """
    Predict the action values from a single state
    """
    state = np.reshape(state, [1, self._input_dim])
    return self._model.predict(state)

```

```

def predict_batch(self, states):
    """
    Predict the action values from a batch of states
    """

```

```
return self._model.predict(states)
```

```
def train_batch(self, states, q_sa):
```

```
    """
```

```
    Train the nn using the updated q-values
```

```
    """
```

```
    self._model.fit(states, q_sa, epochs=1, verbose=0)
```

```
def save_model(self, path):
```

```
    """
```

```
    Save the current model in the folder as h5 file and a model architecture  
summary as png
```

```
    """
```

```
    self._model.save(os.path.join(path, 'trained_model.h5'))
```

```
    plot_model(self._model, to_file=os.path.join(path, 'model_structure.png'),  
show_shapes=True, show_layer_names=True)
```

```
@property
```

```
def input_dim(self):
```

```
    return self._input_dim
```

```
@property
```

```
def output_dim(self):
```

```
    return self._output_dim
```

```
@property
```

```
def batch_size(self):
```

```
    return self._batch_size
```

```

class TestModel:
    def __init__(self, input_dim, model_path):
        self._input_dim = input_dim
        self._model = self._load_my_model(model_path)

    def _load_my_model(self, model_folder_path):
        """
        Load the model stored in the folder specified by the model number, if it exists
        """
        model_file_path = os.path.join(model_folder_path, 'trained_model.h5')

        if os.path.isfile(model_file_path):
            loaded_model = load_model(model_file_path)
            return loaded_model
        else:
            sys.exit("Model number not found")

    def predict_one(self, state):
        """
        Predict the action values from a single state
        """
        state = np.reshape(state, [1, self._input_dim])
        return self._model.predict(state)

    @property
    def input_dim(self):
        return self._input_dim

```

Deep Reinforcement Learning Approach for Smart Traffic Signal Control System

Karthik Reddy Polamreddy

Department of Computer Science
and Engineering

Sathyabama Institute of Science
and Technology

Jeppiar Nagar, Rajiv Gandhi Salai,
Chennai, Tamil Nadu 600119

karthikreddykarthi007@gmail.com

Sandhya DK

Department of Computer Science
and Engineering

Sathyabama Institute of Science
and Technology

Jeppiar Nagar, Rajiv Gandhi Salai,
Chennai, Tamil Nadu 600119

sandhyadk006@gmail.com

Dr. Sujihelen L

Department of Computer Science
and Engineering

Sathyabama Institute of Science
and Technology

Jeppiar Nagar, Rajiv Gandhi Salai,
Chennai, Tamil Nadu 600119

sujihelen.cse@sathyabama.ac.in

Abstract: Traffic congestion is becoming a critical issue with the increasing usage of automobiles. The rapid increase in automobiles has naturally led to traffic congestion all over the world, forcing people to sit idle in their vehicles, consuming more fuel, and increase of air pollution. Traffic light control policies are not optimized, leading vehicles to wait pointlessly for non-existent traffic to pass on the crossing road. The current traffic signal systems follow fixed timing patterns that are not adjusted depending up on the volume of vehicles present on the traffic junction. Creating an effective transportation system is of utmost importance in order to conserve time and reduce the impact on the environment by minimizing carbon emissions. Traffic light control policies can be greatly improved by implementing Deep learning (DL) and Reinforcement Learning (RL) concepts. Reinforcement Learning has become a popular method for controlling traffic signals in recent times as it can learn the best control policies through practical experience. By using RL, this research work aims to enhance traffic control efficiency and minimize issues such as delays, traffic congestion, fuel consumption, and pollution. The goal of this research work is to reduce the cumulative waiting time of vehicles at the junctions. Waiting time, Reward, Queue

Length are considered as performance metrics. Moreover, the proposed approach is designed to be cost-effective and can be implemented in real-time systems.

Keywords: Deep Learning, Traffic Light Control Policies, Reinforcement Learning, Waiting time, Reward, Queue Length.

1. INTRODUCTION

The growth of urban areas in the modern era has resulted in high traffic volumes, particularly in developing countries where the road infrastructure is inadequate. Most traffic junctions across the globe employ fixed signal timings, which use a Round-Robin method to cycle through different signal configurations at regular intervals. This approach, while simple to execute, can exacerbate traffic congestion since it does not consider the traffic that exists at the junction. In other words, the system may activate a green signal for an intersection even if no vehicles are present, which can worsen traffic congestion. Researchers like, Li Li et al. (2016) have looked into using Reinforcement Learning techniques for Traffic Light Control System (TLCS) to get over this constraint. A Machine Learning technique called Reinforcement learning includes teaching an agent to select the best course of action

depending on input from its surroundings. When controlling the traffic signals, the agent must make decisions on varying the traffic lights according to traffic scenarios, such as the volume and density of cars. By training the agent to optimize traffic flow, traffic congestion can be reduced, leading to improved safety, reduced fuel consumption, and reduced travel time for motorists. This approach has shown promise in simulations and small-scale experiments, and ongoing research is exploring its potential in larger-scale, real-world applications.

The Reinforcement Learning techniques for traffic light control system has displayed potential both in simulations and real-world tests. In simulations, virtual environments is created that replicates real-world traffic conditions, allowing them to test different control strategies without risking the safety of drivers and pedestrians explained in N.S. Jadhao et al. (2014).

Simulations allow researchers to model complex traffic scenarios, such as intersections with multiple lanes and traffic volumes and patterns can be varied to test the robustness of the Reinforcement Learning algorithm under different conditions, in D. Krajzewicz et al. (2010) suggested that simulations are better for experimentations. In these simulations, the Reinforcement Learning agent receives feedback based on the action performed by the agent. For example, the agent receives positive reward for reducing congestion or a negative reward for increasing wait times. Over time, the agent learns to make better decisions by adjusting its policy to maximize the expected reward. Simulation studies have been conducted to compare the effectiveness of conventional fixed-time based traffic light control methods with the DRL-based TLCS. The results have shown that Reinforcement Learning can lead to significant improvements in traffic flow and reduced

travel time for motorists, particularly during peak hours when congestion is highest.

The topic of traffic signal control has considerable potential to be revolutionized by Neural Networks and Deep Learning approaches, elucidated in W. Genders et al. (2018). These methods, when paired with Reinforcement Learning, enable traffic signals to dynamically adjust in real-time to changing traffic circumstances, resulting in improved traffic flow, less congestion, and increased safety. Neural Network is used to find patterns in data. They are made up of interconnected nodes or neurons that can learn from experience and are modelled after the composition and operation of the human brain. A subclass of Neural Networks known as Deep Learning employs numerous layers to extract progressively complicated features from the input data. Neural Networks and Deep Learning are used to analyse data related to real-time traffic, such as vehicle counts and speeds, and to forecast traffic situations in the context of traffic light control. When deciding when to switch traffic lights, the Reinforcement Learning agent can utilize these predictions as guidance.

Traffic signal control systems can make more precise forecasts and respond to shifting traffic circumstances by processing traffic data using Deep Neural Networks. This can significantly enhance traffic flow, resulting in shorter wait times and shorter travel times for drivers. The application of neural networks and deep learning for traffic signal control is a fascinating area of research. Currently, there are ongoing investigations into the optimal design of these systems for practical implementation. These methods could revolutionize traffic control and make cities safer and more effective with further development. To improve the TLCS policies, we are developing a smart TLCS that combines

DRL and Simulation. In this research, we suggest a fully functional system that can take the role of existing systems at traffic crossings. This project will involve implementing Deep Reinforcement Learning algorithm and evaluating its performance on a simulated traffic network, and comparing its performance to that of traditional fixed-time control methods. Ultimately, the goal of the project is to develop more efficient and sustainable traffic light control systems that can control traffic flow more efficiently and reduce environmental impacts. The suggested system requires far less hardware and computing resources than the systems now in use.

The remainder of the paper is organized using the format below. Literature Survey for TLCS is demonstrated in Section 2. The workflow of proposed system and Phases involved in workflow is elucidated in Section 3. Results and Discussions are demonstrated in Section 4. Conclusion of the research work and suggestions for enhancing the suggested method is illustrated in Section 5.

2. Related Work

A significant issue in transportation engineering is TLCS. For many years, traditional traffic light control techniques like fixed-time-based TLCS in use. The potential of these techniques to react to varying traffic conditions and deliver top performance is constrained. Recently, the idea of using Reinforcement learning (RL) for traffic light regulation has gained more and more traction. Machine learning's RL subfield focuses on making decisions in the face of uncertainty. Reinforcement agent interacts with the environment and gets feedback from the agent, either it can be a positive reward or a negative reward based on the action performed by the agent in the

environment. Agent tries to choose the best possible actions.

Many studies have looked into the application of RL approach for TLCS. For instance, Li et al. (2019) suggested a Deep RL technique for enhancing traffic signal regulation at a single intersection. Comparing their strategy to conventional ones, the traffic flow improved significantly. For the purpose of optimizing control of traffic signal in a network of intersections, Zhang et al. (2020) suggested a distributed RL technique. In terms of trip time and delay, their solution performed better than the alternatives.

Multi-agent RL for TLCS is also a compelling approach worth exploring. Multi-agent RL enables agents to collaborate and learn from one another in order to accomplish a shared objective. A multi-agent RL approach for TLCS in a vast urban network was put forth by Wu et al. in 2021. In terms of trip time and traffic flow, their strategy outperformed more established techniques.

Simulation of Urban Mobility (SUMO) is widely used for creating traffic junction, which is the environment in the project. It enables researchers to recreate intricate traffic situations and assess how well different traffic management techniques operate. SUMO has been utilized in multiple studies for a variety of purposes in the field of transportation research. To simulate how linked and automated vehicles will affect how traffic moves along a highway network, Li et al. (2020) employed SUMO, for instance. Their findings demonstrated that automated and linked vehicles can dramatically lessen traffic congestion and enhance traffic flow. SUMO was utilized by Gao et al. (2018) to assess the effectiveness of the Continuous Flow Intersection (CFI), a unique intersection design. Their findings

demonstrated that, in comparison to conventional intersection designs, the CFI can shorten delays and enhance traffic flow. The optimization of traffic light control is another area of research that has utilized SUMO. For instance, Zhang et al. (2019) created a method based on simulation for improving TLCS using SUMO. To determine the ideal timings for traffic signals in a complicated network of intersections, their method used Reinforcement learning.

Studies that assess the effects of various transport policies and initiatives have also employed SUMO. To assess the effect of a congestion pricing policy on traffic flow in a downtown area, for instance, Wang et al. (2020) employed SUMO. Their findings demonstrated that the congestion pricing scheme can dramatically shorten travel times and reduce traffic. For transportation researchers, SUMO is a potent tool that enables the simulation of intricate traffic scenarios and the evaluation of various transportation plans. SUMO has been employed in several studies for a variety of purposes, including the assessment of novel intersection designs, the enhancement of TLCS, and the assessment of transportation policies and interventions.

3. Proposed Work

The proposed system of traffic light control system using DRL aims to optimize traffic flow in a simulated traffic network by training a Reinforcement agent with an optimal policy for selecting traffic signal phases. The system employs a deep Reinforcement learning algorithm, which involves an agent performs action on the environment to maximize the cumulative reward of the Reinforcement agent '. In this case, the reward signal represents the effectiveness of the agent's selected traffic

signal phase in reducing congestion and minimizing delays for vehicles in the network.

The proposed system of traffic signal control using Reinforcement learning has the potential to improve traffic flow and reduce congestion and delays in a simulated traffic network. The system can learn from experience and adapt to changes in traffic patterns, making it a promising approach to traffic signal control. The Figure 1 shows the workflow of the proposed system.

3.1. Components Involved in Proposed Work

1. **Environment:** The environment is the external world in which the agent makes decisions and operates. The agent makes decisions in the environment to maximize the benefits after receiving observations and incentives from it.
2. **Agent:** The Reinforcement agent performs actions on the environment with an optimal policy. It receives observations from the environment and selects actions to perform based on the policy it has learned.

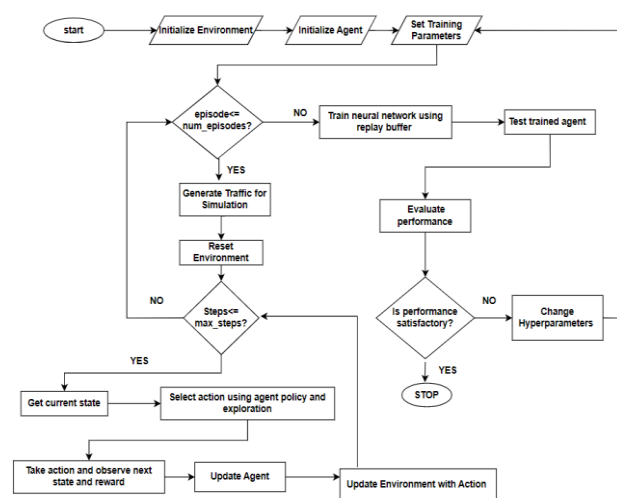


Fig. 1. Workflow of the proposed system

3. **State:** The state depicts the environment's current status at any given time. The history of the agent's observations and activities has a role in this. Oncoming lanes are divided into cells, which show whether at least one car is present or not. Each arm has 20 cells. The leftmost lane has 10 of them, while the other three lanes have the other 10. The junction has 80 cells in total.
4. **Action:** The action is the agent's output, which depicts what should be done given the environment's current condition. Choosing one of the four possible traffic light phases that were planned. Ten seconds are allotted to each step. Activation of a 4-second yellow phase occurs when the phase changes.
5. **Reward:** The Agent gets reward for the action performed on the Environment. The reward may be a positive reward or a negative reward. This signal represents the degree of the agent's success in achieving its goal. The cumulative waiting time of vehicles needs to be reduced for an efficient traffic light control system. Waiting time is the period of time since it was first parked when it has been standing still and moving at a complete stop. vehicle waiting time is no longer taken into account once it crosses a junction or enters an oncoming lane. As a result, the agent is rewarded favourably for reducing waiting times, which is the desired result.
6. **Policy:** The policy is the strategy or rule that the agent uses to select actions given the present state. mapping between states and actions is done by using policy. Agent learns the optimal policy through interaction with the environment.
7. **Replay Memory:** Replay memory is a key component in neural networks that are used in Reinforcement Learning, particularly in DRL. It is a buffer that stores the experiences of an agent in the form of tuples of observations, actions, rewards, and next-state observations. During the training of a DRL agent, the replay memory is used to store and recall past experiences for improving the training efficiency and stability of the neural network.
8. **Value Function:** Value function that calculates the agent's anticipated cumulative reward from a specific state. It is learned through iterative updates and is used to assess the goodness of a condition or an action.
9. **Neural Network:** The policy, value function, or both may be approximated using the neural network, which is a function approximator. Deep neural networks are often used to accomplish it, and backpropagation and gradient descent are used to train the networks.
10. **Learning Mechanism:** In this project, the relationship between states and actions are understood by the agent by using Neural Networks. For updating the action values of the Reinforcement agent uses the Bellman equation $Q(\text{state}, \text{action}) = \text{reward} + \gamma \max_{a'} Q'(\text{new State}, \text{new Action})$. The neural network has 4 hidden layers with a total of 300 neurons each, 80 input neurons representing the state, and 4 output neurons representing the 4 potential actions. At the conclusion of each episode, a certain number of random batched of data stored in replay memory is used for the neural networks training process. The Q-

learning equation is used for updating the action values and gathered samples are used for neural networks training. The replay of prior events is also made easier by this method.

3.2. Phases involved in Workflow

Phases involved in Workflow of the proposed work, DRL Approach for Smart TLCS is divided into following steps:

3.2.1. Environment Creation

For the agents to perform their actions, we set up the environment for the agent, which basically is the agent's world. The rules and dynamics of an environment cannot be influenced by the actions of the agent which interacts. We create the environment using the tool SUMO. SUMO is an open-source simulation tool for creating Traffic junctions. In sumo the creation of vehicles is possible. Each vehicle has a specific id, used for tracking the movement of vehicles. The Environment in SUMO is created using SUMO NETEDIT tool. It is flexible tool for creating traffic simulations that are pretty much similar to real-world simulations. The 4-way intersection can be generated using SUMO, which gives the base foundation for the environment creation. Each lane consists of 4 lanes which are incoming and 4 lanes which are outgoing. Each lane consists of a unique ID. Discretization of oncoming lanes into cells is done so that the presence or absence of any vehicles on a lane will be known. These cells are not fixed for each lane and each cell will have a unique ID, with which a vehicle's presence on a lane will be inspected. Figure 2 shows the Environment created using SUMO NETEDIT.

3.2.2. Traffic Generation

Traffic generation is a crucial part of simulation models used for predicting traffic flows and evaluating traffic scenarios. In this simulation model, 1000 cars are generated for every episode using a Weibull distribution. The Weibull distribution with shape 2 to determine the timing of car arrivals. The distribution ensures a quick increase in car arrivals until mid-episode, followed by a slow decrease to accurately reflect real-life traffic situations. Additionally, majority of the vehicles spawned to travel straight, while some vehicles turn left or right to account for different directions that vehicles may take. The traffic generation process is fair and unbiased since every vehicle has an equal chance of being spawned at the beginning of each road. The variability in the traffic generation process ensures that the simulation model accurately reflects the unpredictability of real-life traffic situations.

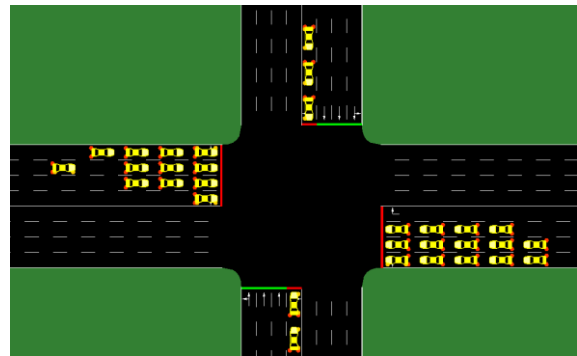


Fig 2. Environment Creation Using SUMO

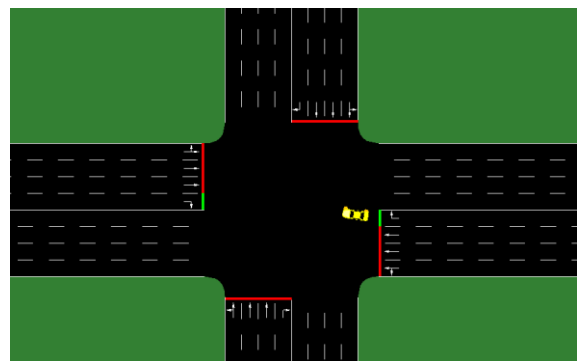
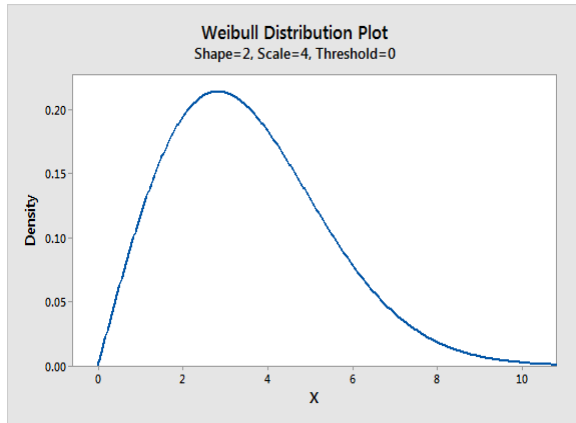


Fig 3. Vehicles taking turn at the intersection

The Weibull Distribution graph with slope 2 covers all cases of variable rates of traffic flow, including slow and steady as well as heavy traffic flow rates. Figure 4 shows the Weibull Distribution graph for generation of traffic.



3.2.3. Model Training

The training phase involves several steps. Firstly, the agent is initialized with random weights for the neural network. Then, the agent gets the environment's current state and chooses the best possible next state of the environment based on the optimal policy. This process involves the selection of appropriate actions that maximize the cumulative reward, as determined by the underlying Reinforcement learning algorithm. During each episode of the training phase, the agent takes multiple actions and receives corresponding rewards. These experiences are stored in the agent's memory buffer. After accumulating a specific number of experiences, a set of random experience replays is sampled from the memory buffer, and the Q-learning equation is utilized to update the neural network during the training phase.

The Q-learning equation helps the agent to update the action values by taking into account the observed reward and the estimated future rewards. This update

process continues for a set of episodes until the agent's performance improves and maximizes the rewards received. The neural network is trained with Backpropagation algorithm. This algorithm modifies the weights of the neurons based on the difference between the predicted and actual action values. Minimization of loss function is done by using Backpropagation algorithm. This helps the agent learn the relationship between state and action and improves its ability to choose actions based on the environment's current state.

3.2.4. System Testing

The system is tested to ensure that the trained model meets the desired performance goals and requirements. To test the proposed system, simulations can be conducted on various traffic scenarios using different performance metrics. The performance metrics can include cumulative waiting time, cumulative negative reward and queue length. The system can also be tested for its robustness against different types of traffic, such as heavy or light traffic, and against different traffic patterns.

During the testing phase, the system's behaviour can be compared with traditional traffic control methods, such as fixed-time control or actuated control. This comparison can help demonstrate the effectiveness of the proposed system compared to traditional methods. In addition, the system can be tested in real-world traffic scenarios to validate the simulation results and ensure that the system is ready for deployment. The real-world testing can be conducted in a controlled environment, such as a closed course, before being deployed in actual traffic conditions. Overall, system testing is crucial for verifying the effectiveness and reliability of the proposed traffic signal control system.

3.2.5. Model Evaluation

Model evaluation is the process of assessing the performance of a Machine Learning model to determine its accuracy and reliability. The evaluation of a model is essential to determine whether the model is useful for making predictions and to identify areas where the model can be improved. The system's performance is evaluated regularly to identify any areas for improvement. The Reinforcement Learning model is retrained periodically to incorporate new traffic data and optimize signal timings further. If the performance of the trained model is not up to the mark, then change the hyperparameters and again start the training process.

Python 3.10 is used for developing DRL approach for Smart TLCS. SUMO with version 1.14 is used. TensorFlow is used for Working with Neural Networks. The model is trained in laptop with i5 processor. For training the model it takes around 40 hours. The model is tested with various rates of traffic flow.

4. Results and Performance Analysis

The proposed system can be evaluated using the following evolution metrics.

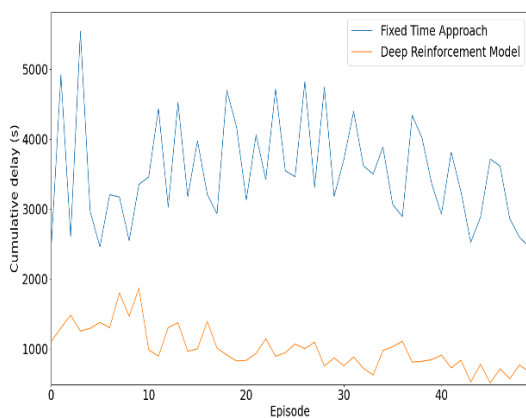


Fig 5. Episodes vs Delay Graph for Reinforcement model and Fixed time TLCS

4.1. Cumulative Waiting time

Cumulative waiting time refers to the total amount of time that vehicles have to wait at a traffic signal before being allowed to proceed. This waiting time is calculated by summing the waiting times of all vehicles that have passed through the intersection during a given time period. Episodes vs waiting time plot graph for Deep Reinforcement model and Fixed time approach is shown in Figure 5. Blue

color line graph represents the plot for the fixed time approach and the orange color line graph represents the plot for the DRL algorithm. The Episode vs Waiting Time plot will show that the waiting time for the DRL algorithm decreases more rapidly than that of the existing system. This suggests that the Deep Reinforcement Learning algorithm is learning faster and performing better at optimizing traffic signal timing. The comparison between the two systems indicates that the DRL algorithms has the ability to significantly decrease the congestion of traffic and delays at intersections, resulting in a more efficient traffic flow.

4.2. Average Queue Length

Average queue length is a metric used to evaluate the performance of traffic signal control systems in managing traffic flow and reducing congestion. It refers to the average number of vehicles waiting in a queue at a traffic signal during a given time period. Average Queue Length used in this research work is 700 meters. The vehicles that exist within 700 meters is taken into consideration, and the vehicles after 700 meters are not taken into consideration. Episodes vs Average Queue Length plot for the Deep Reinforcement model and Fixed time approach is shown in Figure 6. Blue color line graph represents the plot for the

Fixed time approach and the orange color line graph represents the plot for Deep Reinforcement Learning algorithm. The Episode vs Average Queue Length plot will show that the queue length for the Deep Reinforcement Learning algorithm decreases more rapidly than that of the existing system. This suggests that the Deep Reinforcement Learning algorithm is learning faster and performing better at optimizing traffic signal timing. The comparison between the two systems indicates that the Deep RL algorithms have the ability to significantly decrease the congestion of traffic and delays at intersections, resulting in a more efficient traffic flow.

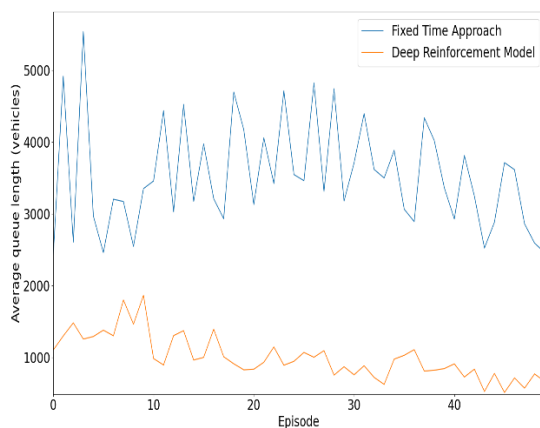


Fig. 6. Episodes vs Average Queue Length plot for Reinforcement model and Fixed time TLCS

4.3. Cumulative Negative Reward

Episodes vs Cumulative Negative Reward plot graph for Deep Reinforcement model and Fixed time approach is shown in Figure 7. Blue color line graph represents the plot for the fixed time approach and the orange color line graph represents the plot for Deep Reinforcement Learning algorithm. The Episode vs Cumulative Negative Reward plot will show that the Cumulative Negative Reward for the Reinforcement Learning

algorithm decreases more rapidly than that of the existing system. This suggests that the Reinforcement Learning algorithm is learning faster and performing better at optimizing traffic signal timing. The comparison between the two systems indicates that the Deep RL algorithms have the ability to significantly decrease the congestion of traffic and delays at intersections, resulting in a more efficient traffic flow.

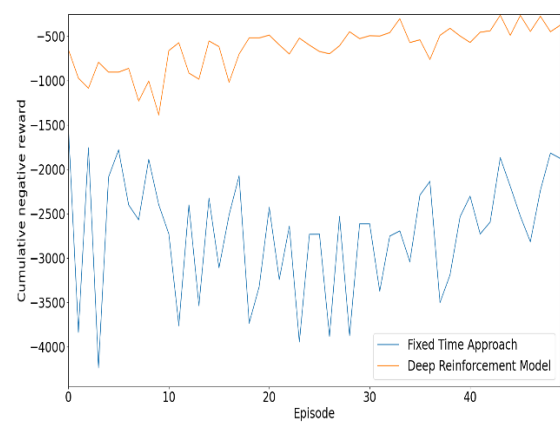


Fig. 7. Episodes vs Reward Graph for Reinforcement model and Fixed time TLCS

5. Conclusion

The use of Deep Reinforcement approach for Smart Traffic Signal Control has shown promising results in improving traffic flow and reducing congestion. The application of Deep Reinforcement Learning techniques has led to significant improvements in the optimization of traffic signal timing plans, with better performance compared to conventional approaches. By leveraging the power of machine learning algorithms, traffic signal control systems can become more adaptive and responsive to changing traffic conditions, resulting in a more efficient and sustainable transportation network. The results demonstrate that the DRL approach for smart TLCS outperforms Fixed time based TLCS in terms of reducing

traffic delay and improving the efficiency in finding better traffic signal timings. By implicitly modelling control actions and changes in system states, DRL algorithms, in contrast to flow-based TLCS schemes, simultaneously learn the dynamics of the TLCS and the best control strategy. Using the Deep Reinforcement Learning technique that has been developed and demonstrated to significantly outperform traditional methods will help you create a better signal timing strategy.

This research work can be further extended by including:

1. **Emergency Vehicles:** Traffic clearance for the emergency vehicles, that allow them to bypass traffic and reach their destination faster.
2. **Customized solutions for specific traffic scenarios:** v: Reinforcement learning algorithms can be customized for specific traffic scenarios, such as rush hour traffic or special events. This customization can help optimize traffic flow in these scenarios and reduce congestion.

References

- [1]. Li Li, Yisheng Lv, and Fei-Yue Wang, "Traffic Signal Timing via Deep Reinforcement Learning", IEEE/CAA J. of Autom. Sinica, vol. 3, no. 3, pp. 247-254, 2016.
- [2]. A. Salaymeh, L. Schwiebert, and S. Remias, "Multi-agent reinforcement learning for optimizing traffic signal timing," in CS & IT Conference Proceedings, vol. 11, CS & IT Conference Proceedings, 2021.
- [3]. N. Bhave, A. Dhagavkar, K. Dhande, M. Bana, and J. Joshi, "Smart signal-adaptive traffic signal control using reinforcement learning and object detection," in the 2019 third international conference on I-SMAC (IoT in social, mobile, analytics and cloud) (I-SMAC), pp. 624–628, IEEE, 2019.
- [4]. C. Li, F. Yan, Y. Zhou, J. Wu, and X. Wang, "A regional traffic signal control strategy with deep reinforcement learning," in 2018 37th Chinese Control Conference (CCC), pp. 7690–7695, IEEE, 2018.
- [5]. W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis, "Qtcp: Adaptive congestion control with reinforcement learning," IEEE Transactions on Network Science and Engineering, vol. 6, no. 3, pp. 445–458, 2018.
- [6]. M. Eom and B.-I. Kim, "The traffic signal control problem for intersections: a review," European transport research review, vol. 12, pp. 1–20, 2020.
- [7]. W. Genders and S. Razavi, "Evaluating reinforcement learning state representations for adaptive traffic signal control," Procedia computer science, vol. 130, pp. 26–33, 2018.
- [8]. S. A. Celtek, A. Durdu, and M. E. M. Ali, "Evaluating action durations for adaptive traffic signal control based on deep q-learning," International journal of intelligent transportation systems research, vol. 19, no. 3, pp. 557–571, 2021.
- [9]. D. Krajzewicz, "Traffic simulation with sumo-simulation of urban mobility," Fundamentals of traffic simulation, pp. 269–293, 2010.
- [10]. D. Leitner, P. Meleby, and L. Miao, "Recent advances in traffic signal performance evaluation," Journal of Traffic and transportation engineering (English edition), 2022.

[11]. S. Touhbi, M. A. Babram, T. Nguyen-Huu, N. Marilleau, M. L. Hbid, C. Cambier, and S. Stinckwich, "Adaptive traffic signal control: Exploring reward definition for reinforcement learning," *Procedia Computer Science*, vol. 109, pp. 513–520, 2017.

[12]. P. A. Rao, B. N. Kumar, S. Cadabam, and T. Praveena, "Distributed deep reinforcement learning using tensorflow", in *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, pp. 171–174, IEEE, 2017.

[13] N. S. Jadhao and A. S. Jadhao, "Traffic signal control using reinforcement learning," in *2014 fourth international conference on communication and network technologies*, pp. 1130–1135, IEEE, 2014.

[14]. A. L. Bazzan, "Opportunities for multiagent systems and multiagent reinforcement learning in traffic control", *Autonomous Agents and Multi-Agent Systems*, vol. 18, pp. 342–375, 2009.



Transaction Successful

08:56 pm on 11 Apr 2024

Paid to



XXXXXXXXXXXXXXXXXXXXXXXXXXXX ₹2,200

XXXXXXXXXXXXXXXXXXXXXXXXXXXXring

XXXXXX7368

Indian Bank

Sent to

: 67XXXXXX68@IDIB000S082.ifsc.npci



Transfer Details



Message

ICSIE240222

Transaction ID

T2404112056157793796477

Debited from



XXXXXX4316

₹2,200

UTR: 410294570904

Powered by

