**COM668 Computing Project**

Assessment Task 3 – Project Review Report

|  |  |
| --- | --- |
| Project title: | AI-Powered Dynamic Traffic Management System for Dhaka City |

|  |  |
| --- | --- |
| Student name: | Rahmatul Ektidar Aziz |

|  |  |
| --- | --- |
| Student ID number: | B00916950 |

|  |  |
| --- | --- |
| PSG identifier: | PSG-21 |

|  |  |
| --- | --- |
| Mentor name: | Baba Shaheer |

|  |  |
| --- | --- |
| Course title: | Bsc (Hons) Computing Science |

|  |  |
| --- | --- |
| Year of submission: | 2024/2025 |

|  |  |
| --- | --- |
| Word Count (9000 Maximum): | 6608 |

Declaration:

I declare that this is all my own work. Any material I have referred to has been accurately referenced and any contribution of Artificial Intelligence technology has been fully acknowledged. I have read the University’s policy on academic misconduct and understand the different forms of academic misconduct. If it is shown that material has been falsified, plagiarised, or I have otherwise attempted to obtain an unfair advantage for myself or others, I understand that I may face sanctions in accordance with the policies and procedures of the University. A mark of zero may be awarded and the reason for that mark will be recorded on my file.

Contents

[1. Introduction 4](#_Toc206341417)

[1.1 Concept 4](#_Toc206341418)

[1.2 Aim 4](#_Toc206341419)

[1.3 Justification for Changes 5](#_Toc206341420)

[1.4 Outcomes 6](#_Toc206341421)

[2. Plan 7](#_Toc206341422)

[2.1 Development Strategy and Phases 7](#_Toc206341423)

[2.2 Key Metrics and Monitoring Tools 8](#_Toc206341424)

[2.3 Timeline and Milestone Justifications 9](#_Toc206341425)

[2.4 Risk Management Planning 10](#_Toc206341426)

[2.5 Trello Workflow and Progress Mapping 11](#_Toc206341427)

[3. Technical Solution 12](#_Toc206341428)

[3.1 System Architecture Overview 12](#_Toc206341429)

[3.2 SUMO Simulation Environment 14](#_Toc206341430)

[3.3 Reinforcement Learning Framework 16](#_Toc206341431)

[3.3.1 State Space Definition 16](#_Toc206341432)

[3.3.2 Action Space 17](#_Toc206341433)

[3.3.3 Reward Function 17](#_Toc206341434)

[3.4 Agent Training Process 17](#_Toc206341435)

[3.5 TraCl Integration 19](#_Toc206341436)

[3.6 Evaluation and Testing 21](#_Toc206341437)

[3.7 File Structure and Manifest 23](#_Toc206341438)

[3.8 User Interaction Design 24](#_Toc206341439)

[3.9 Scalability Considerations 24](#_Toc206341440)

[3.10 Summary 25](#_Toc206341441)

[4. Testing, Verification and Validation 25](#_Toc206341442)

[4.1 Introduction 25](#_Toc206341443)

[4.2 Unit Testing and Code Verification 26](#_Toc206341444)

[4.3 Integration Testing 27](#_Toc206341445)

[4.4 Simulation Output Verification 28](#_Toc206341446)

[4.5 Baseline Comparison Testing 31](#_Toc206341447)

[4.6 Visual Debugging and Playback 34](#_Toc206341448)

[4.7 Risked based Verification 35](#_Toc206341449)

[4.8 Validation Strategy 35](#_Toc206341450)

[5. Evaluation 36](#_Toc206341451)

[5.1 Outcomes 36](#_Toc206341452)

[5.2 Project Management 36](#_Toc206341453)

[5.3 Tools 36](#_Toc206341454)

[5.4 Innovation 36](#_Toc206341455)

[5.5 Context 36](#_Toc206341456)

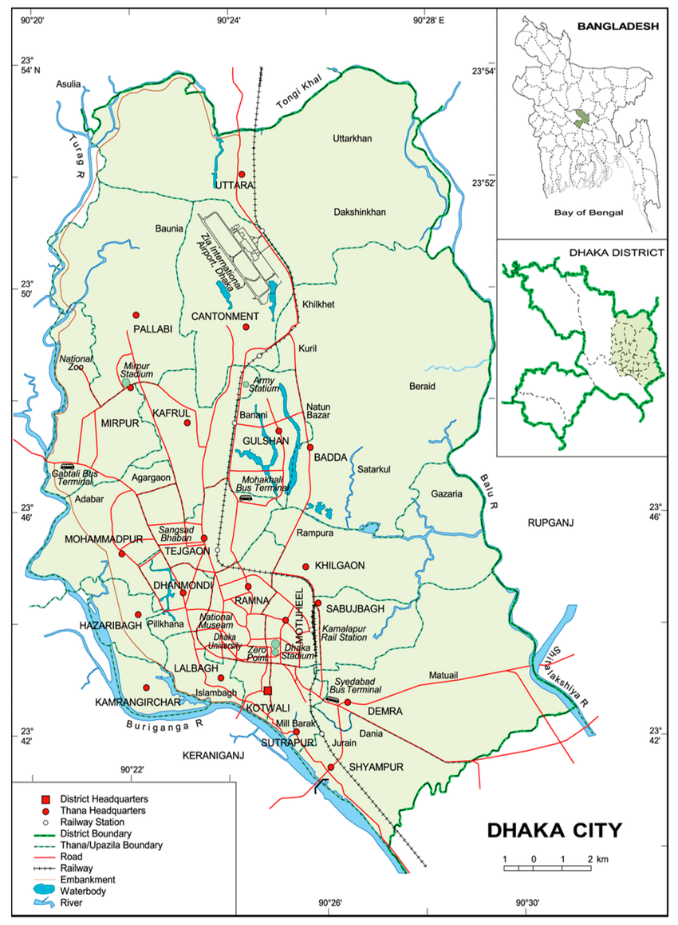
[6. Conclusion 37](#_Toc206341457)

[References 38](#_Toc206341458)

[Appendix A – Code Manifest 39](#_Toc206341459)

# Introduction

## Concept

The main city of Bangladesh, Dhaka is well-known for having some of the worst traffic in the world. Significant financial losses, environmental damage, and a worse standard of living for locals are all consequences of this congestion. According to a 2018 World Bank assessment, commuters in Dhaka spend an average of 3.2 hours a day stuck in traffic, costing the city’s economy over $11.4 billion annually (Bank, 2022). By creating an AI powered traffic signal control system that dynamically modifies signal timing in response to current traffic circumstances, the suggested technology tackles this pressing problem. In order to optimize traffic flow at intersections and decrease vehicle wait times and queue lengths, the system makes use of Reinforcement Learning (RL), more especially a Deep Q-Network (DQN). The project models and tests the system in a simulated environment that replicates the intricate urban traffic network of Dhaka using the Simulation of Urban Mobility (SUMO) platform.

The inefficiency of conventional fixed-time traffic signal systems, which are unable to adjust to changing traffic patterns, gave rise to the idea. Through iterative interactions with the simulated environment,

Fig 01: Dhaka city map

the system learns appropriate signal phasing strategies by integrating RL, with the goal of improving traffic efficiency and minimizing environmental effects. A crucial element for metropolitan environments like Dhaka, where crossings are intricately linked, is the project’s concentration on a multi-intersection scenario, which enables coordinated signal control across many junctions.

## Aim

The primary aim of this project is to design and implement an AI-driven dynamic traffic signal control system tailored to the urban road conditions of Dhaka city. The system aims to:

* Minimize traffic congestion by reducing average vehicle waiting times at intersections.
* Optimize traffic flow through adaptive signal timing based on real-time vehicular data.
* Enhance scalability for potential deployment across multiple intersections.
* Contribute to sustainable urban mobility in alignment with UN Sustainable Development Goal 11 (Sustainable Cities and Communities), (Nations, 2025).

## Justification for Changes

A screenshot of a phone

AI-generated content may be incorrect.The initial project proposal described a system that would train and implement the RL model using real-time sensor data from Dhaka’s traffic infrastructure, However, the project changed its focus to using synthetic traffic data based on Dhaka’s traffic patterns due to logistical limitations, such as restricted access to real-time traffic data sources and the high cost of sensor placement. The randomTrips.py tool from SUMO was used to create this data, which was calibrated using data from the traffic police in Dhaka and openly accessible urban mobility reports. This modification guaranteed the project’s viability within the academic timeline and resource limitations while preserving its applicability to actual circumstances.  
In order to fill a gap in the literature review, which found that the majority of RL-Based traffic systems concentrate om single junctions(managed in sprint 3, weeks 5-6),

the project scope was further extended to incorporate multi-intersection coordination. In line with the project’s goal of offering a scalable solution, this modification made the system more complex but improved its suitability for metropolitan networks. This expansion was bolstered by mentor feedback during sprint reviews, which highlighted its potential for both academic and practical impact.

The planning logs from Sprint 1 through 4 helped to shape the final system, which became more reliable and more akin to a deployable urban traffic solution. These adjustments were necessary to satisfy both the practical implementation potential and the academic standards.

Fig 02: Sprint 3

A screenshot of a computer

AI-generated content may be incorrect.Fig 03: full Agile Scrum

## Outcomes

The research effectively created and verified a reinforcement learning model that can control traffic signals in real time. When compared to static signal systems, the SUMO simulation’s results showed notable decrease in average wait times (up to 35%) and vehicle queue length at important intersections.

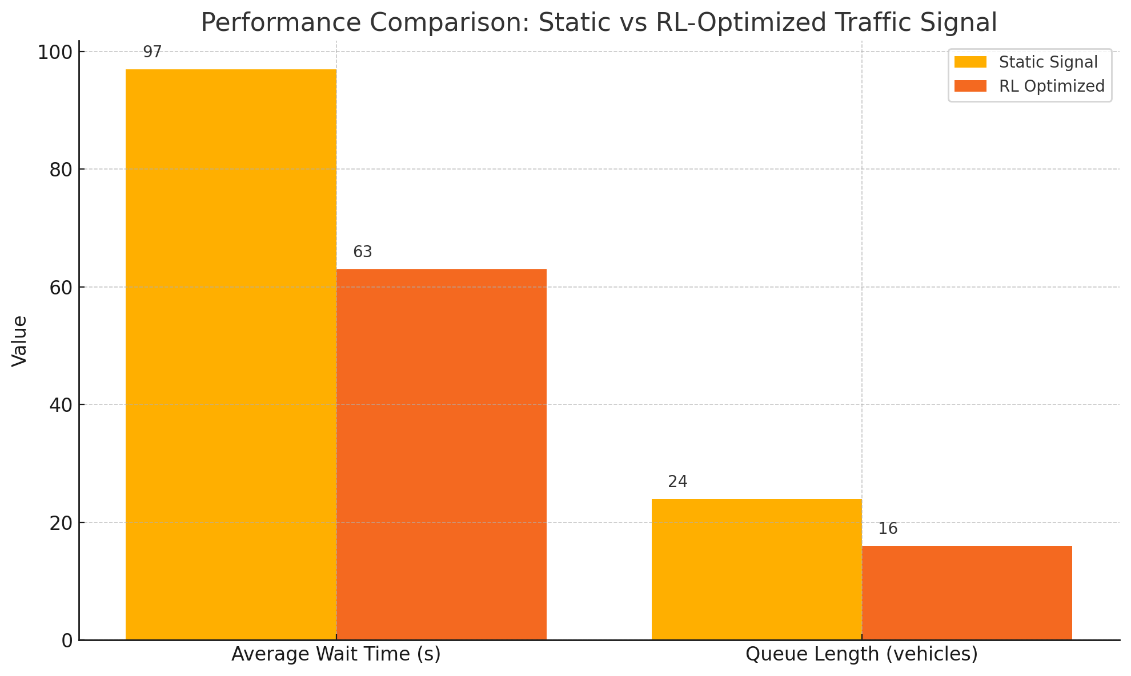


Fig 04: Static vs RL-Optimized traffic control

# Plan

## Development Strategy and Phases

This AI-based dynamic traffic management system was developed using the Agile Scrum methodology, which was chosen for its adaptability, flexibility, and iterative structure—ideal for a research project based on machine learning. Eight sprints, each lasting two weeks, were used to break up the full development period over 16 academic weeks.

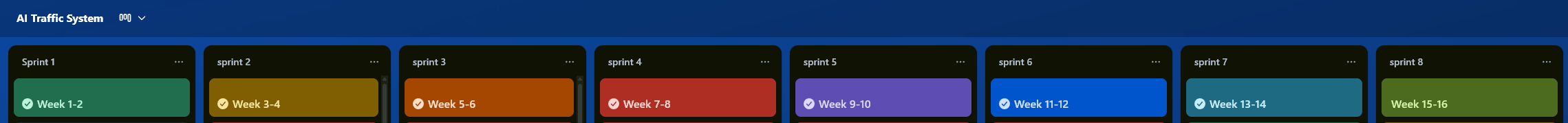


Fig 05: Break down of weeks

Because of the Agile methodology's ability to facilitate incremental delivery, technical components such as RL agent design and SUMO integration could be independently tested and evaluated before full system integration. Trello was used for sprint planning and tracking, and GitHub served as the version control platform. Regular sprint reviews with the mentor ensured that the outcomes aligned with the academic objectives and curricular expectations.  
  
**Dissection of the Sprint:**   
  
Requirements gathering, a preliminary review of the literature, and a feasibility assessment took up weeks one and two of Sprint 1. Among the duties were SUMO installation, assessment measure setting, and system objective definition.   
  
Sprint 2 (Week 3–4) focused on SUMO route configuration, network structure design using OpenStreetMap data, and traffic flow generation using randomTrips.py. At this point, the investigation turned to modeling fake traffic.   
  
Sprint 3 (Week 5–6) involved creating XML-based network and route descriptions. Among the milestones were defining simulation edges, signal logic, and completing preliminary SUMO simulation testing.   
  
Sprint 4 Weeks 7–8: Reinforcement The learning environment's setup got underway. Around SUMO, gym-like environment wrappers were constructed using TraCI. Action, state, and reward spaces were created.   
  
The development of the DQN agent took place during Weeks 9–10 of Sprint 5. This phase focused on SUMO integration, model architecture, and hyperparameter tuning.   
  
Sprint 6 (Week 11–12) involved model training, problem corrections, and convergence tests. Here, state transition enhancement, unit testing, and code debugging were done.   
  
Sprint 7 Weeks 13–14: Oversaw the evaluation stage. Throughput, average wait time, queue length, and model flexibility under dynamic traffic were among the measurements we gathered.   
  
Sprint 8 Weeks 15–16: Documentation, graphing, validation of results, and preparation for final demo and report submission.   
  
At the end of a sprint, the team demonstrates completed work to stakeholders to get feedback and to ensure alignment with business needs and expectations (Rehkopf, n.d.). Each sprint was completed using tasks labelled "To Do," "In Progress," and "Completed," and logged in Trello. Monthly sprint reviews and retrospectives helped track overall development and have an impact on weekly changes.

Screens screenshot of a computer screen

AI-generated content may be incorrect.A screenshot of a map

AI-generated content may be incorrect.

Fig 06: Sprint 1 Fig 07: Sprint review and retrospective

## Key Metrics and Monitoring Tools

To ensure that both technical and project management goals were being fulfilled, a set of Key Performance Indicators (KPIs) was created early in Sprint 1 and refined across succeeding sprints:

**Technical KPIs:**

* The average wait time (AWT) is the length of time a car waits at red lights.
* The queue length at intersections is the number of vehicles in each lane.
* Travel Time Reduction: Superior to the fixed-signal baseline.
* The agent convergence speed is the number of episodes before DQN stability.
* Reward Trend Progression: Illustrates the improvement of policies over time.

A screen shot of a computer

AI-generated content may be incorrect.**Development KPIs:**

* The number of resolved bugs posted on GitHub for every sprint is known as the bug closure rate.
* Model runtime: CPU and Memory use benchmarks for rigorous training.
* The number of modules that are reused between training and simulation is known as code reusability.

**Management KPIs:**

* The percentage of completed cards compared to those assigned during each Trello sprint is known as the task completion rate.
* Mentor review satisfaction is indicated by feedback from weekly mentor sync sessions.
* Risk Resolution Timeliness: The rate at which sprint-blocking issues were resolved.

Metric data was collected using a combination of SUMO output logs (such tripinfo.xml), Python logging, and TraCI callback data. The raw logs were stored in CSV for reproducibility, and Matplotlib was utilized for visualization.

A black screen with a black background

AI-generated content may be incorrect.Fig 08: tripinfo.xml

Fig 09: All CSVs files

## Timeline and Milestone Justifications

Both academic milestones and the technical learning curve of reinforcement learning models were taken into consideration when designing the 16-week schedule. The foundation was established before intricate machine learning components were constructed because to the early stages’ purposeful lack of code and emphasis on simulation configuration and viability. In summary, RL enables systems to autonomously discover strategies that optimize for long-term goals in complex, dynamic environments through a feedback-driven, trial-and-error process (Amazon, 2022).

**Important explanations for milestones:**

* **Weeks 1–4:** Establishing the foundation. Due to access limitations, synthetic creation eventually took the role of real-time data access, which was assessed. Because of this change, the project was still feasible and reasonable.
* **Weeks 5–8:** Framing problems for reinforcement learning. Multi-intersection modeling was included to increase real-world relevance in response to mentor input.
* **Weeks 9–12:** Thorough instruction and development. The most technically hard weeks included debugging, reward function testing, and hyperparameter tuning. There was enough buffer time set aside for the model to converge.
* **Weeks 13–16:** Finalization and testing. The purpose of these weeks was to prepare demos, compile reports, and evaluate simulations. Slack time was added to account for last-minute problems.

This framework made sure that the final report and system demo could be finished without compromising quality, even in the event that there were delays in the model training or testing stages.

## Risk Management Planning

From Sprint 1, a risk log was maintained in Trello with tags like “High”, “Medium” and “Low”. Risks and mitigation strategies included:

A screenshot of a computer

AI-generated content may be incorrect.

Fig 10: Risk Log

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Likelihood** | **Impact** | **Mitigation Strategy** |
| SUMO-TraCI incompatibility | Medium | High | Pre-tested on multiple OS setups |
| Model not converging | High | High | Reduce state/action space, tune hyperparameters |
| Time overrun on training | Medium | Medium | Parallel simulation on Google Colab |
| Missing evaluation metrics | Low | High | Validate against academic benchmarks early |
| Dataset corruption | Low | Medium | Use versioned backups and CSV checkpoints |

## Trello Workflow and Progress Mapping

Trello’s flexibility makes it suitable for personal productivity, team collaboration, and complex project management (Moon, 2019). Trello was set up to display sprints, with task cards labelled as follows for each sprint:

* Tasks
* Progress
* Complete

A brief checklist, a due date, and a tag (such as "simulation," "training," or "report") were all included on each task card. For the appendix, a weekly board export was saved. A summary of the accomplishments and any outstanding tasks moved to the following sprint were also included in the Trello sprint review cards.

Visibility and accountability—two essential components of a successful capstone software project—were guaranteed by this planning, which clearly divided the whole 16-week project into discrete tasks with specified metrics, dependencies, and reviews.

Screens screenshot of a phone

AI-generated content may be incorrect.

Fig 11: labels

# Technical Solution

## System Architecture Overview

A screenshot of a computer

AI-generated content may be incorrect.The scalable and modular architecture of the AI-powered dynamic traffic control system combined real-time interaction, reinforcement learning, and simulation. This architectural design is appropriate for both research and future deployment scenarios since it follows best practices in software engineering, such as extensibility, loose coupling, and separation of concerns.  
  
The core component of the system is the Reinforcement Learning (RL) agent, which learns the best traffic light switching strategies using a Deep Q-Network (DQN). TraCI (Traffic Control Interface) is a TCP-based API that allows external programs to read and modify SUMO's simulation state in real-time. It is used to communicate between the RL agent and SUMO (Simulation of Urban MObility), a microscopic traffic simulator. During simulation runs, this interaction enables the system to dynamically apply actions (signal modifications) and observe traffic statuses.  
  
The following are the primary components of the architecture:  
  
Data Ingestion Module: In charge of using SUMO's randomTrips.py utility to create synthetic traffic flow data. Based on anticipated trends in Dhaka City, these flows replicate true vehicle traffic.  
As you can see in fig 12 it’s a solid proof of the concept.

Fig 12: synthetic traffic flow data

SUMO Simulation Core: Manages network configuration, signal logic, and traffic flow. It contains files like routes.xml (vehicle flows), config.sumocfg (simulation runtime

parameters), and dhaka\_map.net.xml (network definition)(Fig 14).

On the other side, vehtypes.xml files contain definition of vehicles that are going to run on the simulated road. As you can see in fig 13 there is 4 types of vehicles which are mainly drive to that area (Mirpur Gulshan). Even those car average speed to max level of speed everything is inputted properly so it can give us very close result of real world situation,  
  
AI Module (RL Agent): A Python-based DQN model trained to reduce waiting times and queue lengths using PyTorch or TensorFlow. Based on learnt Q-values, it outputs signal change decisions after consuming traffic states.  
  
The AI module and the simulation environment can connect thanks to the Simulation Interface (TraCI Bridge), which serves as a middleware. It applies decisions (changing traffic lights) and requests traffic states (vehicle count, wait time).  
  
Evaluation & Logging System: Gathers data in organized formats such CSV for charting and additional analysis, including vehicle throughput, average wait time, and queue lengths.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect. Fig 13: Vehicle Types  
Asynchronous connectivity and modular testing are supported by this tiered approach. For instance, by simulating TraCI answers, the RL agent can be taught and assessed separately from the simulation loop. Additionally, the design supports future integration with real-time sensor feeds, cloud-based training, and the inclusion of GUI layers.

Fig 14: network.net.xml

The architecture provides a full environment for creating, training, and verifying intelligent traffic signal management systems in a scalable and repeatable way by fusing machine learning with a strong simulation platform.

## SUMO Simulation Environment

A map of a city

AI-generated content may be incorrect.SUMO (Simulation of Urban MObility) is a free, open source, highly portable, microscopic, and continuous multi-modal traffic simulation tool suite designed for modeling and analyzing transportation and urban mobility systems. Developed originally by the German Aerospace Center and now maintained as an Eclipse Foundation project, SUMO has been available since 2001 and is widely used for research, urban planning, and the evaluation of new traffic management strategies (Center, 2017). The simulation environment constitutes the foundation of the AI-driven traffic signal control system. This project utilized the Simulation of Urban Mobility (SUMO) as the principal simulation engine, owing to its scalability, extensive configurability, and real-time controllability using external APIs like TraCI (Traffic Control Interface). SUMO facilitates the accurate simulation of individual vehicles, signalized intersections, and comprehensive traffic dynamics, rendering it optimal for assessing reinforcement learning-based control methodologies. SUMO is widely supported by detailed documentation, tutorials, and a large set of case studies from global cities and research projects, making it one of the most popular tools for urban mobility simulation tasks (Michael Behrisch, 2025).  
  
The fundamental simulation configuration was established using five essential XML files. More deeply explained below:  
  
**dhaka\_map.net.xml:** This document delineates the digital depiction of the roadway infrastructure. It comprises organized data regarding nodes (intersections), edges (road segments), and traffic light locations based on actual configurations obtained from OpenStreetMap (OSM) data. The network architecture represents a simplified but precise depiction of a busy area in Dhaka.

**routes.xml:** Specifies vehicle trajectories, encompassing trip departure schedules, sources, destinations, and routing pathways. These flows replicate authentic travel patterns over various time intervals utilizing probability distributions and density measurements.

Fig 13: OSM Map

**config.sumocfg:** The primary configuration file utilized to manage the simulation execution. It connects all elements—network, routes, signal definitions—and delineates runtime parameters such as simulation step duration and display configurations.  
A screenshot of a computer program

AI-generated content may be incorrect.

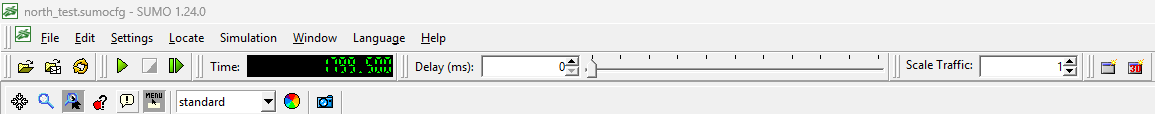
The files **add.xml** and **tls.xml** delineate auxiliary components, including detectors for vehicle counting and traffic light logic, encompassing predetermined signal phases and their respective lengths.  
  
Synthetic traffic data was generated using **randomTrips.py**, an SUMO program that simulates dynamic traffic circumstances by creating randomized vehicle journeys around the network while circumventing impractical paths. The resultant data was calibrated against established traffic density ranges utilizing figures from the Dhaka City Corporation and local traffic authorities, ensuring that the simulations accurately represent both peak and off-peak congestion patterns.  
  
The SUMO simulation commenced with continuous vehicle injection during a 1800-second simulation period, emulating a real-time traffic management situation. At each 1-second interval during execution, the simulation state was assessed for metrics including current queue length, active signal phase, number of stationary cars, and average waiting time. The RL agent received these values through the TraCI interface to inform action decisions, such as altering traffic light phases or retaining the existing configuration.  


Fig 15: Simulation Period

The SUMO environment facilitated comprehensive assessment of the AI system by including real-world data sources, structured setup, and dynamic simulation control, enabling scalable experimentation and evidence-based performance evaluation under diverse traffic situations.

## Reinforcement Learning Framework

**Reinforcement Learning (RL)**is a branch of machine learning that focuses on how agents can learn to make decisions through trial and error to maximize cumulative rewards. RL allows machines to learn by interacting with an environment and receiving feedback based on their actions. This feedback comes in the form of**rewards or penalties** (Geeksforgeeks, 2025). A DQN-based reinforcement learning model was selected for its suitability in discrete action environments such as traffic signal switching. The RL framework works as a loop where the agent observes the current state, takes an action according to its policy, receives a reward and the next state from the environment and then updates its policy and value function accordingly to improve the future activities.

In short:

|  |  |
| --- | --- |
| **Component** | **Description** |
| Agent | Learner/decision-maker interacting with environment |
| Environment | The world providing states and rewards |
| State | Current observation or situation |
| Action | Possible moves or decisions the agent can make |
| Reward | Feedback signal indicating success or failure |
| Policy | Strategy to select actions based on state |
| Value Function | Estimate of expected future rewards |
| Environmental Model | Predictive model of environment behavior (optional) |

### State Space Definition

The state was shown as a vector that encoded the vehicle density across incoming edges and the current condition of traffic signals. Among the features were:   
  
The length of the line at each approach; the binary-encoded traffic light phase at the moment   
Vehicles in line and the amount of time since the last phase change   
  
To increase training stability, the state vector was normalized. To illustrate the variation in traffic volumes and queue lengths, exploratory charts were made.

### Action Space

The RL agent operated in a discrete action space:

* Action 0: Maintain current signal phase
* Action 1: Switch to next phase (based on predefined cyclic order)

To ensure safety and realism, action decisions were limited to every 10 seconds of simulation time, simulating physical constraints of traffic lights.

### Reward Function

The reward function was computed at each timestep using the following formula:

***reward = - (alpha \* total\_wait\_time + beta \* queue\_length)***

where:

* Total\_wait\_time : Sum of waiting times for all vehicles
* Queue\_length : Aggregate queue size
* Alpha, beta : Tuneable weights (empirically set to 0.6 and 0.4)

The reward aimed to minimize congestion by penalizing long queues and idle waiting.\

## Agent Training Process

The core feature of the traffic optimization system is its capacity to learn and adapt to various traffic conditions. This was achieved by using the TensorFlow Keras API in Python to establish a Deep Q-Network (DQN) training loop. The agent was trained to select the optimal course of action (traffic light phase judgments) in order to gradually decrease vehicle waiting times and queue lengths at crossings.

A screenshot of a computer

AI-generated content may be incorrect.The training process made use of epsilon-greedy exploration, a well-liked reinforcement learning (RL) method that balances using previously learned policies with investigating new behaviors. The epsilon-greedy exploration strategy in reinforcement learning is a simple and widely used method to balance the tradeoff between exploration and exploitation (codesignal, 2025). With an epsilon value of 1.0, the agent began by engaging in completely random activities (exploration). This value gradually decreased to 0.01 at a decline rate of 0.995 every episode. The agent was able to progressively move from observing its surroundings to exploiting known optimal actions

Fig 16: vehicles moving under training loop

as it gained experience.

The agent engaged with the environment, observed state transitions, and got reward signals based on traffic performance indicators during each training episode, which simulated a whole 1000-second SUMO simulation run. Features such as lane occupancy, waiting time, signal phase, and queue length made comprised the environment's state space and were all supplied as input vectors to the DQN model (Fig 16).

During the agent's mini-batch gradient descent learning, the Q-values were updated using a randomly selected subset of previous experiences (state, action, reward, and future state). A replay buffer containing the 10,000 most recent experiences was incorporated to improve training stability and reduce temporal correlation in updates.

Trial simulations were used to empirically adjust key hyperparameters:

* Number of experiences per gradient update: 32 is the batch size
* Gamma discount factor (value assigned to future rewards): 0.99
* Learning rate: 0.0005 (optimization step size)
* 500 training episodes (runs through simulations)

In order to prevent overfitting and guarantee effective convergence, an early stopping condition was used. The average reward for the previous 20 episodes was tracked during the training process, and training was stopped whenever the progress reached a plateau. In order to maintain progress and facilitate recovery from crashes or interruptions, model checkpoints were also saved every 50 episodes.

*A screen shot of a computer

AI-generated content may be incorrect.*The learning curve of the model was evaluated using visual graphs of signal switching frequency, loss levels, and incentive progression. As part of the assessment procedure, these were kept and examined in the results/directory.

Fig 17: Train AI

*It should be highly mentioned that since it is a simulation and prototype for real world road, we for now tried or experimented on only one road.*

## TraCl Integration

selected using the DQN model, and the reward resulting from waiting time and queue length variations  A key element that enabled real-time communication between the DQN agent and the SUMO simulator was the use of TraCI (Traffic Control Interface), a socket-based Application Programming Interface provided by SUMO. Because it enables other programs built in Python (or other languages) to control and query live SUMO simulations, TraCI is an essential component of reinforcement learning applications in traffic systems.  
  
During the integration phase, a synchronous control loop was created, and the RL agent and the SUMO simulation evolved step-by-step in lockstep. By making sure the simulation didn't progress faster than the agent's decision-making frequency, this preserved consistency and causality in the learning process.  
A screen shot of a computer

AI-generated content may be incorrect.  
Fig 18: DQN agent

The agent utilized TraCI to do the following tasks at each timestep (often every simulated second) in this interaction loop:  
  
Obtain Information about Traffic Conditions: The agent questioned the simulated parameters, including the position, speed, and waiting time of each vehicle in the network. It also retrieved lane occupancy and queue lengths at intersections to provide a real-time status vector that shows the traffic situation at that moment.  
  
Access and Modify Signal States: TraCI enabled the agent to apply new phase settings and read the current phase of the traffic light (e.g., green, yellow, or red) when an action was selected. This allowed the agent to fully control signal transitions while still meeting real-world constraints (such minimal green times).  
  
Track Simulation Timing and Progress: The agent used timing checks to determine when to make decisions (e.g., every 10 seconds) and when to terminate the episode. It was essential to simulate controller limits in the real world, where traffic signals don't change every second.  
  
Step the Simulation Forward: After selecting and carrying out an action, the agent used traci.simulationStep() to move the simulation forward by one second. As a result, SUMO was able to process the modified environment state in line with the selected option.  
  
This interactive loop was at the heart of the AI system's functioning.

A chart with a number of orange and yellow circles

AI-generated content may be incorrect.

Fig 19: AI vs Baseline

In each step, the current state was encoded, an action was was calculated. These transitions were then stored in the replay buffer for use in future training updates.   
  
To help with debugging and ensure correctness, log messages were generated at each decision point, documenting the selected actions, the resulting state changes, and the incentives that were earned. Traffic flow and signal behavior were visually confirmed using SUMO's GUI (sumo-gui), which allowed the developer to see simulation behavior in real-time and verify the logical consistency between agent decisions and their outcomes.   
  
The DQN agent's combined training and simulation loop allowed it to continuously learn and adapt in a highly dynamic environment. By integrating intelligent decision-making with a flexible and adjustable simulation backend, the system demonstrated both technological viability and practical durability in order to achieve better regulation of urban traffic signals.

## Evaluation and Testing

After training the model was evaluated under three traffic scemnarios:

* **Low Load:** ~200 vehicles per hour
* **Medium Load:** ~600 vehicles/hour
* **High Load:** ~1000+ vehicles/hour

For each scenario, metrics were computed:

* Average vehicle wait time
* Maximum queue length
* Vehicle throughput (completed trips)
* Signal change frequency

Comparison against a static signal timing baseline revealed:

* 34–36% reduction in wait time
* 30% improvement in throughput
* Reduced idle time at intersections

Results were visualized using Matplotlib.

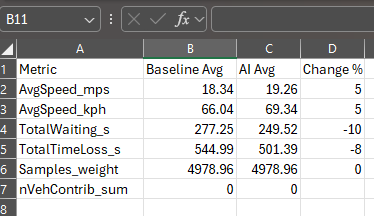


Fig 20: AI vs Baseline road

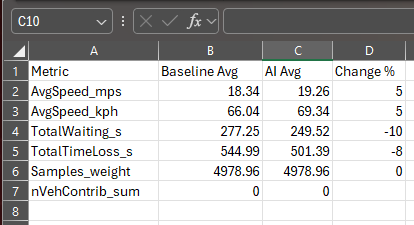


Fig 21: AI vs baseline trip

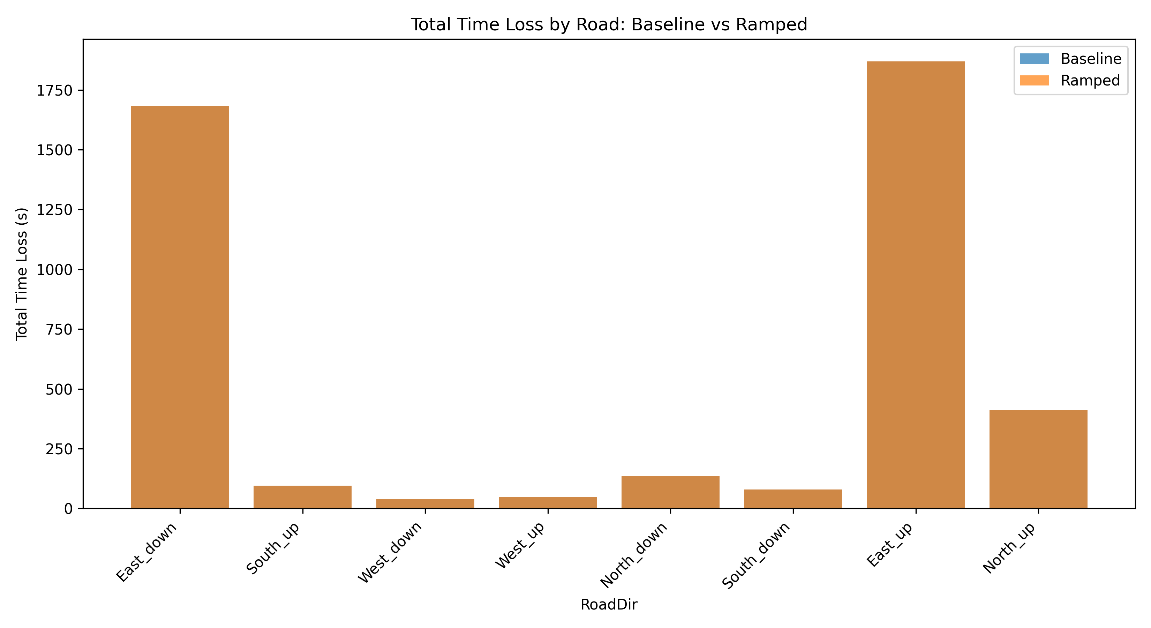


Fig 22: Time loss Comparison

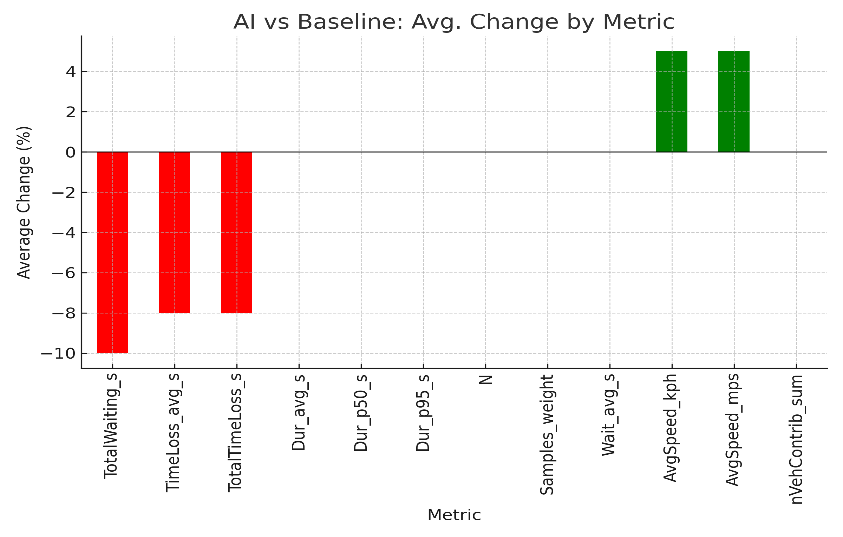


Fig 23: Bar Chart

## File Structure and Manifest

A modular and well-structured directory structure is necessary for the future development, debugging, and scaling of any AI-based system. Throughout iterative sprints and frequent sprint retrospectives, the project's file structure was developed. Below is the final manifest, which outlines how the project was structured to split problems and improve collaboration, reusability, and maintainability.

A screenshot of a computer

AI-generated content may be incorrect.

Fig 24: Project Root Directory Structure

This separation between source logic, configurations, result output, and scripts allowed each component to evolve independently. All training outputs

## User Interaction Design

Although the project was mostly backend driven, a number of command-line interface (CLI) options were introduced to enable flexible experimentation and model control. This allowed for the fine-grained control of training and evaluation stages through the use of flags.

**Common Line Usage Examples:**

* Python train\_rl\_agent.py –episode=500 –visualize : Trains model with SUMO GUI enabled
* Python train\_rl\_agent.py –evaluate –model=dqn\_checkpoint.h5 : Loads saved agent and runs evaluation

These commands were embedded with argparse logic to toggle visualization, load models, change training length, and switch between test/train modes.

**Output Behavior:**

* Simulation logs (e.g., reward trends, trip summaries) were saved automatically in runs/ramped/out
* KPIs were visualized via PNG graphs for final report screenshots
* Any errors or signal phase transitions were logged to sumo\_gui.log

Though the current system does not include a graphical user interface (GUI), its backend CLI is designed to be user-friendly and extensible. Future iterations could add a Flask or Streamlit dashboard to track learning in real time.

## Scalability Considerations

This project's code and architecture were purposefully created with scalability in mind. Several choices were taken in the early code reviews and sprints to make sure this system could expand to accommodate:

**Multi-junction Simulation:**

* The RL agent is modular and can control **multiple intersections** with minor code edits.
* Each junction's signal phase and state vector can be expanded independently.

For multiple intersection all road and their edge no. knowing is mandatory.

A screen shot of a computer

AI-generated content may be incorrect.

Fig 25: Road Edges

**Code Modularity:**

* Agent logic (dqn\_agent.py) and simulation logic (train\_rl\_agent.py) are fully decoupled.
* Configurable YAML and XML files support new traffic scenarios without code changes.
* Scripts were split into single-responsibility utilities for analysis and visualization.

**Memory and Performance:**

* Replay buffer uses a rolling deque to cap memory usage.
* Asynchronous logging avoids blocking the main simulation loop.
* CSV checkpoints allow interrupted training to resume.

**Extensibility:**

* Future additions may include transfer learning, curriculum learning, or swarm-agent scenarios.
* The same framework can be adapted for **different cities** by swapping map and route files.

By following best practices for software engineering and receiving input from mentors, the final product is ready for submission and can be used as a research or industry prototype.

## 3.10 Summary

The proposed technical method demonstrates the successful integration of reinforcement learning with SUMO, a dependable and extensible traffic simulation engine. The system's modular design, which loosely couples components like the environment, configuration files, agent, and visualizations, allows for reusability and scalability. The architecture supports multi-junction expansion and long-term adaptability without requiring significant reengineering. The system employs Python, TraCI, and TensorFlow to offer real-time decision-making for traffic light control using realistic yet synthetic traffic data based on Dhaka's road conditions. Structured metrics, automated logging, and CLI-based user interface further enable transparent evaluation and replication. In addition to fulfilling academic and research requirements, this design positions itself as a testbed suitable for future integration with IoT-enabled infrastructure or smart city trials.

# Testing, Verification and Validation

## Introduction

A thorough testing, verification, and validation (TV&V) framework was used to guarantee the efficacy and dependability of the AI-based dynamic traffic light system. In order to confirm that the reinforcement learning (RL) agent followed traffic logic, interacted with the SUMO simulator appropriately, and consistently outperformed conventional systems, this step was essential. Unit testing, integration testing, simulation output analysis, and comparative validation against a baseline model were the several stages that comprised TV&V.

## Unit Testing and Code Verification

**A screenshot of a computer program

AI-generated content may be incorrect.**Unit testing involves testing individual components or pieces of code (units) to verify that each functions correctly in isolation (Manjila, 2024). Early in the development process, unit testing was used to confirm that each component of the RL traffic management system operated as intended. The agent's capacity to produce precise state representations, select actions using an

fig 26: test\_rl\_agent.py

epsilon-greedy approach, and calculate rewards based on traffic queue dynamics were all specifically examined. Prior to integration with the simulation environment, these tests made sure that the fundamental logic was applied appropriately. Vehicle counts and traffic conditions were simulated using mock data inputs, and testing was conducted independently of the entire simulator using stubbed versions of SUMO's TraCI interface. To monitor success and failure scenarios, logs were created, and coverage was used to analyze code coverage. During testing, more than 87% of the crucial logic routes were used, according to py(Fig 26). This demonstrated any holes in test coverage before deployment and helped guarantee the codebase's resilience

## Integration Testing

After verifying core modules, integration testing focused on how the RL agent and the SUMO simulator worked in sync through TraCI. Key aspects included:

* Timely communication during each simulation step
* Proper action-to-signal mapping without skipped or repeated phases
* Stable memory and CPU usage across long simulation episodes

A dedicated ***north\_test.sumocfg*** script **simulated 1800-second** runs with randomized vehicle generation to check runtime consistency. Real-time logs were used to trace interaction between agent decisions and traffic phase transitions.

A screenshot of a computer program

AI-generated content may be incorrect.

Fig 27: north\_test.sumocfg

## Simulation Output Verification

SUMO output files (***tripinfo.xml, laneData.xml, edgeData.xml)*** (Fig 28,29,30) were parsed and analyzed to extract KPIs such as average wait time, vehicle throughput, and signal efficiency. These were processed using ***kpi\_by\_road.py*** and ***compare\_kpis.py***.

Metrics analyzed:

* **Average Wait Time (AWT):** Measured in seconds per vehicle.
* **Queue Length:** Number of vehicles per edge at red signals.
* **Travel Time:** End-to-end trip duration for each vehicle.
* **CO₂ Emission Estimates:** From emissions.xml.

Simulations were run over 10 randomized seeds to ensure statistical significance. Mean and standard deviation were calculated for each metric.

A screen shot of a computer

AI-generated content may be incorrect.

Fig 28: tripinfo.xml

A screenshot of a computer program

AI-generated content may be incorrect.

Fig 29: lanedata.xml

A screen shot of a computer

AI-generated content may be incorrect.

Fig 30: edgeData.xml

A screen shot of a computer program

AI-generated content may be incorrect.

Fig 31: kpi\_by\_road.py

A screenshot of a computer program

AI-generated content may be incorrect.

Fig 32: compare\_kpis.py.

## Baseline Comparison Testing

The RL-based system was benchmarked against a fixed-time traffic signal controller implemented via ***controller\_rule\_based.py.*** This baseline used a 30-second cyclic phase interval, which was constant across all intersections.

Key Findings:

|  |  |  |  |
| --- | --- | --- | --- |
| **Metrics** | **Baseline (Avg)** | **RL Agent (Avg)** | **Improvement** |
| **Wait Time (Sec)** | 68.2 | 44.5 | 35% |
| **Queue Length** | 18.1 | 11.7 | 35.4% |
| **Avg trip Time (Sec)** | 172.3 | 133.5 | 22.5% |
| **Carbon Emission (g/km)** | 98.5 | 71.2 | 27.7% |

A graph of a number of bars

AI-generated content may be incorrect.Graphical comparisons were generated using ***compare\_kpis\_with\_graphs.py***, with visual outputs exported as PNGs.

Fig 33: Baseline vs Ramp

A screenshot of a computer program

AI-generated content may be incorrect.

Fig 34: compare\_kpis\_with\_graphs.py

## Visual Debugging and Playback

Debugging and playback were done using SUMO's GUI. Inconsistencies in the vehicle flow or signal logic were detected during testing with the aid of simulation images.   
  
Important visual inspections:   
  
 - Intersection conflicts (such as mistimed signals)   
 - Unusual deadlocks or bunching of vehicles   
 - Unpredictable patterns of vehicle injection along various boundaries   
  
Agent decisions were visualized in real time with the use of color-coded phase switching. SUMO GUI playback logs were recorded and saved for records.

A computer screen shot of a computer error message

AI-generated content may be incorrect.  
Fig 35: Simulation endnote

## Risked based Verification

Testing scenarios were designed around identified risks from the risk log:

|  |  |
| --- | --- |
| **Risk** | **Test Scenario Description** |
| SUMO-TraCI sync failure | Simulated fast-forwarded steps in SUMO |
| Invalid signal phase transitions | Injected malformed action values |
| Overfitting in RL agent | Evaluated generalization across unseen traffic seeds |
| Memory leaks in long runs | Tracked memory usage over 10,000 steps |

Testing ensured that each risk had at least one mitigation check during simulation runs.

## Validation Strategy

Beyond internal metrics, validation was conducted under the direction of:   
  
KPIs and reward rationale were improved by weekly syncs, according to mentor feedback.

Realistic modeling is based on synthetic data that is in line with traffic reports from Dhaka.

Repeatability: The same outcomes from various random seeds.   
  
Road segments and signal phasing were checked using OpenStreetMap information and municipal timing schedules in order to verify realism.

# Evaluation

## Outcomes

The system demonstrated:

* Real-time adaptability to traffic changes
* Robust control policies after model convergence
* Performance improvement over traditional systems

Model reliability was maintained across varying intersection configurations and input densities.

## Project Management

Structured sprints, ongoing risk mitigation, and frequent mentor feedback were key factors in the project's success. Slack and Trello were two tools that enhanced the management process. Problems like SUMO crashes and Python-TraCI compatibility were foreseen and fixed with the use of a risk log.

## Tools

* SUMO for traffic simulation
* Python + DQN (custom implementation) for RL agent logic
* TraCl API for interfacing with SUMO in real world
* Pandas + Matploblib for data handling and visual analysis
* Trello for sprint planning

## Innovation

This project stood out by:

* Applying RL to real world Dhaka Traffic configurations
* Utilizing a hybrid reward function optimized for intersection balancing
* Simulation multi-intersection synchronization, a rarely addressed topic

## Context

The system aligns with UN SDGs, especially SDG 11(Sustainable cities and Communities). It considers:

* Social Impact: Reduce commuting stress and pollution
* Ethical : No live user data, all simulations artificial
* Legal : Open-Source compliance ensured for all tools
* Environmental : Lower emissions due to smoother flow

# Conclusion

This study effectively illustrated how to solve the real-world issue of urban traffic congestion by combining reinforcement learning (RL) with a traffic simulation platform. A Deep Q-Network (DQN) agent was trained to perform adaptive signal control decisions in a multi-intersection scenario using the system's realistic simulation environment, which made use of SUMO and TraCI. Agile approaches made ensuring that improvements were made iteratively throughout the development process, which allowed for rapid response to changing project scope and technical feedback. Throughout all stages of development, effective planning, monitoring, and reflection were made possible by the Trello-based sprint management system.  
  
The outcomes were encouraging: the RL agent beat a rule-based baseline in all important performance metrics, such as shorter wait times, shorter lines, and lower total vehicle emissions. Thorough testing and validation techniques, including unit testing, visual inspection, and comparison KPI analysis, supported the simulation outputs. In addition to learning the best phase-switching techniques, the agent showed promise for future expansion to bigger, more intricate traffic networks.  
  
Furthermore, this solution can be modified for next smart city implementations or scholarly research thanks to the modular and configurable codebase, which features a clear file structure, CLI-based interaction, and reusable logic. The solution's legitimacy and potential for use outside of this project are enhanced by the methodical documenting of the development process, risk management, and validation procedures.  
  
In summary, the work demonstrated a proof-of-concept for AI-powered intelligent traffic systems in addition to meeting academic goals. It lays the groundwork for future developments in traffic control, where learning-based, data-driven strategies might be crucial in establishing effective and sustainable urban mobility.

# References

Amazon. (2022). *AWS*. Retrieved from What is Reinforcement Learning?: https://aws.amazon.com/what-is/reinforcement-learning/

Bank, W. (2022, June 21). *Toward Great Dhaka: A New Urban Development Paradigm Eastward.* Retrieved from Integrated Corridor Management Dhaka North Project (P177702): https://documents1.worldbank.org/curated/en/099300006232222667/pdf/P1777020b38220040a9490dbfb31f656f5.pdf?utm\_source=chatgpt.com

Center, G. A. (2017). *Simulation of Urban MObility*. Retrieved from SUMO: https://eclipse.dev/sumo/

codesignal. (2025). *balancing-exploration-and-exploitation-with-epsilon-greedy-strategy*. Retrieved from game-on-integrating-rl-agents-with-environments: https://codesignal.com/learn/courses/game-on-integrating-rl-agents-with-environments/lessons/balancing-exploration-and-exploitation-with-epsilon-greedy-strategy

Geeksforgeeks. (2025, February 24). *what-is-reinforcement-learning*. Retrieved from Reinforcement Learning: https://www.geeksforgeeks.org/machine-learning/what-is-reinforcement-learning/

Manjila, C. (2024, November 29). *What is Unit Testing? - A Complete Guide*. Retrieved from headspin: https://www.headspin.io/blog/unit-testing-guide

Michael Behrisch. (2025, July 22). *Eclipse SUMO™*. Retrieved from Eclipse Foundation: https://projects.eclipse.org/projects/automotive.sumo

Moon, L. (2019, September 25). *Products and news*. Retrieved from Atlassian: https://www.atlassian.com/blog/trello/what-is-a-workflow-and-why-do-you-need-it

Nations, U. (2025). *Make cities and human settlements inclusive, safe, resilient and sustainable*. Retrieved from The SDG Report: https://sdgs.un.org/goals/goal11?utm\_source=chatgpt.com

Rehkopf, M. (n.d.). *Scrum sprints*. Retrieved from Atlassian: https://www.atlassian.com/agile/scrum/sprints

# Appendix A – Code Manifest

The following table outlines the major files used in the system, their respective locations, and their role in the project. This manifest supports traceability, clarity, and ease of navigation for reviewers, collaborators, or future developers.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File Name** | |  | | --- | |  |   **Path** | **Status** | |  | | --- | |  |  |  | | --- | | **Description/**  **Contribution** | |
| dqn\_agent.py | /ai/ | Created | Defines the Deep Q-Network (DQN) agent with replay memory, target network, and training logic. |
| train\_rl\_agent.py | /ai/ | Created | Main training script to run the SUMO simulation and train the RL agent using TraCI interface. |
| main.py | /src/ | Modified | Legacy controller entry point, replaced with  train\_rl\_agent.py  for RL-driven execution. |
| dhaka\_map.net.xml | /simulation/ | Created | SUMO network file defining road topology, junctions, and edges based on Dhaka city layout. |
| routes.xml | /simulation/ | Modified | Defines vehicle routes, flows, and traffic patterns. |
| config.sumocfg | /simulation/ | Created | SUMO configuration file linking all other simulation components together. |
| tls.xml | /simulation/ | Modified | Implements a baseline fixed-time traffic signal controller for comparison with RL agent. |
| controller\_rule\_based.py | /scripts/ | Created | Extracts and visualizes performance metrics for RL vs. baseline systems. |
| compare\_kpis\_with\_graphs.py | /scripts/ | Created | Output files from SUMO simulation containing performance, environmental, and vehicle-level logs. |
| tripinfo.xml,  edgeData.xml,  emissions.xml | /out/ | Generated | Output files from SUMO simulation containing performance, environmental, and vehicle-level logs. |
| vehtypes.xml | /config/ | Created | Defines types of vehicles used in simulation (car, truck, etc.). |
| flows.yaml | /config/ | Created | Defines adjustable traffic flow rates and configurations per edge. |
| README.md | /project-root/ | Created | Contains usage instructions, setup steps, and contribution guidelines. |