aogokul_b24@tx.vjti.ac.in
+91 82912 54798

# TrafIQ :
## Smart Traffic Signal Optimization

PROJECTX 2025

PREPARED BY

OJAS GOKUL ALAI
SECOND YEAR, B.TECH [TEXTILE TECHNOLOGY]

VJTI, MUMBAI

# ABSTRACT

Traffic congestion has become a defining challenge of urban life in India, with cities routinely paralysed by gridlocked intersections, prolonged wait times, and rising levels of fuel consumption and air pollution. At the heart of this dysfunction lies an antiquated system of fixed-timing traffic signals — blind to the real-time movement of vehicles, insensitive to fluctuating demand, and incapable of adapting to the dynamic rhythm of urban mobility.

This static approach results not only in inefficiency, but in tangible social and environmental costs such as:

- increased carbon emissions
- wasted fuel
- delayed emergency responses
- mounting commuter frustration.

Despite the rapid growth of cities and vehicles, traffic management systems have remained stubbornly unresponsive, unable to match the pace of urban transformation.

This project proposes to confront this issue by exploring an intelligent, adaptive traffic signal system capable of learning from real-time traffic conditions and responding dynamically. While the technical solution is central, the broader vision is one of systemic reform — a shift toward more sustainable, responsive, and humane urban infrastructure

# TABLE OF CONTENTS


***Contents*** ***Page no.***

*1. Preface.....04*

- *About me*
- *Project Description*
- *Motivation*
- *Key Objectives*

*2. Methodology.....06*

- *Execution Overview*

*I.* ***Simulation Environment Setup.....06***
*Road Network Construction*
*Traffic Demand Generation*
*Demand Calibration*
*Real-time Interaction with TraCI ( Traffic Control Interface)*
*Signal Phase Status*

***II. Reinforcement Learning.....09***
*State Representation*
*Action Space*
*Reward Function*
*Action Selection Strategy (for Q-learning)*

***III. Training Reinforcement Learning Agents.....11***
*Single Intersection Training*
*Multiple Intersections Training using MARL*

***IV. Evaluation and Benchmark.....15***
*Evaluation Setup*
*Performance Metrics*
*Visualisation and analysis*

***V. Hardware prototype and real world extension.....16***
*Hardware*
*Workflow*

*3. Project Workflow.....17*

*4. References.....18*

# PREFACE

## ABOUT ME

I, Ojas Gokul Alai, am an incoming second-year undergraduate at Veermata Jijabai Technological Institute, pursuing a B.Tech in Textile Technology. I have cultivated a keen interest in machine learning, image processing, and probability—areas that captivate me due to my deep curiosity about the mechanisms of intelligent systems and the principles that govern how machines learn to address complex challenges.

My technical foundation includes proficiency in C, C++, and Python, complemented by experience in web development. As I continue to deepen my knowledge in artificial intelligence and machine learning, I am eager to engage with and contribute meaningfully to the open-source community.

## PROJECT DESCRIPTION

India's urban traffic is characterized by severe congestion, prolonged wait times, and elevated pollution levels—issues largely stemming from fixed-timing traffic signals that fail to respond to real-time conditions. This project aims to tackle these challenges by developing an adaptive traffic signal control system leveraging Genetic Algorithms, Q-Learning, and Proximal Policy Optimization (PPO), implemented within the SUMO traffic simulation environment.To simulate realistic and dynamic traffic conditions, real-time vehicle counts and flow data will be generated using YOLO-based object detection on video feeds. This data will be integrated with the SUMO environment via TraCI to enable responsive signal control. The system will optimize traffic light timings, including a multi-agent architecture to manage multiple intersections collaboratively. Performance will be rigorously benchmarked using the RESCO standard.

To demonstrate real-world applicability, a hardware prototype will be built using IR sensors, a Raspberry Pi, and Arduino-controlled LEDs, showcasing the system's practical deployment capabilities.

## MOTIVATION FOR THE PROJECT

As someone who has experienced the daily frustration of traffic congestion in Indian cities, I'm motivated to tackle the inefficiencies caused by outdated, fixed-timing signals. Through this project, I aim to apply my passion for machine learning to develop an adaptive traffic signal system that responds to real-time conditions, helping reduce delays, lower emissions, and make urban commuting smoother and more sustainable.

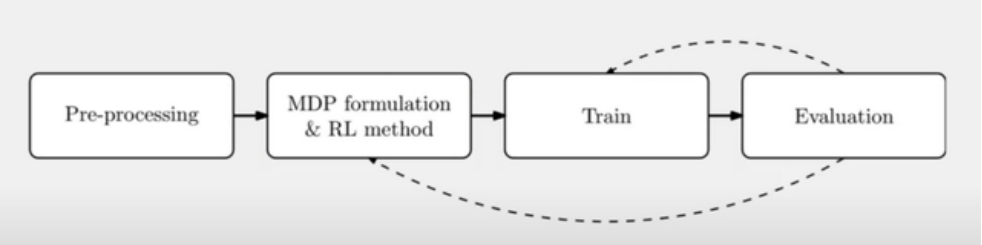
## KEY OBJECTIVES

- Develop an adaptive traffic signal control system using reinforcement learning (RL) within the SUMO simulation environment using TraCI api and YOLOv5 model.
- Demonstrate significant improvements over traditional fixed-time or rule-based traffic signal strategies.
- Design realistic state, action, and reward structures incorporating queue length, waiting time, throughput, fuel consumption, and $CO_2$ emissions.
- Compare the performance of various RL algorithms such as Genetic Algorithm, Q-Learning, DQN, Independent Q-Learning, IDQN, and IPPO algorithms for both single and multiple intersections using RESCO benchmark parameters
- Prototype real-world deployment by integrating sensor feeds (IR sensors, Raspberry Pi) and controlling physical traffic lights (LEDs, Arduino)

# METHODOLOGY

## EXECUTION OVERVIEW

The TrafIQ pipeline begins by constructing a high-fidelity traffic simulation in SUMO, calibrated to real-world conditions via YOLO-based video analysis. YOLO is first used offline to extract vehicle counts, lane-wise flow rates, and turning ratios from recorded traffic footage; these parameters inform SUMO route files and traffic demand models. During simulation, a Python controller connected via TraCI fetches state data (vehicle positions, speeds, queues, emissions) at one-second intervals. Deep RL agents then consume this state to select optimal signal phases, which are applied back into SUMO to influence traffic. We validate and benchmark algorithms on a single intersection—including rule-based, GA, Q-Learning, PPO, and DQN controllers—and use the RESCO framework for standardized performance comparison. Subsequently, TrafIQ scales to multi-intersection networks on real city maps using MARL techniques such as IDQL, IPPO. Finally, a hardware prototype leveraging Raspberry Pi, IR sensors, and LEDs demonstrates real-time traffic signal control.




# 1. TRAFFIC SIMULATION ENVIRONMENT SETUP

## 1. ROAD NETWORK CONSTRUCTION

***Single or Multiple Intersections:***

Build the target road network in SUMO, either manually (using netedit for a 4-way intersection/group of intersections) or by importing OpenStreetMap data for larger/city-scale maps.

RESCO's real-world maps can be used for SUMO simulations.

## 2. TRAFFIC DEMAND GENERATION

***Synthetic Demand:***
Generate initial traffic flows using statistical models (e.g., Weibull distributions) to simulate realistic arrival rates and turning movements.
***Real-World Calibration:***
Capture real-world intersection videos at fixed intervals, apply YOLOv5 for vehicle detection and object tracking (eg. DeepSort) to extract vehicles per unit time, lane wise turning movements and flow distribution.

## 3.DEMAND CALIBRATION

***Data Integration:***
Convert YOLO-based empirical counts into lane-level flow rates and turning ratios.
***Route File Adjustment:***
Iteratively adjust the SUMO route file so that simulated vehicle counts, turning rates, and flow patterns closely match the observed data.

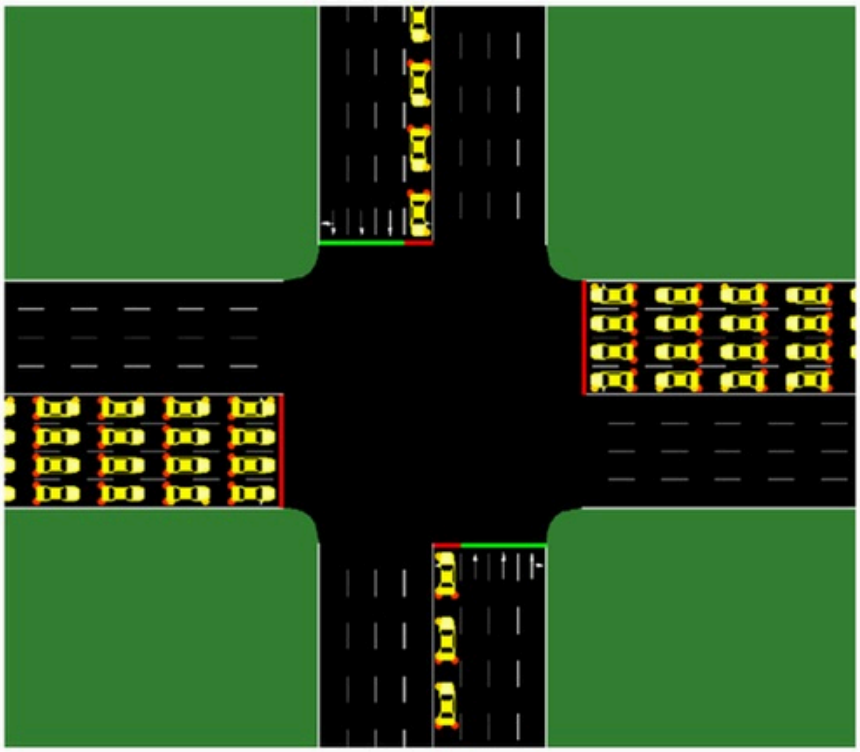
## 4.REAL-TIME INTERACTION WITH TRACI (TRAFFIC CONTROL INTERFACE)

TraCI allows external Python control of simulation parameters. Enables dynamic interaction with traffic lights, vehicle data, and environment states.
Sample outputs accessible through TraCI:

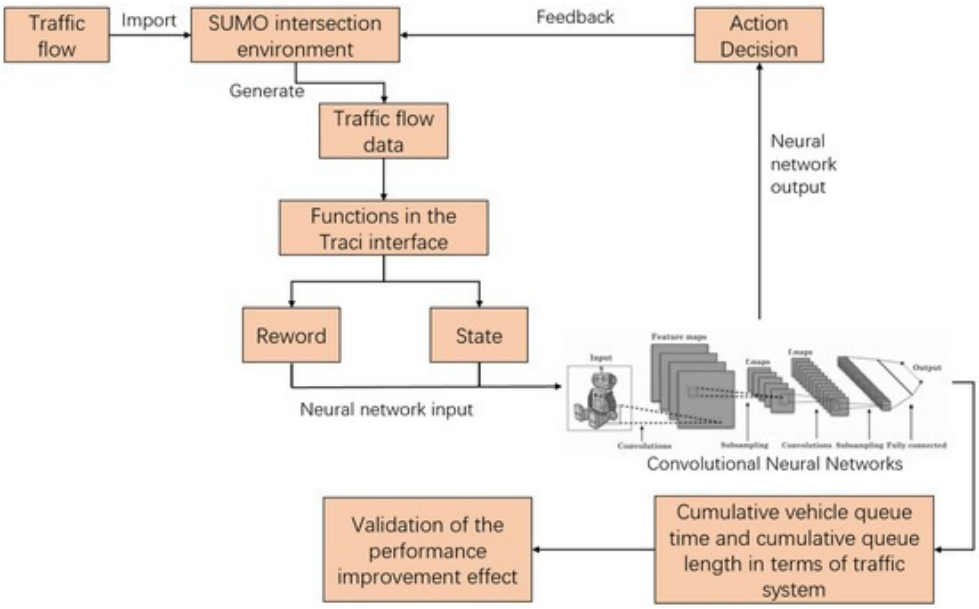
- Vehicle speeds, halting status, waiting time
- Lane occupancy
- Signal phase status

## 5. SIGNAL PHASE STATUS

***YOLO for Real-World Mapping:***
Use YOLO only for real-world video analysis and calibration, not for extracting simulation states.
***Numerical State Representation:***
Convert extracted data into numerical vectors (e.g., queue lengths, binary occupancy, velocities) for RL agent input.



SUMO INTERFACE




TECHNOLOGY ROADMAP

# 2. REINFORCEMENT LEARNING

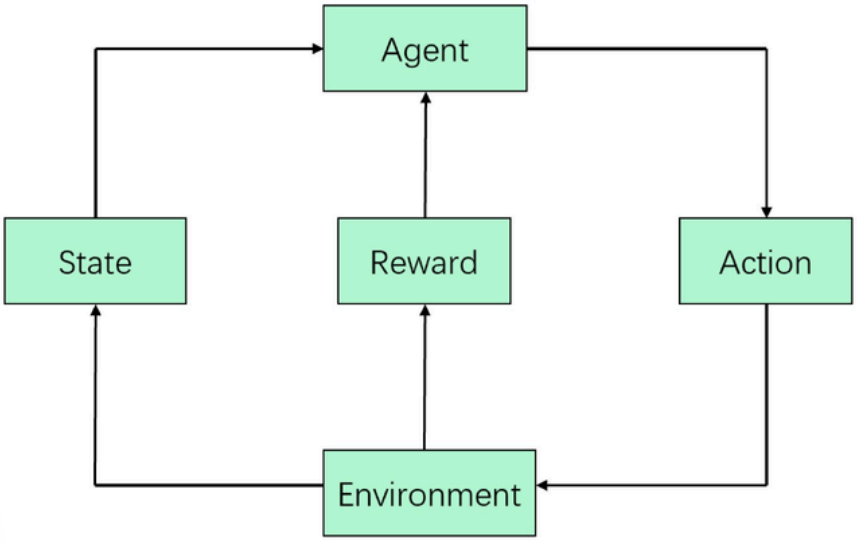
Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards over time.
RL can be mathematically modeled as a Markov Decision Process (MDP), which is well defined by five-tuples (S, A, T, R, γ) having the following meanings:

State (S): Possible set of state space. S = s1, s2, …, a finite of states. The state is a mathematical description of the environment that is relevant and informative to the decision to be made.

Action (A): Possible set of action space. A = a1, a2, … a finite set of actions. Action is the decision made by the controller in terms of how to control the environment. The environment is the target of control.

Transition function (T): which defines the probability of moving between the different environment states. It's predicting how the environment will evolve if we take action at state st.

Reward Function (R): The reward function predicts the immediate rewards which at discrete time step t selects an action at given the state st, and that maximizes the cumulative future discounted reward on return




## 1. STATE REPRESENTATION

Each agent (intersection) observes the following at each time step t, encapsulated in a vector state st usually represented as a matrix:
**st=[QN,QE,QS,QW;VN,VE,VS,VW;P1,…,PK;te]**

i. Queue Length per Lane

Qtd=Number of halting vehicles in direction d∈{N,W,S,E} Q_t^d = {Number of halting vehicles in direction }d {{N,W,S,E}}Qtd=Number of halting vehicles in direction d∈{N,W,S,E}

ii. Normalized Mean Speed per Lane

Vd is the average speed on approach d, scaled between 0 and 1.

4 normalized speeds [VN,VE,VS,VW]

iii. Pk is a simple one-hot flag: it's 1 for the currently green phase k, 0 otherwise.

iv. te is the seconds elapsed since the last phase change.

Hence our st vector is a (4 + 4 + K + 1) dimensional column vector which fully captures the traffic and signal state at time 't'.

Scalability Note: State can be extended to a lane-wise formulation (e.g.4 lines per direction gives 16 queue entries), allowing the agent to capture finer traffic features such as asymmetric lane congestion or turn-lane delays. This is especially useful for deep RL models like DQN or PPO, and future iterations will consider this enhanced encoding.

## 2. ACTION SPACE

It is a discrete action. Change or hold the current signal phase, or select the next phase in a cycle. The agent selects a discrete action at from:

A={NS_straight,NS_left,EW_straight,EW_left,All_Red}

## 3. REWARD FUNCTION:

We use a multi-objective reward shaping strategy to balance efficiency, fairness, and sustainability:

$$r = -w_1 d\sum Qtd - w_2 d\sum Wtd - w_3 St - w_4 Et$$

1

- ∑dQtd is the total queue length over all approaches.
- ∑dWtd\∑dWtd is the aggregate waiting time of all vehicles.
- St is the number of vehicles currently stopped (speed = 0).
- Et is the $CO_2$ emissions produced in this step (from SUMO's emission model).
-

The weights of these parameters are tunable via grid search or reward-sensitivity experiments.

### 4. ACTION SELECTION STRATEGY (FOR Q-LEARNING)

To balance exploration and exploitation during RL training, we adopt the ε-greedy strategy: at each step, the agent selects the action:

$$a = \begin{cases} \arg\max_a Q(s,a), 1-\varepsilon \\ random(A), \varepsilon \end{cases}$$

This prevents overfitting by allowing the agent to occasionally explore new actions instead of always choosing the current best.

## 3. TRAINING REINFORCEMENT LEARNING AGENTS

This phase focuses on developing, training, and evaluating different RL agents for traffic signal control under both single and multi-intersection scenarios.

### SINGLE INTERSECTION TRAINING

In the single intersection setup, the environment consists of one isolated 4-way junction. Agents are trained to optimize traffic signal decisions using various algorithms.
Algorithms:

***1. Rule Based Algorithm:***
Operates on fixed-time or threshold-triggered conditions. Green phases are cycled or triggered based on conditions like queue length exceeding a limit:

if Qi > Qthreshold ⇒ trigger_phase_i

Simple and computationally cheap, but fails to adapt to complex or dynamic traffic conditions.

***2. Genetic Algorithm (GA)***:
GA encodes signal phase durations as chromosomes, where C = [g1,g2...gn], where each gi ∈ [t_min, t_max]

Evolves the population using selection, crossover, and mutation, guided by the fitness function: The fitness function uses a similar equation as the reward function including parameters, queue length, delay, wait time, emissions and number of vehicles passed. This is well-suited for optimizing fixed scheduling policies in non-learning environments.

## ***3. Q-Learning (QL):***

Q-Learning is a value based algorithm which uses a look-up table for each state-action pair, it learns an optimal policy by experiencing consequences of its actions using the bellman-update rule.

At each step, the agent observes the current state (e.g., queue sizes), selects an action (e.g., change signal phase), receives a reward (like reduced waiting time), and updates the Q-value for that state-action pair using the formula:
where,
α = learning rate
γ = discount rate
r = reward
Q(s,a) = Q-value for state s and action a

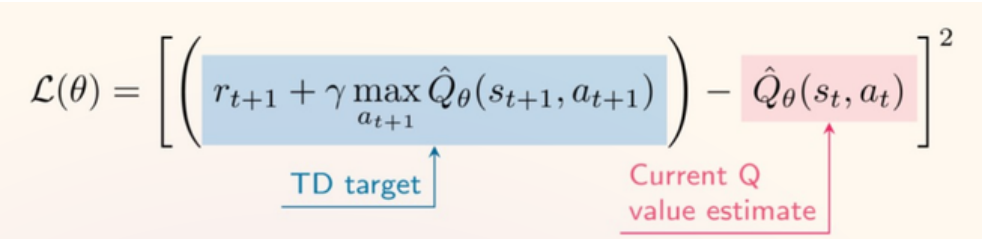
$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right)$$

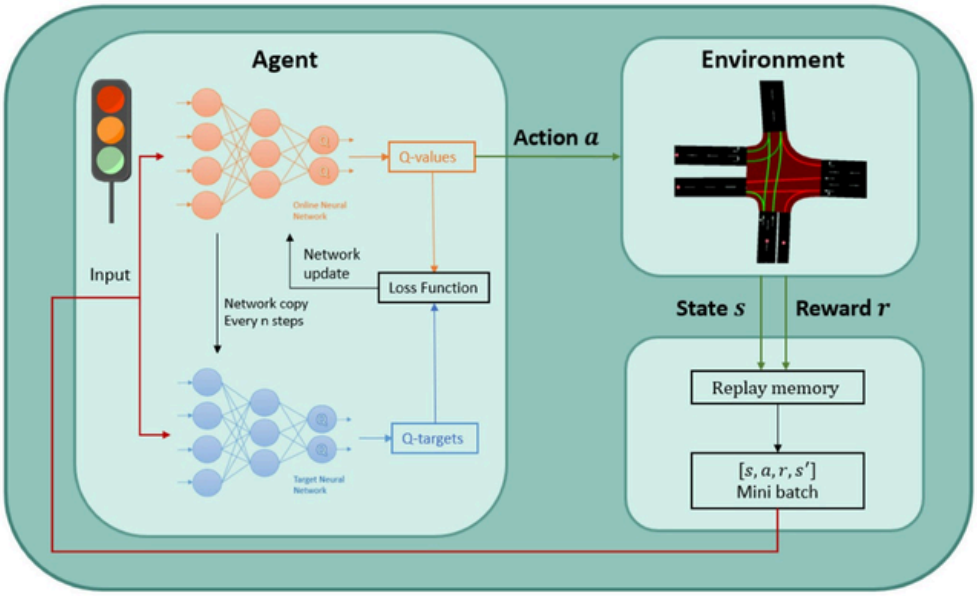
## ***4. Deep Q-Network (DQN):***

DQN combines RL with Neural Networks, which allows agent to handle complex, high-dimensional state spaces such as long queue lengths and waiting times which QL model cannot manage

At each time step, the agent observes the current state, selects a signal phase (action) using the ε-Greedy Policy and receives a reward based on traffic efficiency such as negative waiting time. The Q-Value Function is approximated by a neural network, which updates itself using experience replay and a target network for stability.

***Loss Function:***






## MULTIPLE INTERSECTIONS TRAINING USING MARL

In multi-intersection traffic control, intersections act as independent agents or coordinated learning units.

### ***1. Independent Q-Learning (IQL):***

- Each intersection maintains a local Q-table, mapping discrete traffic states to phase actions.

- Each intersection is controlled by an independent agent, and each agent updates its own Q-values based on its local observations and rewards, treating other intersections as part of the environment.

Qi(si,ai): Q value for agent i in state si taking action ai;

Each intersection agent learns its own policy independently, without explicit coordination or communication, which simplifies the problem but can lead to instability in highly dynamic multi-agent environment.

the updated QL equation for multiple intesections is,

$$Q_i(s_i, a_i) \leftarrow Q_i(s_i, a_i) + \alpha \left[ r_i + \gamma \max_{a_i'} Q_i(s_i', a_i') - Q_i(s_i, a_i) \right]$$

## 2. ***Independent Deep Q-Network:***

Each intersection is controlled by its own agent, each maintaining and training a separate a deep Q-Network. The agents operate independently observing their local environment, observing its particular state and its action.
ε-greedy in IDQN: Each agent independently explores (ε) or exploits (1-ε) its local Q-values.

***Q-Network:***

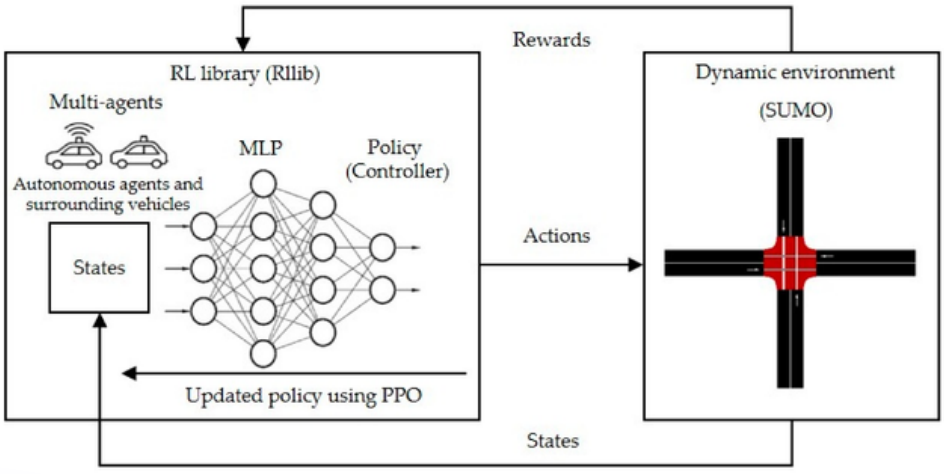
Each agent uses a neural network to estimate the Q-value function Qi (si, ai; θi), where θi are network parameters for agent i. The procedure is similar to DQN acting independently. Each agent use the ε-Greedy Policy and stores the reward in the agent's policy buffer.

Agents do not communicate or coordinate directly; each learns its own policy based on local experience. This makes IDQN scalable and easy to deploy in large networks, though it may not capture complex interactions between intersection.This minimises congestion metrics improving overall traffic flow.

## 3. ***Independent Proximal Policy Optimization (IPPO):***

A MARL System with independent agents which observe the current states, are fed into a multi-layer perceptron representing the policy for each agent.

The policy (actor) and value function (critic) are updated locally using PPO's clipped objective provides more stable and generalized training due to gradient-based updates and policy regularization.




For Sparse Traffic: Tabular Q-Learning/DQN are effective,
For Dense Traffic: PPO/IPPO are effective due to generalization.
Other Algorithms which can be explored for multiple intesections are:

1. MaxRL/Model-Based RL
2. Actor-Critic RL

# 4. EVALUATION AND BENCHMARK

Once the RL agents have been trained, their performance is rigorously evaluated using standardized simulation environments and comparative benchmarks

## EVALUATION SETUP

Agents are tested on the RESCO (Reinforcement learning-based Signal COntrol) benchmark environments:

- Single Intersection
- Corridor Network (3-5 connected intersections)
- Grid Network (extension to city maps)
- Each trained policy is compared against rule-based, fixed-time, and max-pressure control baselines.
- Traffic demand patterns are varied (low, medium, high) to assess robustness.

## PERFORMANCE METRICS

- Average Waiting Time: Mean time vehicles spend stationary across all lanes.
- Queue Length: Total number of halted vehicles per time step.
- Throughput: Number of vehicles that successfully pass through intersections.
- $CO_2$ Emissions: Estimateper-vehicle emissions using TraCI tools.
- Fairness: Distribution of green time across different directions or lanes.

## VISUALIZATION & ANALYSIS

To interpret the performances of various RL policies, python libraries like matplotlib, seaborn can be used to understand agent's behaviour, convergence and robustness. Visualisations include:

- Reward vs Episode Curve: Shows cumulative reward over training episodes to monitor convergence.
- Loss vs Epoch Curve: Plots policy loss, value function loss, and entropy loss to assess training stability.
- Average Waiting Time vs Time: Evaluates real-time agent performance during evaluation.

# 5. HARDWARE PROTOTYPE AND REAL-WORLD EXTENSION

If time permits, we will demonstrate the feasibility of TrafIQ using simple RL-based traffic controller using Arduino/Raspberry Pi and IR sensors.

***Sensors:***

IR proximity sensors on each incoming lane detect vehicle presence.
Integrate YOLO and IR Sensors synergestically, YOLO for Vehicle Counting and IR sensors for queue validation.

***Controller:***

A Raspberry Pi hosts a pre-trained RL model (DQN or PPO) using TensorFlow Lite for lightweight, real-time inference.

***Actuation:***

Arduino reads IR data and handles simple rule-based logic or relay control. Based on sensor inputs and model output, GPIO pins activate LEDs representing traffic lights.

***Workflow:***

1. IR sensors send binary presence signals to the controller.
2. Raspberry Pi runs the trained model and selects the optimal signal phase.
3. The signal phase is transmitted to the Arduino, or the Pi directly controls the LEDs.
4. Timing rules (e.g., yellow buffer, minimum green duration) are enforced.

# WORKFLOW

## WEEK 1-2: TRAFFIC SIMULATION SETUP + REAL-WORLD CALIBRATION

- Tools: SUMO, NetEdit, YOLOv5, OpenCV, OpenStreetMap (OSM)
- Build single and multi-intersection maps using SUMO NetEdit or OSM imports and define .net.xml, .rou.xml, .sumocfg files.
- Collect real-world intersection video data; apply YOLOv5 with object tracking to define our road network, number of vehicles in a particular lane per unit time.
- Estimate vehicle flow rates and turning ratios from frame-wise tracking.
- Calibrate traffic demand in SUMO to match real-world statistics.
- Validate simulated metrics (queue, wait time) against video observations

## WEEK 3: RL ENVIRONMENT & INTERFACE DEFINITION

- Libraries: TraCI, NumPy, Pandas, Matplotlib, OpenAI Gym (if custom env needed)
- Design state space: queue length, vehicle count, speed, signal phase, elapsed time.
- Define action space: discrete phase configurations.
- Formulate reward function using a weighted composite metric.
- Interface SUMO with Python via TraCI to extract real-time state data and apply actions.
- Prepare RL-compatible numerical state vectors.

## WEEK 4-6: RL AGENT TRAINING

- Libraries: PyTorch, Stable-Baselines3, CityFlowER, Matplotlib,

### SINGLE INTERSECTION

- Train baseline models:
- Rule-based controller (static logic)
- Genetic Algorithm (SciPy)
- Train RL models:
- Q-Learning (tabular)
- Deep Q-Network (DQN) using Stable-Baselines3
- Track metrics: episode reward, learning loss, queue reduction

### MULTIPLE INTERSECTION

- Configure multi-agent RL environment with corridor/grid layout.
- Train MARL agents: IDQN and IPPO algorithms.
- Coordinate agents using decentralized policies.
- Use CityFlowER to scale to larger maps if needed.
- Final selection of RL Policies will be decided iteratively duiring the project.

### WEEK 7–8 : EVALUATION & BENCHMARKING

- Libraries: RESCO Benchmark Tools, Seaborn, TensorBoard, Plotly
- Compare models against rule-based, fixed-time, and max-pressure baselines.
- Record and visualize: Average waiting time, $CO_2$ emissions, Queue lengths and throughput
- Create: Reward vs. Episode graphs, Emission heatmaps, radar plots, baseline bar charts.
- Buffer for Agent Improvement (half week):

1. Refine best-performing agents (hyperparameters, policy tweaks)
2. Ensure convergence and stability before formal evaluation.

- Final benchmarking phase.

### WEEK 8 ONWARDS:

- YOLO is deployed in real-time on Raspberry Pi for live video inference. Queue data is sent to traffic light using IR sensor verification and RL signal logic via LEDs. This closes the simulation-to-reality loop and demonstrates practical deployment.
- Final Documentation,

# REFERENCES


https://sumo.dlr.de/docs/index.html
https://www.mdpi.com/2079-9292/13/1/198
https://www.kaggle.com/code/alexisbcook/deep-reinforcement-learning
https://people.engr.tamu.edu/guni/papers/NeurIPS-signals.pdf
https://www.geeksforgeeks.org/machine-learning/yolov5-object-tracker-in-videos/
https://www.youtube.com/watch?v=IUiKAD6cuTA
https://www.youtube.com/watch?v=VnpRp7ZglfA