TrafIQ: Smart Traffic Signal Optimization System

Sofia Abidi

B. Tech. Computer Engineering VJTI

June 2025

About Me

My name is Sofia Abidi. I am a soon-to-be SY Computer Engineer in Veermata Jijabai Technological Institute. I have a deep interest in competitive programming and data structures. Curious to explore the domains of AI/ML, NLP and Data Science. I am proficient in languages C++, Python, Javascript and, Java. It is easy for me to grasp new programming languages and get comfortable with it. I am particularly excited to apply my skills to real-world projects that challenge me. Project-X gives me the perfect opportunity for the same.

Motivation for this project

Traffic management issues is something that personally has affected everyone in urban cities like Mumbai, Bangalore. I always wanted to work on a project that is both meaningful and impactful—one that could genuinely help people in their daily lives. That is why “TrafIQ” stood out to me.

Also, Reinforcement Learning in particular has always intrigued me the most out of all machine learning models. The idea that an agent can learn optimal behaviour by interacting with its environment and improving over time with rewards, penalties is not only fascinating but also well-suited to solving dynamic problems like that of traffic signal control.

Project Overview & Approach

Introduction

Traffic signal optimization is a technique for intelligently managing traffic infrastructure like intersections and roadways using advanced technologies, aiming to enhance the efficiency, safety, and convenience of road transportation. It conforms to the direction of the intelligent, green, and sustainable development of modern cities. Smart transportation combines technologies such as the Internet of Things, cloud computing, big data, and artificial intelligence to gather traffic information through high technology and support the control of traffic management.

Most traditional traffic signal optimization control systems use traditional fixed timing schemes, which cannot be adjusted according to the real-time status of intersections and are prone to traffic congestion if the traffic flow is too high.

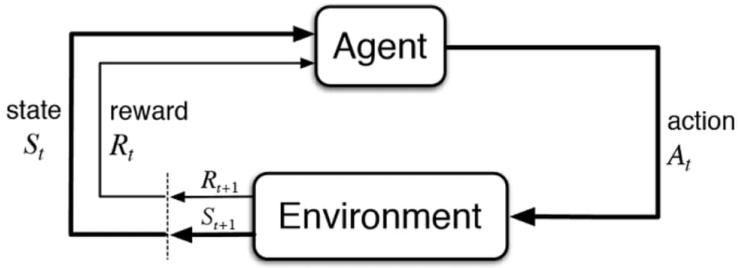
The following manually designed attempts have been made in the past to resolve the issue:

1) Adaptive Traffic Signal Control (ATSC) that set different signals according to different geographical areas,
2) Pressurized routing algorithms,
3) Greedy ideas and modelling two-dimensional automata matrices for vehicles
4) Take holiday factors into account

With growing population and new developments in transportation system, transportation has evolved into intelligent systems known as ITS. Unlike the methods mentioned above, intelligent traffic signal optimization control methods are able to adjust the strategy according to real-time road conditions. Although traffic signal control is a difficult and complicated optimization problem, many intelligent algorithms like the fuzzy logic, evolutionary algorithms and RL have also been used to address this issue.

Reinforcement Learning

The Traffic signal control is basically a matter of sequential decision-making. Hence, it is particularly suited to the Markov Decision Process (MDP) and RL framework where an agent needs to learn from trial and error method through interaction with the environment. MDPs are a mathematical model for sequential decision making, which include a set of interactive objects such as environment and agent. RL includes five model elements: state, action, reward, strategy, and value function. The objective of the agent is to learn an optimal action selection policy, so that the discounted cumulative reward is maximized through repeated interactions with the environment. In simpler words, designing a probabilistic policy of what actions to take given a State 's' to maximize chance of getting rewards. The discount factor (ranging from 0 to 1) is incorporated to prioritize immediate rewards over those received in the future, which are progressively discounted.



To reiterate, the agent's action selection is modelled as a map called policy:

$$\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$$
$$\pi(a, s) = \Pr(A_t = a \mid S_t = s)$$

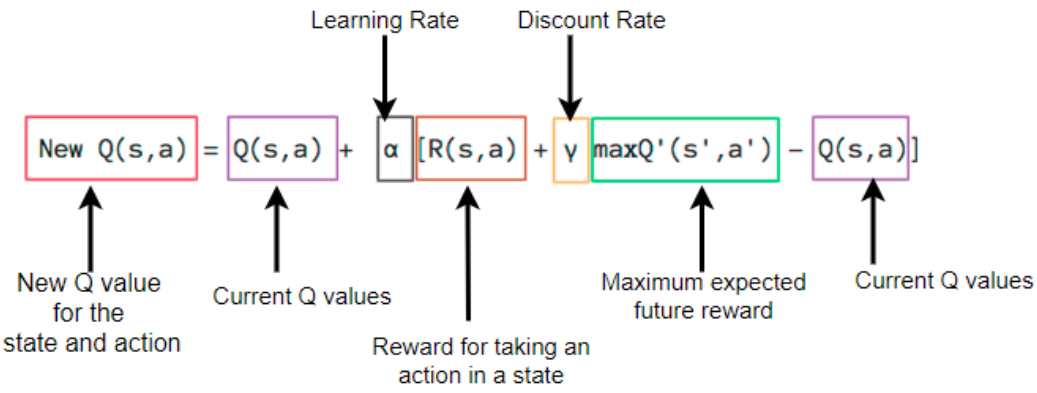
The 'Value Function' of state 's' given policy π is the Expectation of how much reward the agent will get in the future if it starts in that state and enacts that policy.

$$V_\pi(s) = \mathbb{E}\left(\sum_t \gamma^t r_t \mid s_0 = s\right)$$

'P' the state transition probability function can be described as the probability that will lead to state s_{t+1} at time $t + 1$ given action a_t in state s_t at time t

$$\boldsymbol{P} = pr(s_{t+1}|s_t, a_t), a_t \in \boldsymbol{A}, s_t \in \boldsymbol{S}$$

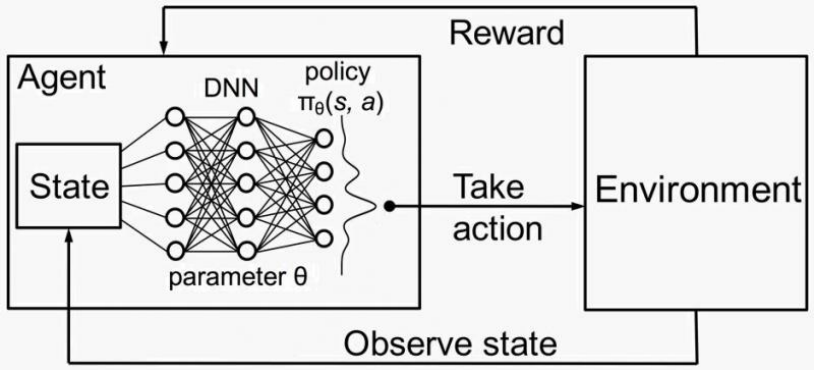
Q-Learning combines the Policy and Value function the agent can learn assuming it does the best possible actions in the future.



The model then picks the action with the best quality at any step.

It is important to note that the **complexity** of using RL in traffic signal control **increases exponentially when state and action spaces expand.** To overcome this barrier, deep learning has recently attracted considerable attention and has been efficiently integrated with RL approaches to yield DRL.

Deep Reinforcement Learning



Deep Reinforcement Learning integrates the perceptual ability of deep learning and the decision-making ability of reinforcement learning to first explore and optimize the system objectives through reinforcement learning, and then, second, to solve the optimization control strategy problem through deep learning algorithms.

In DRL, a set of parameters (e.g. weights and biases of neural network) θ parametrize the policy π, denoted by π_θ ,which gives the probability distribution over actions given state. G function is the cumulative sum of discounted reward for all the iterations in an episode. Iterations referring to the individual time steps or action-decision cycles that occur within a single episode. If an episode consists of *T* iterations then the value of *G* at t^{th} iteration (G_t) is defined as follows:

$$G_t = \sum_{i=t}^{T} \gamma^{i-t} R_i$$

$$J(\theta) = E_{(\pi,\theta)}[G] = \sum_{t=0}^{T} \pi_\theta(a_t|s_t) G_t$$

The ultimate goal is that the parameter θ needs to be updated accordingly so that $J(\theta)$ is maximized.

Single Intersection Approach

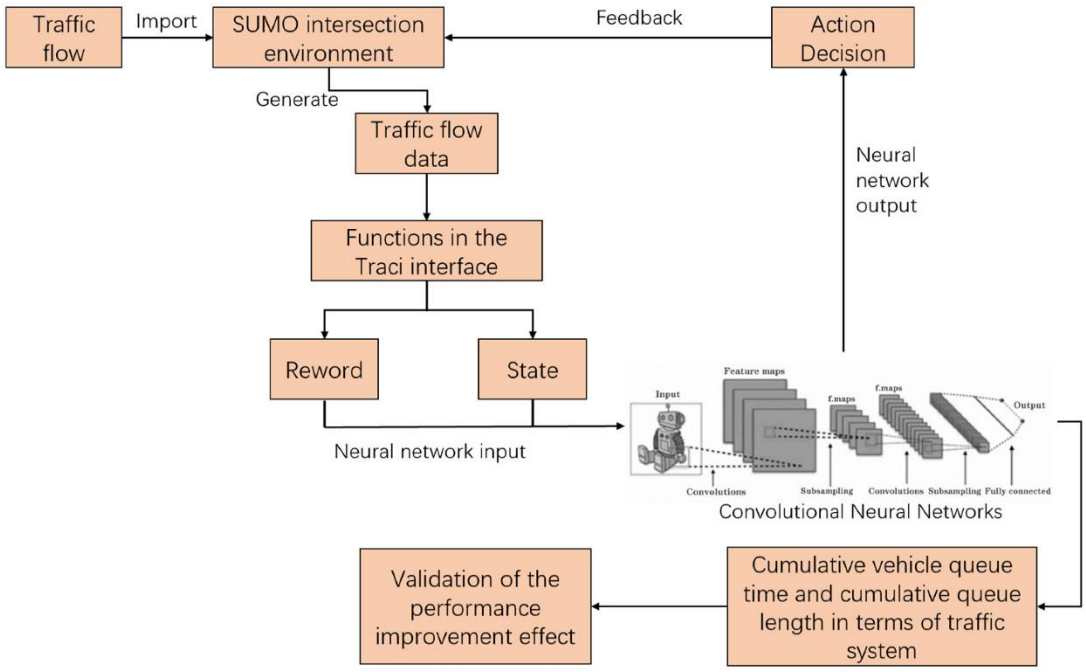


Figure: Uses CNN as the deep learning model

The established single-intersection training scenario is implanted into SUMO simulation software to form the environment. The traffic flow data are processed using the functions under the vehicle module in the Traci interface to calculate the state and reward, the Q-value is calculated from the reward, and the state and Q-value are passed into the neural network for training. The neural network

will use the output generated by each turn prediction, i.e., action, for decision making until the specified training turn is reached, and it will feed back to the SUMO intersection environment to change the phase of the signal in SUMO. In addition, the queue length and queue time of vehicles in the traffic system for each round will be counted.

When the neural network completes the prescribed training rounds, the queue length and queue time of vehicles in the traffic system for each round are compared to verify the **performance** improvement in the algorithm.

Instead of learning from only the most recent experience, DQN stores each experience into a buffer or **experience pool**.

Each experience is a 5-tuple:

- Current state: s_t
- Action taken: a_t
- Reward received: r
- Next state: s_{t+1}
- Done flag (whether episode is over): True / False

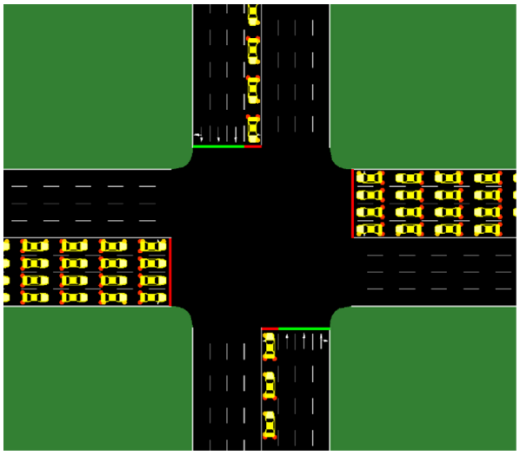
Then during training, random samples (mini-batches) are drawn from this pool. This breaks the strong correlation between consecutive experiences, so that the neural network sees more diverse, **i.i.d. data** (independent and identically distributed). This helps the network converge faster.

In DQN, we build **two neural networks having the same structure**:

- **Main (or online) network**: with parameters θ updates its weights every step (via backpropagation)
- **Target network**: with parameters θ′ updates less frequently, like every few hundred steps, by copying weights from the main network

$$\mathrm{L} = [r + \gamma \max Q(s_{t+1}, a_{t+1}; \theta`) - Q(s, a; \theta)]^2$$

The loss function can help the agent learn more accurate *Q* values, promote their exploratory behavior, and avoid the overfitting of neural networks.



Consider the following image,

State:

To give the neural network a structured input, the intersection is converted into **two matrices**:

(a) Vehicle Location Matrix (16×10)

- Each of the 4 directions has 4 approach lanes (4×4 = 16 lanes total).
- Each lane is split into 10 equal cells from the stop line backward.
- So, you get a 16 rows×10 columns grid (like a heatmap of lane occupancy).
- If a vehicle is present in a cell → mark 1, otherwise → 0.

(b) Speed Matrix (16×10)

- Same shape (16×10), but now each cell stores the average speed of vehicles in that segment.
- To help training, speeds are normalized (divided by the max speed limit).

This approach treats the vehicle positions as equally important. We may improve on this by assigning higher weights to vehicles nearer to the stop line.

Action:

Causes the signal of a set of lanes to turn green in one cycle. All possible actions are defined as a set of phases:

$A=\{EW,EWL,SN,SNL\}$

where *EW* is the east–west-direction-driving vehicles that can go straight and turn right; *EWL* is the east–west-direction-driving vehicles that can only turn left; *SN* is the north–south-direction-driving

vehicles that can go straight and turn right; and *SNL* is the north–south-direction-driving vehicles that can only turn left. If the signal phase sequence is the only factor to be optimized, we may fix the signal durations as standard. If the action changes, a yellow light phase is inserted in between for safety

Reward

The variables in the reward function need to be closely related to the optimization objective of the model. The value of T is the waiting time of all vehicles in the previous time step minus the waiting time of all vehicles in the current time step, which is negative when the road conditions become worse. The Y value is the duration when the signal light switches to yellow. More yellow lights imply **more phase switching**, which increases delay and risk of congestion. Finally, the reward function can be expressed as follows:

$$R = k_1 T + k_2 Y \qquad k_1 + k_2 = 1, k_2 < k_1$$

T has a large influence on optimization efforts.

Action-selection strategy

Once the neural network predicts the best action using Q-values, it still needs to decide whether it should follow the best known action or something new. If the agent always chooses the highest Q-value, it might:

- Adopt a local optimum.
- Overfit to the early traffic patterns.
- Miss better strategies that it hasn't tried yet.

The idea is that the intelligence selects the action with the greatest known reward with the probability of $1 - \varepsilon$ at each decision, called exploitation, and randomly selects an action from the set of actions with probability of ε, called exploration:

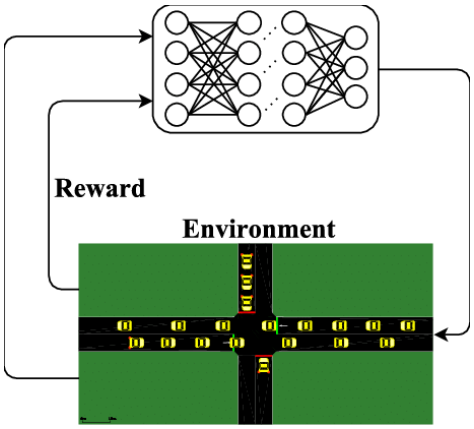
$$a = \begin{cases} \arg\max_a Q(s,a), 1-\varepsilon \\ random(A),\ \varepsilon \end{cases}$$

At the start, set ϵ=1 (pure exploration). As training progresses: Gradually reduce ε (shift toward exploitation). This is called ε decay or annealing.

Expand Model to multiple intersections

There is a need to increase the coordination between the traffic signals of the intersections in order to monitor the traffic flow appropriately. Therefore, controlling the traffic signals of multiple intersections at the same time always provides better traffic management than controlling the traffic signal of a single intersection individually.

- Enables the agent to perceive a spatio-temporal **global traffic state**, maintaining the same input-output neural pipeline.
- While Q-learning was effective for single intersection control, we adopt **Policy Gradient** methods for scaling. These allow for smoother optimization in large action spaces and better convergence in dynamic environments like multi-intersection networks.
- The agent is trained using policy gradient method in various deep neural network models i.e. fully connected neural network (FC NN), Convolutional neural network (CNN)) to approximate the action selection policy for achieving better management of traffic signal.

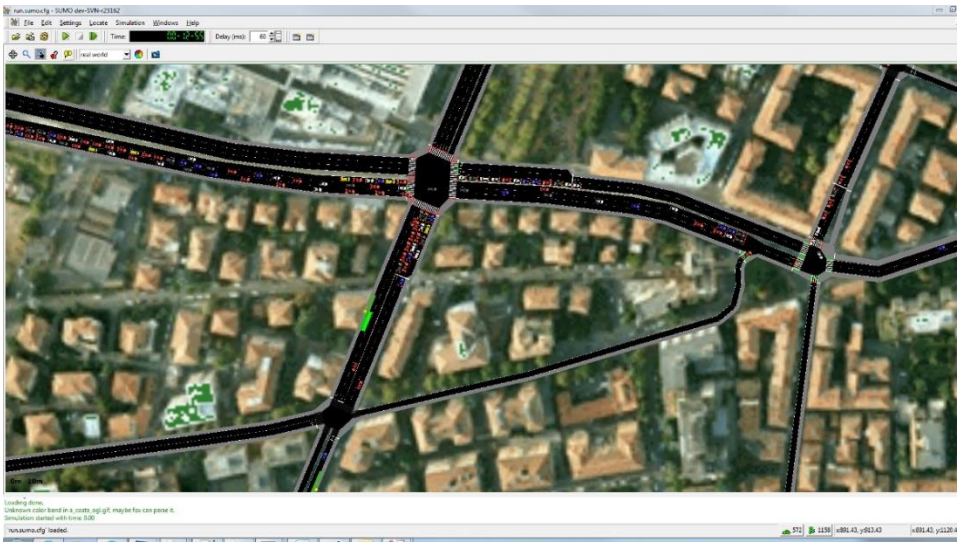


Here, the environment is created with the help of the simulator Simulation of Urban Mobility (SUMO) and TraCI.

SUMO Traffic Simulation Software

SUMO (Simulation of Urban Mobility) is an open source, microscopic, multi-modal traffic simulation software. Its microcosm is reflected in the fact that each vehicle is explicitly modeled with its own individual route of movement in the road network structure. It allows to simulate how a complex traffic flow consisting of many individual vehicles moves through a given road network structure after a given traffic demand.

SUMO has a fast OpenGL graphical user interface that enables the user to design each element of each roadway at an intersection, including the number of roadway strips, directions, and the number of lanes, the location of traffic signals and the phase sequence and duration of traffic signals. In addition, the package Traci in SUMO can easily enable it to communicate with python, which facilitates the simulation. By calling Traci, various data from the traffic simulation, such as vehicle waiting time, number of vehicles on the road, and vehicle speed, can be obtained in real time enabling us to optimize the model better.



File Type	Extension	Description
Network File	.net.xml	Describes the road network: nodes (intersections) and edges (roads). Various parameters of the intersection, such as the

		number of lanes, lane distribution, lane length and transition rules are defined here.
Route File	.rou.xml	Contains vehicle types, routes, departure times, and flows. Defines the movement of vehicles.
Executable File	. sumocfg	Main configuration file that references all other files. Used to launch the simulation.
Trip file	.trip.xml	Defines trips (origin/destination pairs), later converted to routes.
Polygon file	.poly.xml	For drawing background shapes (buildings, regions, etc.)
TLS (traffic light system)	.tll.xml	Traffic light logic definitions.
Demand file (flows)	.flow.xml	High-level definition of traffic demand using flows instead of individual vehicles.
Additional File	.add.xml	For traffic lights, detectors, bus stops, charging stations, etc.

To incorporate actual city maps, we may use OSM Web Wizard that launches the corresponding SUMO simulation.

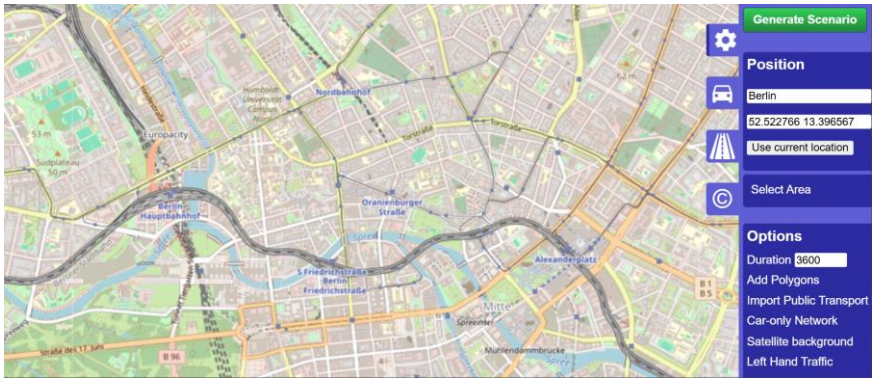
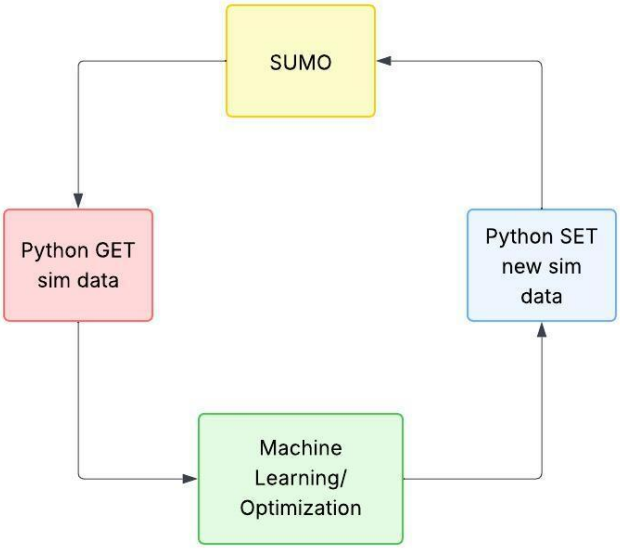


Figure: Interface of OSM Web Wizard

Figure: How to select a particular area of the map

Interaction with SUMO



The pythons' GET function uses the APIs from SUMO to collect data. It passes the data to our optimization models. Inverse to GET, the SET function 'sets' the new values into SUMO. TraCI helps facilitating the same.

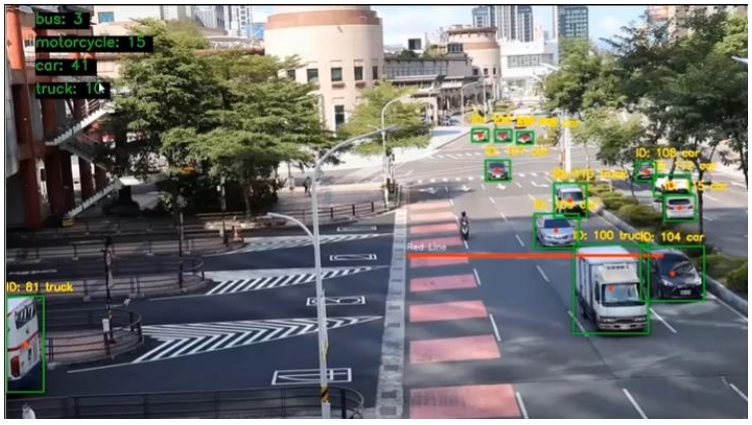
Open-CV

OpenCV (Open Source Computer Vision Library) provides tools for image and video analysis, object detection, and feature extraction—all crucial in interpreting live traffic scenes. OpenCV can process **live, real time video footage** to detect vehicles in each lane. This helps estimate parameters like -

queue length per lane, vehicle density per unit time, average wait time per vehicle, average vehicle count in a time frame.

My approach is to use the **YOLOv11 model**, for accurate vehicle detection and vehicle classification. Here's how I would implement the same:

- Detecting objects and associating them with a tracking ID (using a **BoT-SORT** algorithm) along class detection. A bounding box may contain the object.
- Put a dot in the centre of each object.
- Draw a reference line to count the vehicles when a dot containing object passes through it.
- For the **average wait time estimation,** we may start a timer for each vehicle ID when it enters a Region of Interest (RoI). Calculate the total time it remained within that area.
- Count of vehicles in an RoI in a specific time frame say 1 sec may be appended to a list. The process of counting will be repeated over a defined period (eg, 1 min). The mean of the values in the list would give **the average density of vehicles.**



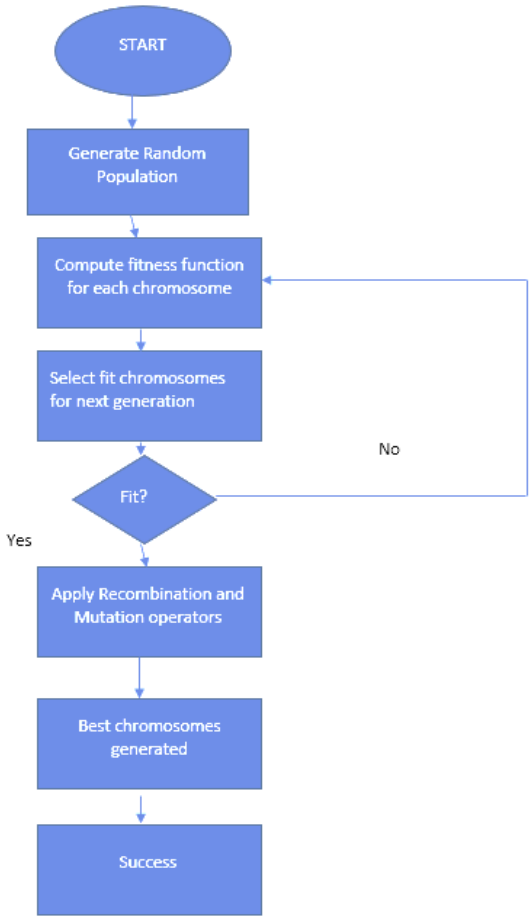
Genetic Algorithms

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. **Genetic algorithms simulate the process of natural selection** which means those species that can adapt to changes in their environment can survive and reproduce and go to the next generation. In simple words, they simulate **"survival of the fittest"** among individuals of consecutive generations to solve a problem.

Each generation consists of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

Here is my approach to incorporate GA in the system:

- Each chromosome represents a possible solution in the form of traffic light configuration, such as the time duration of green, yellow, and red signal.
- A **fitness function** evaluates how effective that configuration is—e.g., by measuring metrics such as average waiting time, vehicle throughput, or queue length.
- The GA iteratively selects the best configurations (selection), combines them (crossover), and makes random tweaks (mutation) to generate new configurations.
- After iterations of many generations, the algorithm converges toward better signal timing strategies.



Rule Based Algorithms

These systems operate on a **set of predefined rules and logic** to make decisions, perform tasks, or derive conclusions. Despite modern advancements, such as machine learning and neural networks, rule-based systems remain crucial owing to their ease of use, and simplicity.

In the context of the project, simple rules can be created like:

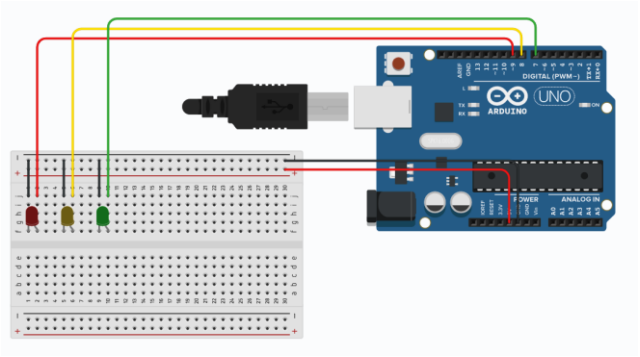
IF north-south traffic > east-west traffic, THEN extend green time for north-south.

IF queue length exceeds threshold, THEN reduce red time.

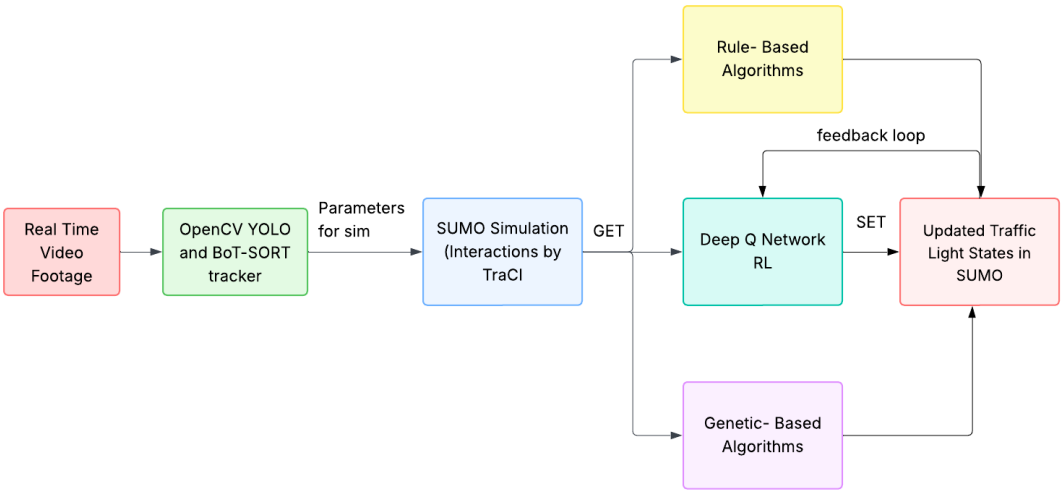
Traffic data (e.g., vehicle count or queue length from SUMO) is checked against the rules. The traffic signal controller responds accordingly.

Hardware Extension

Traffic data—like the number of vehicles waiting at a junction—can be simulated using IR sensors or a Raspberry Pi camera. This data acts as input for the RL model. For hardware deployment, the trained agent can be interfaced with an Arduino-based traffic signal prototype, where LEDs represent signal lights which changes it dynamically ofcourse.



Week-wise Workflow



Week 1: Real-Time Video Processing+YOLO

Set up OpenCV pipeline to process live video feed, detect vehicles per lane using YOLOv11 and BoT-SORT tracker.

Week 2: Creating SUMO Environment+TraCI Integration

Write GET/SET commands to collect traffic data and update signal phases.

Week 3: Rule-Based and Genetic Algorithm Signal Control

Implement rule-based logic to trigger traffic signal changes based on vehicle counts and hardcoded thresholds. Also build a Genetic Algorithm for optimizing fixed signal durations using fitness metrics like wait time and throughput.

Week 4 &5: Single-Intersection Reinforcement Learning with different types of lanes

Implement DQN for a single intersection using matrix-form state inputs and a neural network for Q-value prediction. Train agent using SUMO+TraCI feedback, including ε-greedy policy, experience replay, and dual-network architecture.

Week 6&7: Scaling to Multiple Intersections

Modify the agent to observe and act on global traffic states also switch to policy gradient method to optimize across larger action/state spaces.

Week 8: Documentation of the project & wrapping up.

Week 9: Hardware Integration

Task 1- “Aptitude Test”

Task Link: https://github.com/sofiaabidi/x-task-1

```
import cv2 as cv
import numpy as np
def task1(i1path,i2path,i3path,rows,cols=4):
    def apply_gray(imgcpy):
        gray=cv.cvtColor(imgcpy,cv.COLOR_BGR2GRAY)
        return gray
    i1=apply_gray(cv.imread(i1path))
    i2=apply_gray(cv.imread(i2path))
    i3=apply_gray(cv.imread(i3path))
    def findthresholdval(img): #finds the average value of all pixel values
        sumofpix=0
        noofpixels=img.shape[0]*img.shape[1]
        norm=img.tolist()
        for y in range(img.shape[0]):
            for x in range(img.shape[1]):
                sumofpix=sumofpix+norm[y][x]
        threshold_value=sumofpix/noofpixels
        return threshold_value
    def thresholding(img,threshold_value):
        for y in range(img.shape[0]):
            for x in range(img.shape[1]):
                if img[y][x]<threshold_value:
                    img[y][x]=0
                else:
                    img[y][x]=255
        return img
    i1=thresholding(i1,findthresholdval(i1))
    i2=thresholding(i2,findthresholdval(i2))
    i3=thresholding(i3,findthresholdval(i3))

    img=cv.bitwise_or(cv.bitwise_or(i1, i2),i3)
    cv.imwrite("OR.jpg",img)
    img=cv.imread("OR.jpg",cv.IMREAD_GRAYSCALE)

    kernel=np.ones((3,3),np.uint8)
    img=cv.erode(img,kernel,iterations=1)

    num_row_cnt=rows
    COL_OPTIONS=cols #(A-D)

    row_height=img.shape[0]//(num_row_cnt+2)#+2 for top options row and the
bottom margin
    col_width=img.shape[1]//(COL_OPTIONS+1)#+1 for the left margin
    results=[]
```

```
    letters=['A','B','C','D']

    for row in range(num_row_cnt):
        y_start=(row+1)*row_height
        y_end=(row+2)*row_height
        max=0
        flag=0
        for col in range(COL_OPTIONS):
            x_start=(col+1)*col_width
            x_end=(col+2)*col_width
            gridcell=img[y_start:y_end, x_start:x_end]
            white_count=cv.countNonZero(gridcell)
            if white_count>max:
                max= white_count
                flag=col

        letter=letters[flag]
        answer=str(row+1)+letter
        results.append(answer)
    print(",".join(results))
#TEST CASE 1
task1('Raghav.jpg','Ganshyam.jpg','Bhaskar.jpg',8)
#TEST CASE 2
task1('Raghav(1).jpg','Ganshyam(1).jpg','Bhaskar(1).jpg',10)
```

Output:

```
1D,2A,3B,4C,5C,6B,7B,8D
1A,2C,3B,4D,5A,6C,7B,8D,9D,10C
```

Task 2- “Shed your Bugs”

Task Link: https://github.com/sofiaabidi/x-task-2

```
import os
os.chdir('src')
forbidden_keywords=["print(","eval(","exec("]
for dirpath,dirnames,filenames in os.walk(os.getcwd()):
    for file in filenames:
        if file.endswith(".py"):
            full_path=os.path.join(dirpath,file)
            with open(file,'r',encoding='utf-8') as reader:
                violationcnt=0
                flagofforbidden=False
                line=reader.readline()
                c1=0
                c2=0 #counts of """ and '''
                while line != '' and flagofforbidden==False:
                    line=line.rstrip() #exludes trailing whitespace
                    charCount=len(line)
                    if(charCount>80):
                        violationcnt+=1
                    flagofopen=False
                    for char in line:
                        if flagofopen==False:
                            if char=='"':
                                flagofopen=True
                                occurenceindexofopening=line.find('"')
                                if(not('"' in
line[occurenceindexofopening+1:])):
                                    violationcnt+=1
                            elif char=="'" :
                                flagofopen=True
                                occurenceindexofopening=line.find("'")
                                if(not("'" in
line[occurenceindexofopening+1:])):
                                    violationcnt+=1
                        else:
                            break
                    c1=c1+line.count('"""')
                    c2=c2+line.count("'''")
                    for keyword in forbidden_keywords:
                        if(keyword in line):
                            occurenceindex=line.find(keyword)
                            if('#' not in line[0:occurenceindex]):
                                flagofforbidden=True
                    line=reader.readline()
            if(not(c1%2==0 and c2%2==0)):
                violationcnt+=1
```

```
            #print(f"Violation count {violationcnt}")
            #print(f"Forbidden {flagofforbidden}")
            path= os.path.relpath(full_path)
            if violationcnt>5 or flagofforbidden==True:
                print(f"{path}: HIGH RISK")
            elif (violationcnt>0 and violationcnt<=5):
                print(f"{path}: LOW RISK")
            else:
                print(f"{path}: CLEAN")
```

Output:

```
file1.py: LOW RISK
file2.py: CLEAN
file3.py: HIGH RISK
```

Task 3- “Lost in Translation”

Task Link: *https://github.com/sofiaabidi/x-task-3*

```
#pip install nltk on terminal before running
import nltk
from nltk import word_tokenize
from nltk import pos_tag
from nltk import sent_tokenize
from nltk.corpus import words

nltk.download('words')
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('averaged_perceptron_tagger')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('maxent_ne_chunker_tab')

#print(nltk.__version__) to confirm download successful
def preprocess_corpus(corpus):
    sentences=sent_tokenize(corpus)
    tokens=word_tokenize(corpus)
    tagged_tokens=nltk.pos_tag(tokens)
    chunk=nltk.ne_chunk(tagged_tokens)
    NE=[" ".join(w for w, t in ele) for ele in chunk if
isinstance(ele,nltk.Tree)]

    def corrections(word):
        common_mistakes={'e':'c','5':'s','c':'e','1':'l','h':'b'}
```

```
        corrected_word = ""
        flag=0
        for char in word:
            if char in common_mistakes and flag==0:
                corrected_word+=common_mistakes[char]
                flag=1
            else:
                corrected_word+=char

        return corrected_word

    english_words=set(words.words())
    for sentence in sentences:
        tokens=word_tokenize(sentence)
        tagged=pos_tag(tokens)
        corrected_words=[]
        for word,tag in tagged:
            if tag not in ['NNP', 'NNPS'] and word.lower() not in
english_words:
                corrected = corrections(word)
                if corrected.lower() != word.lower():
                    corrected_words.append(corrected)
                else:
                    corrected_words.append(word)
            else:
                corrected_words.append(word)
        result=corrected_words[0]
        for i in range(1,len(corrected_words)):
            if corrected_words[i-1].lower() in
["however","therefore","nevertheless","nonetheless","hence","moreover","conver
sely"]:
                result+=","
            if corrected_words[i] not in ".,!?;:)":
                result+= " "
            result+=corrected_words[i]
        print(result)
    print(NE)
    print()

corpus1="""In April 2023, Sundar Pichai did announce that Google would be
launehing a new AI product namcd Gemini.
Barack Obama also gave a speech at Harvard University, cmphasizing the role of
technology in modern education."""
corpus2="""Project X is an exclusive elub at Veermata Jijabai Technological
Institute, Mumbai, mcant to 5erve as a healthy environment for 5tudents to
learn from each other and grow together.
```

```
Through the guidance of their mcntors these 5tudents are able to complete
daunting tasks in a relatively short time frame, gaining significant exposure
and knowledge in their domain of choice."""
corpus3="""I will be eompleting my BTech dcgree in Mechanical Engineering from
VJTI in 2028"""
corpus4="""However the rcsults were clear"""

preprocess_corpus(corpus1)
preprocess_corpus(corpus2)
preprocess_corpus(corpus3)
preprocess_corpus(corpus4)
```

Output:

```
In April 2023, Sundar Pichai did announce that Google would be launching a new AI
product named Gemini.
Barack Obama also gave a speech at Harvard University, emphasizing the role of tec
hnology in modern education.
['Sundar Pichai', 'Google', 'Gemini', 'Barack Obama', 'Harvard University']

Project X is an exclusive club at Veermata Jijabai Technological Institute, Mumba
i, meant to serve as a healthy environment for students to learn from each other a
nd grow together.
Through the guidance of their mentors these students are able to complete daunting
tasks in a relatively short time frame, gaining significant exposure and knowledge
in their domain of choice.
['Veermata Jijabai Technological Institute', 'Mumbai']

I will be completing my BTech degree in Mechanical Engineering from VJTI in 2028
['BTech', 'Mechanical', 'VJTI']

However, the results were clear
[]
```

References

- “The Knapsack Problem & Genetic Algorithms – Computerphile” https://youtu.be/MacVqujSXWE?si=sG8meUG0j9IAHbUh
- https://www.geeksforgeeks.org/genetic-algorithms/
- https://www.geeksforgeeks.org/artificial-intelligence/rule-based-system-in-ai/
- A. Paul and S. Mitra, "Deep Reinforcement Learning based Traffic Signal optimization for Multiple Intersections in ITS," 2020 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), New Delhi, India, 2020
- Cao, K.; Wang, L.; Zhang, S.; Duan, L.; Jiang, G.; Sfarra, S.; Zhang, H.; Jung, H. Optimization Control of Adaptive Traffic Signal with Deep Reinforcement Learning. *Electronics* **2024**, *13*, 198.
- “Reinforcement Learning: Machine Learning Meets Control Theory” https://youtu.be/0MNVhXEX9to?si=-hoGoigbO2H6bGre
- “SUMO ("Simulation of Urban MObility") Tutorial” https://youtu.be/zQH1n0Fvxes?si=G3bX7g44IhxumD9A