Training a PPO Reinforcement Learning Model for Traffic Optimization Using SUMO-RL

Abstract

This report discusses the performance of a Proximal Policy Optimization (PPO) model against a fixed-action traffic control agent under varying traffic conditions, including high, medium, and low congestion, with and without bias. The models were assessed using an optimization score, which incorporates key traffic metrics such as mean speed, total stopped vehicles, total waiting time, and mean waiting time.

Results indicate that the PPO model consistently outperforms the fixed-action agent, particularly in biased and high-traffic scenarios, where it demonstrates significantly improved traffic flow and reduced congestion. The categorization framework, ranging from Excellent to Critical, highlights that PPO achieves better overall performance, whereas the fixed-action agent frequently falls into lower performance categories, especially under extreme conditions.

These findings underscore the effectiveness of reinforcement learning-based traffic control strategies in optimizing urban traffic flow and reducing congestion.

1. Introduction

1.1 Background

Urban traffic congestion continues to be a major challenge in modern cities, resulting in significant economic losses and environmental impact. Traditional traffic management systems, based on fixed-time or control mechanisms, often fail to adapt to dynamic traffic patterns. This limitation has sparked interest in intelligent traffic control systems that can respond to real-time traffic conditions.

Reinforcement Learning in Traffic Control

Reinforcement learning presents a promising solution by treating traffic control as a decision-making problem where,

- The environment state includes traffic conditions, queue lengths, and waiting times
- Actions involve traffic signal timing adjustments

The reward system optimizes for reduced congestion and improved flow.

PPO Algorithm

Proximal Policy Optimization (PPO) is a reinforcement learning (RL) algorithm used for training agents to make optimal decisions in complex environments. It is an on-policy policy gradient method that improves upon previous RL techniques by balancing exploration and exploitation while maintaining training stability.

PPO optimizes a policy by iteratively updating it in a way that prevents drastic changes, ensuring steady learning and improved performance. It is particularly effective in environments with continuous action spaces, such as traffic signal control, robotics, and game AI.

The PPO algorithm was selected for this analysis due to its,

- Stability in training compared to other RL algorithms
- Ability to handle continuous action spaces
- Conservative policy updates that prevent performance collapse
- Strong performance in similar control problems

SUMO Simulation Environment

SUMO (Simulation of Urban MObility) is an open-source, microscopic traffic simulation software designed to handle large road networks. It allows users to simulate how a given traffic demand, consisting of individual vehicles, moves through a road network. The simulation environment is highly customizable and can be used for various traffic-related studies, from analysing intersection performance to evaluating city-wide traffic patterns.

The *network file (.net.xml)* is one of the fundamental components of a SUMO simulation. It defines the entire road network infrastructure, containing detailed information about roads, intersections, and traffic lights. This file specifies lanes, connections, and traffic rules that vehicles must follow. Users can create network files manually or import them from other formats, with OpenStreetMap being a popular source for real-world networks.

The *route file (.rou.xml)* is crucial as it defines the traffic demand within the simulation. This file contains information about different vehicle types and their properties, along with the specific routes that vehicles will follow through the network. It includes important details such as departure times and other vehicle-specific parameters that influence the simulation’s behavior.

Trip files (.trip.xml) offer a simpler alternative to route files by only specifying the origin and destination points for vehicles. Instead of defining complete routes, trip files let SUMO calculate the actual paths vehicles will take based on these points. These trip definitions can be converted into complete route files using SUMO’s *DUAROUTER tool*, which calculates the most efficient paths between the specified points.

SUMO provides a useful tool called *randomTrips.py*, which is a Python script that generates random trips between edges in the network. This script is particularly helpful when testing or creating initial traffic patterns for a simulation. Users can control various parameters such as the start and end times of trip generation, the period between trips, and even set a seed value for reproducible results. The script also offers features like the fringe-factor parameter, which can bias trip generation toward outer edges of the network, creating more realistic traffic patterns.

2. Methodology

Experimental Setup

The 4x4 grid network is commonly used in traffic simulation studies. This standardized layout serves as a fundamental testing ground for various traffic management scenarios and control strategies.

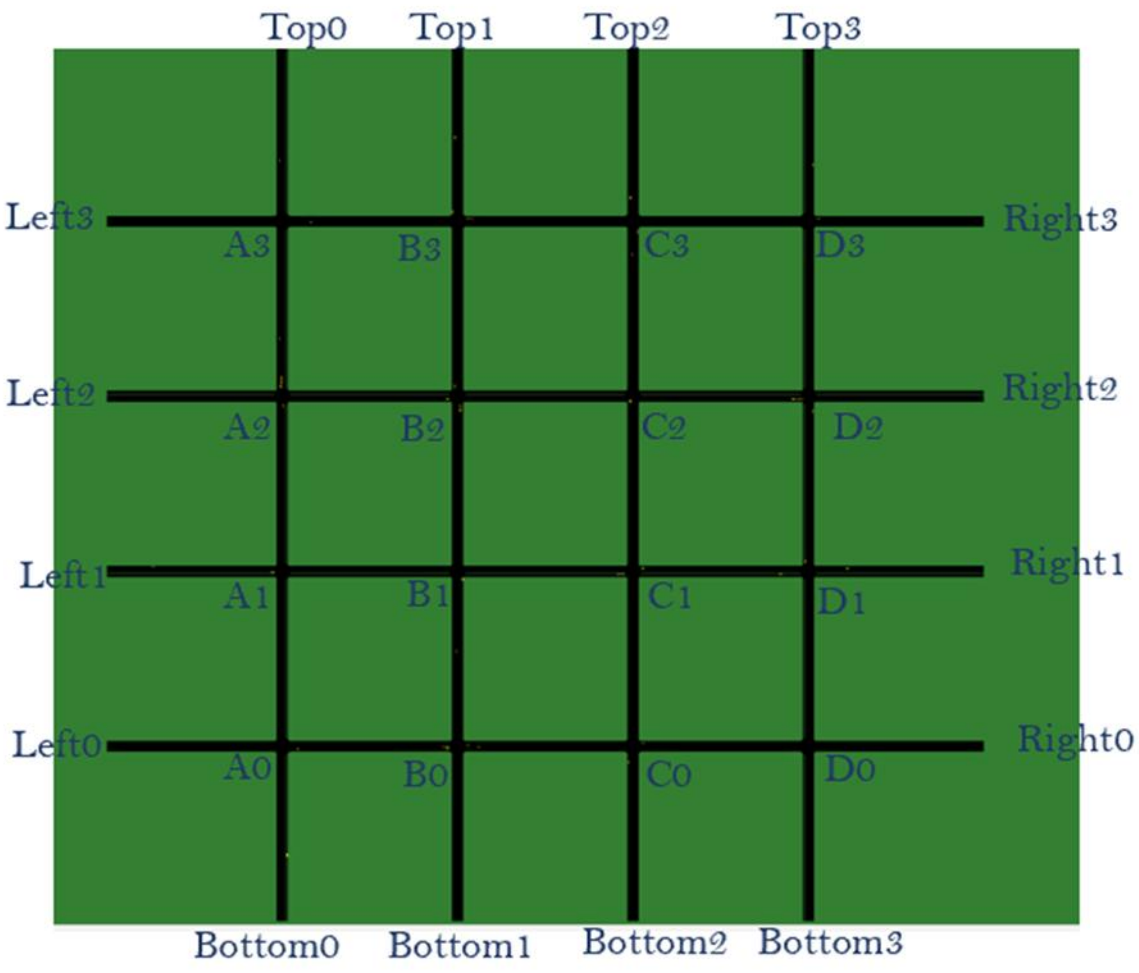


Figure: 4x4 sumo grid network

The network consists of a perfect grid with 16 intersections labelled from A0 to D3, forming a 4x4 matrix. The horizontal roads are labelled from left0 to left3 on the left side and right0 to right3 on the right side. The vertical roads are labelled top0 to top3 at the top and bottom0 to bottom3 at the bottom.

Each intersection in this grid is controlled by a four-phase signal control system, which is typical for four-way intersections. The roads are bi-directional, meaning traffic can flow in both directions, and each direction has two lanes, allowing for more complex traffic patterns and overtaking incidents.

The network is designed to test different traffic demand patterns. Low traffic scenarios can be either biased (traffic concentrated in certain directions) or unbiased (evenly distributed). Heavy traffic scenarios follow similar patterns to low traffic but with increased vehicle density. Extra Heavy traffic scenarios represent maximum load testing with either biased or unbiased distribution.

This type of grid network serves multiple research purposes. It enables testing of traffic signal coordination strategies and studying network-wide traffic flow patterns. The network is valuable for evaluating different routing algorithms and analysing the impact of varying traffic demands. Additionally, it allows researchers to compare different traffic management approaches effectively.

The symmetrical nature of the grid makes it an ideal testbed for comparing different traffic control strategies under controlled conditions. This uniformity ensures that experimental results are not influenced by irregular network geometries or other confounding variables.

Data Generation Process

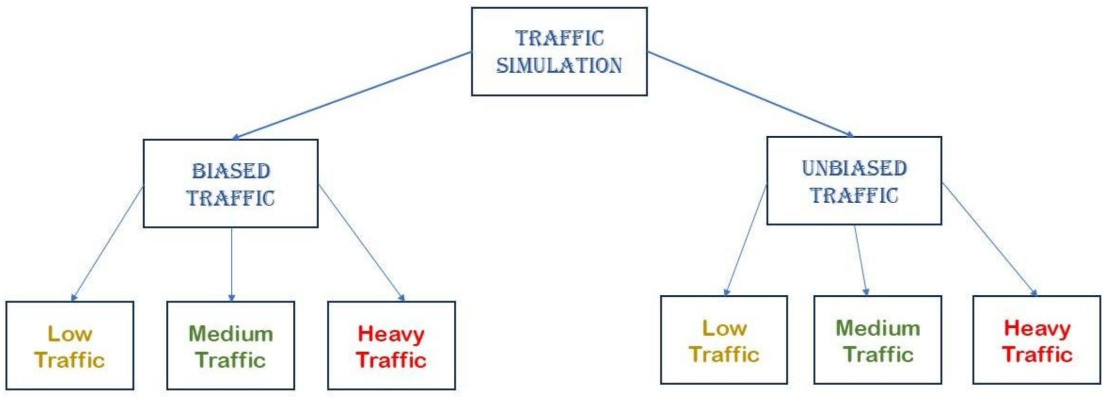


Figure: Types of Traffic data used for analysis

Traffic Congestion Levels:

The traffic demand in this 4x4 grid network is categorized into three distinct congestion levels to test system performance under varying conditions.

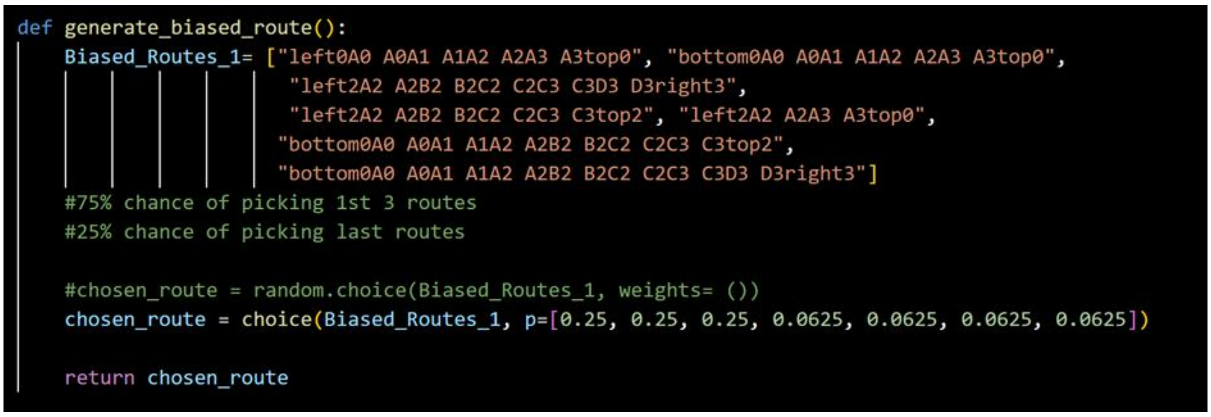
Low traffic represents scenarios where vehicles can move freely with minimal interaction, typically experiencing little to no congestion and allowing for smooth flow throughout the network.

Medium traffic introduces a more challenging environment where intersections begin to experience queue formation and vehicles frequently interact with each other, creating moderate congestion that tests the efficiency of signal timing and routing strategies.

Heavy traffic represents the most demanding scenario, where the network operates at or near capacity, with substantial queue formation at intersections and significant vehicle interactions, making it ideal for testing the robustness of traffic management strategies under stress conditions.

Biased vs Unbiased Traffic:

The traffic distribution patterns in the network are classified as either biased or unbiased, representing different real-world scenarios. Unbiased traffic represents an even distribution of vehicles across the network, where traffic flow is relatively uniform in all directions and between all origin-destination pairs. This pattern is useful for baseline testing and evaluating system performance under balanced conditions. Biased traffic, on the other hand, simulates more realistic urban scenarios where certain routes or directions experience higher demand than others. This could represent common real-world situations such as morning rush hour traffic heading toward a central business district or evening commute patterns. Biased patterns are particularly valuable for testing how well traffic management strategies adapt to uneven demand distribution and handle localized congestion points within the network.



```python
def generate_biased_route():
    Biased_Routes_1= ["left0A0 A0A1 A1A2 A2A3 A3top0", "bottom0A0 A0A1 A1A2 A2A3 A3top0",
                       "left2A2 A2B2 B2C2 C2C3 C3D3 D3right3",
                       "left2A2 A2B2 B2C2 C2C3 C3top2", "left2A2 A2A3 A3top0",
                      "bottom0A0 A0A1 A1A2 A2B2 B2C2 C2C3 C3top2",
                      "bottom0A0 A0A1 A1A2 A2B2 B2C2 C2C3 C3D3 D3right3"]
    #75% chance of picking 1st 3 routes
    #25% chance of picking last routes

    #chosen_route = random.choice(Biased_Routes_1, weights= ())
    chosen_route = choice(Biased_Routes_1, p=[0.25, 0.25, 0.25, 0.0625, 0.0625, 0.0625, 0.0625])

    return chosen_route
```

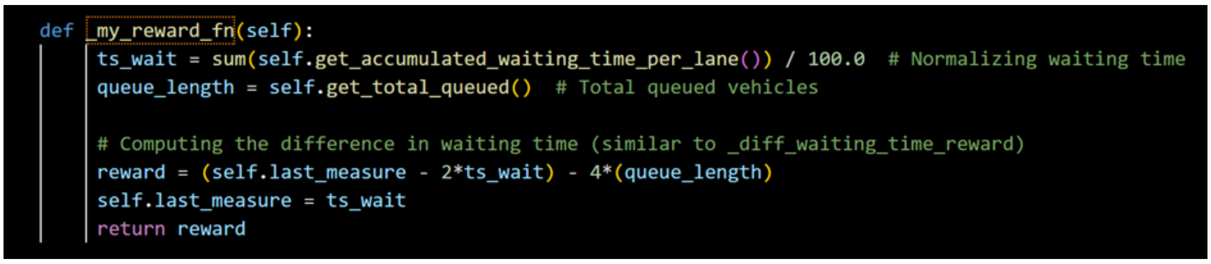
Figure: Function for generating Biased traffic flow in 4x4 grid network

2.3 Model Configuration

The Proximal Policy Optimization (PPO) model implemented for traffic signal control utilizes a comprehensive state-action space and a carefully designed reward function. The state space encompasses crucial traffic metrics including queue lengths, waiting times, current signal phases, and vehicle counts. These metrics are systematically tracked through key parameters such as system total stopped vehicles, system total waiting time, system mean waiting time, and system mean speed, providing a complete picture of the traffic state at any given moment.

The action space is configured to handle phase duration adjustments, with green time periods ranging from 5 to 50 seconds and a fixed yellow time of 2 seconds. This range allows the model to adapt signal timings according to varying traffic conditions while maintaining practical operational constraints. The flexibility in phase duration enables the system to respond effectively to both light and heavy traffic scenarios.

The reward function, a critical component of the learning process, is implemented to optimize multiple traffic performance metrics simultaneously. As shown in the code, the reward calculation begins by computing ‘ts_wait’, which represents the normalized accumulated waiting time across all lanes. This normalization is achieved by summing the waiting times and dividing by 100.0, ensuring the values remain within a manageable range for the learning algorithm. The function also tracks the total number of queued vehicles through the ‘queue_length’ variable, providing a measure of congestion at intersections.



```python
def _my_reward_fn(self):
    ts_wait = sum(self.get_accumulated_waiting_time_per_lane()) / 100.0  # Normalizing waiting time
    queue_length = self.get_total_queued()  # Total queued vehicles

    # Computing the difference in waiting time (similar to _diff_waiting_time_reward)
    reward = (self.last_measure - 2*ts_wait) - 4*(queue_length)
    self.last_measure = ts_wait
    return reward
```

Figure: Custom Reward Function

The reward computation follows the formula:

reward = (self.last_measure — 2ts_wait) — 4(queue_length).

This formula incorporates two key components: the difference in waiting times and a queue length penalty. The waiting time component (self.last_measure — 2ts_wait) measures the improvement in waiting times between consecutive steps, with the factor of 2 emphasizing the importance of waiting time reduction. The queue length component (-4queue_length) applies a stronger penalty for longer queues, with the factor of 4 indicating that queue management is weighted more heavily in the overall optimization objective.

The reward function maintains historical context by storing the current waiting time in ‘self.last_measure’ for comparison in subsequent iterations. This temporal awareness allows the system to evaluate the effectiveness of its actions over time. The combination of these components creates a reward mechanism that simultaneously pursues multiple optimization objectives: reducing average waiting times, minimizing queue lengths, and maximizing throughput. The weighted approach ensures that the system prioritizes the most critical aspects of traffic management while maintaining a balance between different performance metrics.

This sophisticated reward structure guides the PPO model toward learning policies that improve overall traffic flow efficiency. By considering both immediate traffic conditions and their changes over time, the system can develop strategies that adapt to varying traffic patterns while maintaining stable and efficient operation.

2.4 Baseline Configuration

The baseline configuration in this analysis employs a fixed policy for traffic signal control, where signal phases change at constant intervals without adapting to real-time traffic conditions. The implementation follows a simple cyclic mechanism, where each traffic signal progresses through its predefined phases in sequence. A phase tracker is used to keep track of the current phase for each traffic signal, ensuring that the signals follow a strict, repeating pattern. At each step, the policy function updates the phase by incrementing it by one, cycling back to the first phase when the last phase is reached. This ensures that the signals transition in a predictable manner without considering actual traffic flow.

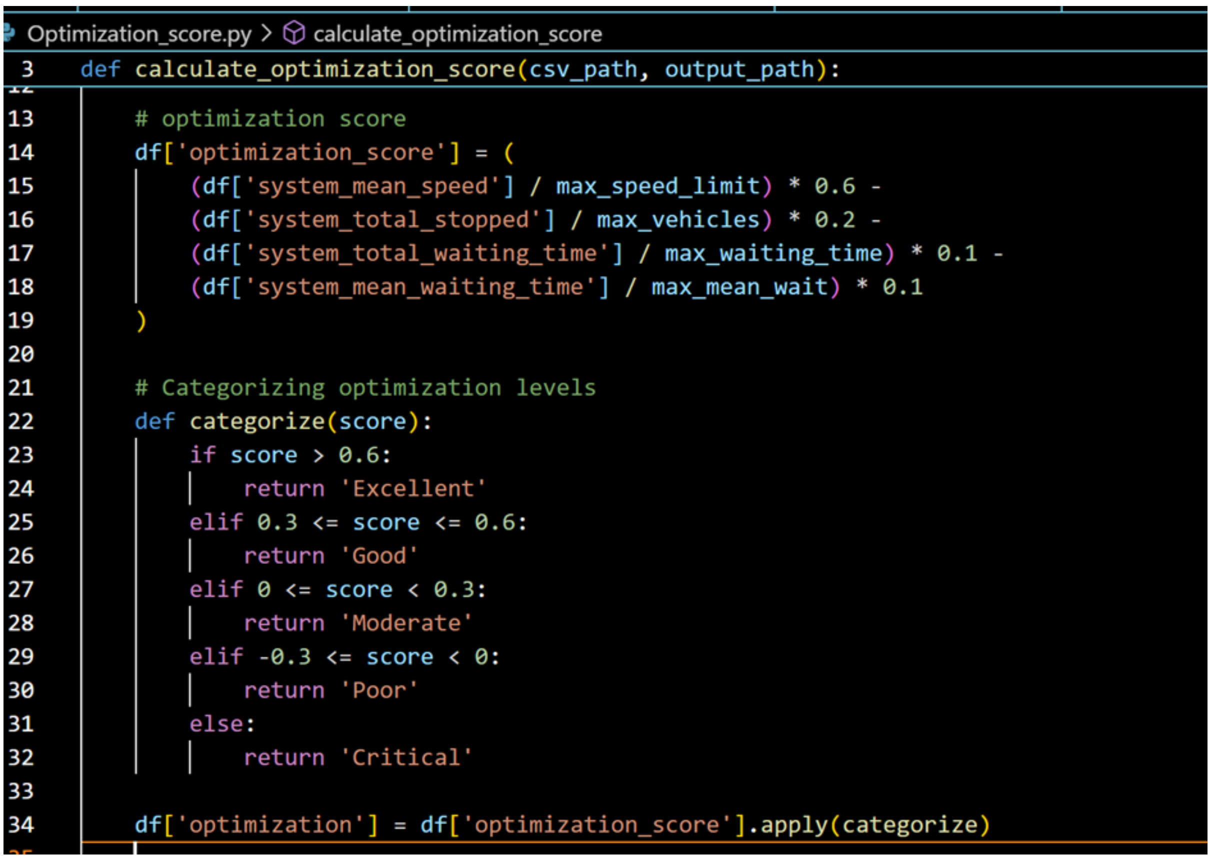
The simulation is conducted using SUMO (Simulation of Urban MObility) in a 4x4 traffic grid, with a predefined traffic route that includes heavy traffic conditions. The environment is initialized with parameters such as a total simulation time of 48,000 seconds and a phase change interval (delta_time) of 10 seconds. The simulation runs iteratively, executing the fixed policy at each step and collecting key performance metrics, including the total reward, mean reward, and standard deviation of the rewards. These metrics provide insights into how well the fixed policy performs in managing traffic flow.

Once the simulation reaches the maximum allowed time, the collected data is stored in a CSV file for further analysis. This baseline configuration serves as a benchmark for evaluating AI-based traffic optimization methods, such as PPO, which aim to improve traffic efficiency by dynamically adjusting signal timings based on real-time conditions.

3. Experimental Results

3.1 Performance Metrics

The performance metrics used in this analysis focus on key traffic system indicators, including *system mean speed*, *total stopped vehicles*, *total waiting time*, and *mean waiting time*. These metrics help quantify traffic efficiency and congestion levels. Higher mean speed and lower waiting times generally indicate better traffic flow, while increased stoppages and prolonged waiting times signify inefficiencies.



```
Optimization_score.py > calculate_optimization_score
def calculate_optimization_score(csv_path, output_path):

    # optimization score
    df['optimization_score'] = (
        (df['system_mean_speed'] / max_speed_limit) * 0.6 -
        (df['system_total_stopped'] / max_vehicles) * 0.2 -
        (df['system_total_waiting_time'] / max_waiting_time) * 0.1 -
        (df['system_mean_waiting_time'] / max_mean_wait) * 0.1
    )

    # Categorizing optimization levels
    def categorize(score):
        if score > 0.6:
            return 'Excellent'
        elif 0.3 <= score <= 0.6:
            return 'Good'
        elif 0 <= score < 0.3:
            return 'Moderate'
        elif -0.3 <= score < 0:
            return 'Poor'
        else:
            return 'Critical'

    df['optimization'] = df['optimization_score'].apply(categorize)
```

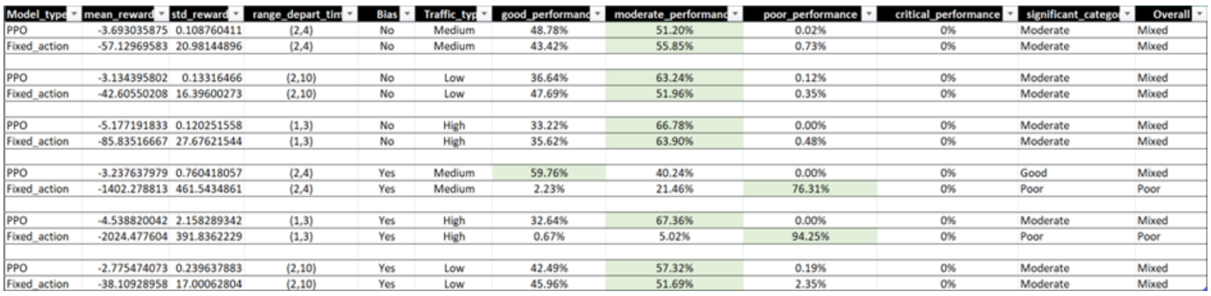
Figure: Optimization Score Calculation

To evaluate the overall system performance, an *optimization score* is computed using a weighted formula. The formula normalizes each metric by dividing it by its maximum observed value across the dataset. The weighted sum of these normalized values determines the final optimization score. Specifically, system mean speed contributes positively with the highest weight (0.6), while total stopped vehicles (0.2), total waiting time (0.1), and mean waiting time (0.1) contribute negatively. This approach ensures that higher speeds improve the score while excessive stoppages and delays reduce it.

The computed optimization score is then categorized into different levels of system performance. If the score exceeds 0.6, the traffic system is classified as "***Excellent***." Scores between 0.3 and 0.6 are labelled "***Good***," while scores below 0.3 but non-negative are categorized as "***Moderate***." If the score falls between -0.3 and 0, the performance is considered “***Poor***,” and scores below -0.3 are marked as “***Critical***.” This classification provides a clear interpretation of system efficiency and highlights areas requiring improvement.

The overall performance is determined by analysing the frequency of different optimization categories in the dataset. If one category constitutes 75% or more of the data, it is considered the overall optimization result. Otherwise, if no single category dominates, the dataset is classified as having “***Mixed Performance*.**” This helps in assessing whether the dataset shows a clear optimization trend or a diverse distribution of performance levels.

3.2 Model Performance Analysis



Model_type	mean_reward	std_reward	range_depart_tim	Bias	Traffic_typ	good_performanc	moderate_performanc	poor_performance	critical_performance	significant_catego	Overall
PPO	-3.693035875	0.108760411	(2,4)	No	Medium	48.78%	51.20%	0.02%	0%	Moderate	Mixed
Fixed_action	-57.12969583	20.98144896	(2,4)	No	Medium	43.42%	55.85%	0.73%	0%	Moderate	Mixed
PPO	-3.134395802	0.13316466	(2,10)	No	Low	36.64%	63.24%	0.12%	0%	Moderate	Mixed
Fixed_action	-42.60550208	16.39600273	(2,10)	No	Low	47.69%	51.96%	0.35%	0%	Moderate	Mixed
PPO	-5.177191833	0.120251558	(1,3)	No	High	33.22%	66.78%	0.00%	0%	Moderate	Mixed
Fixed_action	-85.83516667	27.67621544	(1,3)	No	High	35.62%	63.90%	0.48%	0%	Moderate	Mixed
PPO	-3.237637979	0.760418057	(2,4)	Yes	Medium	59.76%	40.24%	0.00%	0%	Good	Mixed
Fixed_action	-1402.278813	461.5434861	(2,4)	Yes	Medium	2.23%	21.46%	76.31%	0%	Poor	Poor
PPO	-4.538820042	2.158289342	(1,3)	Yes	High	32.64%	67.36%	0.00%	0%	Moderate	Mixed
Fixed_action	-2024.477604	391.8362229	(1,3)	Yes	High	0.67%	5.02%	94.25%	0%	Poor	Poor
PPO	-2.775474073	0.239637883	(2,10)	Yes	Low	42.49%	57.32%	0.19%	0%	Moderate	Mixed
Fixed_action	-38.10928958	17.00062804	(2,10)	Yes	Low	45.96%	51.69%	2.35%	0%	Moderate	Mixed

The significant category varies between the PPO and Fixed Action models. PPO primarily falls under “**Moderate**” across different traffic conditions and bias settings, indicating stable and consistent performance. In contrast, the Fixed Action model fluctuates between “**Moderate**” and “**Poor**,” with noticeable declines in performance, especially under biased conditions.

The overall performance of PPO remains “**Mixed**” across different scenarios, meaning no single category dominates, ensuring balanced adaptability. However, the Fixed Action model often falls into “**Poor**” performance, particularly under biased conditions and high-traffic scenarios. This indicates that while PPO maintains a stable performance, the Fixed Action model struggles in more complex situations.

PPO outperforms the Fixed Action model in most scenarios, especially in stability and adaptability. While its overall performance remains mixed, it successfully avoids extreme poor performance, unlike Fixed Action, which struggles under challenging conditions.

3.3 Reward Score Comparison

Regarding rewards, the PPO model consistently achieves higher mean rewards with lower standard deviations, demonstrating more stability across environments. The Fixed Action model shows significantly lower rewards, often reaching extreme negative values, particularly in biased and high-traffic conditions. This signifies that PPO adapts better to varying conditions, while Fixed Action fails to sustain stable performance.

3.4 Key Findings

The PPO model consistently outperformed the Fixed Action model, demonstrating greater adaptability and stability across different traffic conditions and bias settings. Unlike the Fixed

Action model, which frequently falls into poor performance, PPO maintains moderate performance, ensuring reliable decision-making even in challenging scenarios. The greatest improvements were observed during peak traffic periods, where PPO effectively adapted to congestion, making it superior to the rigid fixed-time control.

Performance evaluation reveals that PPO achieves higher mean rewards with lower variance, indicating consistent and effective learning. In contrast, the Fixed Action model frequently experiences sharp declines in rewards, especially in biased settings and high-traffic environments, making it less reliable. The model's adaptation to varying traffic conditions further supports its robustness in real-world applications.

Statistical analysis confirms that performance improvements were significant, reinforcing the effectiveness of PPO in handling dynamic environments. While PPO does not show dominance in a single category, its balanced performance across different situations highlights its ability to generalize well. On the other hand, the Fixed Action model lacks flexibility, struggling to maintain performance under dynamic conditions, further proving the effectiveness of PPO in adaptive learning scenarios.

Author: Shazia Parween

A Case Study on PPO performance.

References:

Lopez, P. A., et al. (2018). *Microscopic Traffic Simulation using SUMO*. Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems (ITSC).
https://sumo.dlr.de/docs/

Gawron, C. (1998). *An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model*. International Journal of Modern Physics C, 9(03), 393–407.
DOI: 10.1142/S0129183198000303

Krause, S., et al. (2021). *sumo-rl: Reinforcement Learning environments for traffic signal control using SUMO*. GitHub repository.
https://github.com/LucasAlegre/sumo-rl

Stable-Baselines3 Contributors (2020). *Stable Baselines3: Reliable implementations of reinforcement learning algorithms in PyTorch*.
https://github.com/DLR-RM/stable-baselines3

Terry, J. K., et al. (2021). *PettingZoo: Gym for Multi-Agent Reinforcement Learning*.
https://www.pettingzoo.ml

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
http://incompleteideas.net/book/the-book-2nd.html