



פרויקט מס' BS-CS-25-126

Smart Traffic Control

תחבורה חכמה

פרויקט הנדסי

דו"ח מסכם

הוכן לשם השלמת הדרישות לקבלת

תואר ראשון במדעי הטבע . **B. Sc**

מאת

אור אביטל, לוטם כהן

בהנחיית ד"ר דביר רוס

הוגש למחלקה למדעי המחשב

המכללה האקדמית להנדסה סמי שמעון

באר-שבע

תאריך לועזי

22/06/2025

תאריך עברי

י"ט אלול התשפ"ה



Smart Traffic Control

תחבורה חכמה

פרויקט גמר

דו"ח מסכם

הוכן לשם השלמת הדרישות לקבלת

תואר ראשון במדעי הטבע . **B. Sc**

מאת

אור אביטל, לוטם כהן

הוגש למחלקה למדעי המחשב

המכללה האקדמית להנדסה סמי שמעון

באר-שבע

_____ תאריך :	_____ חתימת סטודנט 1 :
_____ תאריך :	_____ חתימת סטודנט 2 :
_____ תאריך :	_____ חתימת המנחה :
_____ תאריך :	_____ אישור ועדת הפרויקטים :



Table of Content:

Abstract	5
1. Introduction	6
2. Literature review	7
2.1 Background	7
2.1.1 Deep learning	7
2.1.2 Reinforcement learning	7
2.1.3 Q-Learning	9
2.1.4 Deep reinforcement learning	9
2.1.5 Deep Q-Learning	10
2.1.6 Multi-Agent Reinforcement Learning	10
2.2 Related work	10
2.2.1 Conventional Traffic Light Control	11
2.2.2 Q-Learning for traffic light control	11
2.2.3 Deep Q-Network (DQN) for traffic light control	11
2.2.4 Multi-Agent RL for traffic light control	12
3. Method	12
3.1 Simulation Environment Design	12
3.1.1 Traffic Light Phases	13
3.1.2 Car Arrivals	13
3.1.3 State Representation	14
3.2 Agent Architecture and Action Space	14
3.2.1 classifying the State Space	14
3.2.2 Action Definition	14
3.2.3 Q-Table Structure	15
3.3 Performance Monitoring and Visualization	15
3.4 Learning Algorithm	16
3.5 Reward Function	17



3.6	Training Procedure	18
4.	Results	19
4.1.	Comparative Results	19
4.2.	Visualization	27
5.	Discussion	28
6.	Conclusion	30
7.	References	31



תקציר:

מטרת פרויקט זה היא לפתח סימולציה של אותות תעבורה המותאמים לתנאי תנועה בזמן אמת באמצעות אלגוריתמים למידת חיזוק. המערכת תמזער את הגודש על ידי לימוד אסטרטגיות תזמון איתות אופטימליות באמצעות ניסוי וטעייה. על ידי השוואת תוצאות עם שיטות בקרת תעבורה מסורתיות, הפרויקט שואף לתרום למחקר על מערכות תחבורה חכמות.

Abstract:

The goal of this project is to develop a simulation of traffic signals that adapt to real-time traffic conditions using reinforcement learning algorithms. The system will minimize congestion by learning optimal signal timing strategies through trial and error. By comparing results with traditional traffic control methods, the project aims to contribute to research on intelligent transportation systems.



1. Introduction:

Traffic congestion has become one of the most urgent challenges in today's urban environments. Millions of people worldwide spend countless hours each day stuck in traffic, affecting productivity and quality of life.

Current road intersections management is done through traffic lights. The inefficient traffic light control causes numerous problems, such as long delays of travelers, huge waste of energy and worsening air quality. In some cases, it may also contribute to vehicular accidents. [2].

Traffic congestion has also become increasingly costly. For example, Wei, Yao, Zheng & Li [3] noted that traffic congestion costs Americans \$124 billion a year.

Improving traffic conditions could increase city efficiency, improve the economy, and ease people's daily life. One way to reduce traffic congestion is by intelligently controlling traffic lights. To dynamically adjust traffic lights according to real time traffic, people have been using reinforcement learning techniques such as Deep Q-learning [3].

Another solution being used today is Multi- Agent reinforcement learning. Researchers have begun to apply multi-agent reinforcement learning (MARL) algorithms to traffic light control. One of the main contributions of this method is that it no longer treats different kind of vehicles with the same priority. Buses are used as urban public transport, and the number of passengers is more than the number of passengers on ordinary vehicles. Therefore, buses are given a higher priority to pass the intersection [4].

In summary, traffic congestion is a critical issue, but solutions like reinforcement learning and multi-agent systems offer promising approaches. This review aims to analyze existing researches and to demonstrate these methods and their concepts as they're crucial for advancing effective traffic management systems.



2. Literature Review

2.1. Background:

In this section, we briefly introduce concepts and fundamentals in machine learning, deep learning and cover some RL basics. [7]

In order to solve traffic congestion in vehicular networks, we need to establish a model of deep reinforcement learning.

2.1.1. Deep Learning:

In machine learning, input data is often transformed through manual feature engineering before training. In deep learning, the process involves one or more hidden layers between the input and output layers. Each unit in a hidden layer calculates its input as a weighted sum of the outputs from the previous layer, followed by a nonlinear activation function to create a new representation. Weights link the units across layers, guiding the flow of data. After forward computation from input to output, errors are calculated and backpropagated through the layers to update the weights and optimize a loss function, improving the model's performance [7]

2.1.2. Reinforcement learning

Reinforcement learning is one category of algorithms in machine learning, which is different from supervised learning and unsupervised learning [2].

In the standard reinforcement learning model an agent interacts with its environment. This interaction takes the form of the agent sensing the environment, choosing an action to perform in the environment based on its input. The action changes the environment in some manner and this change is communicated to the agent through a scalar reinforcement signal [6] , [7].



The Environment

In reinforcement learning, the environment is a dynamic system with which the RL agent interacts through trial and error, receiving observations that guide its actions. Ideally, the agent can observe the environment's state, enabling it to make optimal decisions based on true states [6].

The Reinforcement Function

As stated previously, RL systems learn a mapping from situations to actions by trial-and-error interactions with a dynamic environment. The “goal” of the RL system is defined using the concept of a reinforcement function, which is the exact function of future reinforcements the agent seeks to maximize. After performing an action in a given state the RL agent will receive a reward in the form of a scalar value. The RL agent learns to perform actions that will maximize the sum of the rewards when starting from some initial state and proceeding to a terminal state [6].

The Value Function

To explain the issue of how the agent learns to choose “good” actions, or even how we might measure the utility of an action we need to define two terms:

A policy determines which action should be performed in each state; a policy is a mapping from states to actions.

The value of a state is defined as the sum of the reinforcements received when starting in that state and following some fixed policy to a terminal state.

The optimal policy would therefore be the mapping from states to actions that maximizes the sum of the reinforcements when starting in an arbitrary state and performing actions until a terminal state is reached. Under this definition the value of a state is dependent upon the policy. The value function is a mapping from states to state values and can be approximated using any type of function approximator (e.g., multilayered perceptron, memory based system, radial basis functions, look-up table) [6].



With the development of deep learning and reinforcement learning, they are combined as deep reinforcement learning to estimate the Q value. [2].

2.1.3. Q-Learning:

The Q-learning algorithm consists of several key components:

- **Q-table**: Stores the estimated value of each state-action pair, representing the expected future reward.
- **Bellman update**: A formula used to iteratively update Q-values based on the reward received and the maximum future reward.
- **Learning rate (α)**: Controls how much newly acquired information overrides old values.
- **Discount factor (γ)**: Determines the importance of future rewards.
- **ϵ -greedy policy**: Balances exploration (trying new actions) and exploitation (using the best-known action).
- **State classifying**: Simplifies the state space by grouping continuous values into fixed categories.

These elements allow the agent to gradually learn optimal decisions through repeated interaction with the environment. [10]

2.1.4. Deep Reinforcement Learning:

Deep RL combines reinforcement learning (RL) with deep neural networks to enable agents to make decisions in complex environments [2]. In general, DRL is based on training deep neural networks to approximate the optimal policy, and/or the optimal value functions [8].

2.1.5. Deep Q-Learning

Deep Q-Network (DQN) is a deep reinforcement learning method used to map states to Q-values, with each Q-value corresponding to a possible traffic light phase. One of the main advantages of Deep Neural Networks (DNNs) is the capability of automating feature extraction from raw input data. A DQN uses this benefit of deep learning in order to represent the agent's observation as an abstract representation in learning an optimal control policy [5].

The algorithm learns via **experience replay**. At each time step, the DQN stores the agent's experience (current state, action, reward, and next state) in a data set. It then randomly samples a batch of experiences from this dataset to update the network's parameters. This experience replay mechanism helps stabilize the learning process and reduce correlations between updates [5].

2.1.6. Multi-Agent Reinforcement Learning

In MARL, there are several agents operating in fully cooperative environments with partial observations. Each agent makes decisions based on partial information with the aim of improving outcomes for all agents in the environment.

No single agent can observe the state of the environment. Instead, each agent receives a private observation that is correlated with that state. We assume the agents cannot explicitly communicate and must learn cooperative behavior only from their observations [9].

2.2. Related Work:

In this section, we first introduce conventional methods for traffic light control, then introduce methods using reinforcement learning.



2.2.1 Conventional Traffic Light Control:

Early traffic light control methods can be divided into two main categories. The first is pre-timed signal control, where green light durations are predetermined based on historical traffic patterns, without accounting for changes in real-time traffic demand.

The second category is vehicle-actuated control, which uses real-time traffic data to adjust signals. While this approach is effective in handling high variability in traffic, it heavily relies on manually crafted rules tailored to current conditions and does not consider future traffic scenarios, making it incapable of achieving globally optimal performance [3].

2.2.2 Q-Learning for traffic light control:

A trial-and-error learning algorithm that allows one to learn the action values (Q values) of states and actions. This is one of the earliest approaches to solving the problem. The algorithm updates the Q table based on the rewards received, when one wants to find the actions that maximize the cumulative reward. Using a Q-learning model is effective in small state spaces, but has difficulty dealing with the large complexity of real road networks. In addition, it requires manual updating of the Q table, which makes it impractical in dynamic environments [2].

2.2.3 Deep Q-Network (DQN) for traffic light control:

DQL integrates a deep neural network to estimate Q-values, overcoming the limitations of table-based Q-Learning. DQN uses deep learning techniques to generalize the Q-value estimates to new situations. The algorithm is effective in large and complex state and action spaces, such as traffic control. Being able to learn meaningful representations of traffic situations, for example by processing security camera images, and prone to instability and fluctuations during learning. Also often leads to better results than classical methods [2], [3], [5].

2.2.4 Multi-Agent Reinforcement Learning (MARL) for traffic light control:

The MARL approach uses multiple intersections as multiple agents in an urban traffic network. The approach considers the mutual effects between agents, with the goal of maximizing the overall reward. Using MARL improves performance in controlling large traffic networks, where cooperation between traffic lights is critical. This method deals better with the dynamics of the environment compared to single-agent approaches and is able to adapt traffic light control even when there is uncertainty in the data. It is often combined with other algorithms such as DQN [4].

3. Method

This section describes the technical framework used in our project: the simulation environment, agent design, reinforcement learning settings, reward function, and training process. Our goal was to build a realistic environment for evaluating reinforcement learning techniques in controlling traffic light systems at a single intersection.

3.1. Simulation Environment Design

We developed a custom environment in Python using the OpenAI Gym interface to simulate a four-way intersection. Each direction (north, south, east, and west) has three incoming lanes: straight lane, right-turn lane and a left-turn lane. In total, the intersection handles 12 incoming traffic lanes.

The environment is implemented using several key classes:

- **Lane class:** Manages vehicle queues and tracks performance metrics including vehicles passed and average waiting times
- **Car class:** Represents individual vehicles with arrival timestamps for precise waiting time calculations

- **TrafficLight class:** Controls phase transitions and maintains active direction states
- **Intersection class:** Serves as the main environment following the OpenAI Gym interface

3.1.1 Traffic Light Phases

The environment uses a simplified traffic light controller that cycles between 6 possible non-conflicting movement groups (phases):

Phase 0: ["north_straight", "south_straight"]

Phase 1: ["north_left", "south_right"]

Phase 2: ["east_straight", "west_straight"]

Phase 3: ["east_left", "west_right"]

Phase 4: ["north_right", "south_left"]

Phase 5: ["east_right", "west_left"]

At each time step, the agent selects a phase to activate, determining which directions receive green lights. This design ensures simultaneous flow from compatible directions while preventing traffic conflicts. The phase-based approach simplifies the control problem while maintaining realistic intersection behavior.

3.1.2 Car Arrivals

New vehicles arrive at each lane independently at every time step based on a Poisson distribution with a configurable arrival rate λ (default: $\lambda = 2$). This simulates the random nature of real-world traffic.



3.1.3. State Representation

Each environment state is represented by a 13-dimensional vector:

- Queue lengths for the 12 lanes (integer counts of cars)
- Index of the currently active phase (integer from 0 to 5)

This state representation provides complete observability of the intersection status, enabling the agent to make informed decisions based on current traffic conditions across all approaches.

3.2. Agent Architecture and Action Space

The agent implements Q-learning with a discrete state-action space to handle the complexity of traffic intersection control.

3.2.1 classifying the State Space

To reduce the dimensionality of the Q-table, each queue length is classified into 4 bins:

- **Category 0:** 0–6 cars (low congestion)
- **Category 1:** 7–13 cars (moderate congestion)
- **Category 2:** 14–20 cars (high congestion)
- **Category 3:** 21+ cars (severe congestion)

The discretized state becomes a tuple: $(q_1, q_2, \dots, q_{12}, \text{phase_index})$ where $q_i \in \{0,1,2,3\}$ and $\text{phase_index} \in \{0,\dots,5\}$. This discretization preserves essential traffic congestion patterns while maintaining computational tractability.

3.2.2 Action Definition

The agent selects actions as tuples: $\text{action} = (\text{phase_index}, \text{car_group_index})$ where:

- **phase_index** $\in \{0, 1, 2, 3, 4, 5\}$: Specifies which movement group to activate
- **car_group_index** $\in \{0, 5, 10, 15, 20, 25, 30\}$: Specifies the index of the number of vehicles to process from each active lane, corresponding to values $\{0, 5, 10, 15, 20, 25, 30\}$

This leads to a total of $6 \times 7 = 42$ possible actions.

3.2.3 Q-Table Structure

The Q-table maps discrete state tuples to a 6×7 matrix of action-values, one for each possible pair (phase and discharge count). This table is updated as the agent interacts with the environment.

3.3. Performance Monitoring and Visualization

The simulation includes a comprehensive real-time visualization system that provides detailed monitoring of intersection dynamics and agent behavior. The graphical interface displays a realistic four-way intersection with clearly marked lanes, road boundaries, and pedestrian crosswalks. Individual vehicles are rendered as colored rectangles with windshields and wheels, queued in their respective lanes with different colors to enhance visual distinction. Traffic lights are positioned at each lane approach, displaying red or green states based on the current active phase selected by the agent. Queue length indicators appear as numbered circles near each lane, providing instant feedback on congestion levels. The visualization updates in real-time during both training and evaluation phases, displaying the current time step, active traffic light configuration, and intersection status in the title bar. This visual monitoring system enables qualitative assessment of agent learning progression, identification of traffic flow patterns, and validation of policy effectiveness through direct observation of intersection behavior.

3.4. Learning Algorithm

The Q-learning algorithm consists of several key components:

Q-table: Stores the estimated value of each state-action pair, representing the expected future reward. Our implementation uses a dictionary structure mapping discrete state tuples to 6×7 numpy arrays, with dynamic initialization for new states encountered during traffic scenarios.

Bellman update: A formula used to iteratively update Q-values based on the reward received and the maximum future reward. Our implementation splits this into two steps: first calculating the target value:

```
target = reward + self.gamma * np.max(self.q_table[s_]),
```

and then updating using the equivalent form:

```
self.q_table[s][a0, a1] = (1 - self.alpha) * self.q_table[s][a0, a1] + self.alpha * target.
```

The implementation includes validation checks to ensure states exist in the Q-table and actions are within valid bounds before updating.

Learning rate ($\alpha = 0.1$): Controls how much newly acquired information overrides old values. This moderate rate balances learning speed with stability for traffic control applications.

Discount factor ($\gamma = 0.9$): Determines the importance of future rewards. The high value emphasizes long-term traffic flow optimization over immediate gains.

ϵ -greedy policy: Balances exploration (trying new actions) and exploitation (using the best-known action). Starts with $\epsilon = 1.0$ (full exploration), decays by 0.995 per episode, and maintains minimum exploration of $\epsilon = 0.1$.

State discretization: Simplifies the state space by grouping continuous values into fixed categories. Queue lengths are discretized using



`int(min(3, queue_length // 7)),`

creating four categories (0-6, 7-13, 14-20, 21+ cars) that preserve essential congestion patterns.

These elements allow the agent to gradually learn optimal decisions through repeated interaction with the traffic environment, with Q-table persistence enabling continued learning across training sessions.

3.5. Reward Function

The reward function is carefully crafted to balance multiple objectives:

$$\begin{aligned} \text{reward} = & -(0.25 \times \text{waiting_penalty} + 0.1 \times \text{max_queue} + \text{overcrowding_penalty} \\ & + \text{severe_overcrowding_penalty}) \\ & + 3.0 \times \text{passed_this_step} + \text{underload_bonus} \end{aligned}$$

Components Analysis:

- **Waiting penalty:** Combines average waiting times ($0.5 \times \text{wait_time}$) and queue lengths ($0.3 \times \text{queue_length}$) across all lanes
- **Overcrowding penalties:**
 - Moderate penalty ($0.8 \times \text{excess}$) for queues > 35 cars
 - Severe penalty ($1.2 \times \text{excess}$) for queues > 45 cars
- **Throughput reward:** 3.0 points per vehicle successfully processed
- **Underload bonus:** 0.3 points per car below 25-car threshold, encouraging proactive queue management

This multi-objective formulation encourages efficient vehicle processing while penalizing excessive congestion and rewarding balanced traffic distribution across all approaches.



3.6. Training Procedure

The agent training follows an episodic reinforcement learning protocol:

Initialization: Each episode begins with empty lanes and reset traffic light state. The exploration rate (ϵ) decays episodically to gradually shift from exploration to exploitation.

Step-by-step Process:

1. State observation: Agent receives current 13-dimensional state vector
2. Action selection: ϵ -greedy strategy balances exploration and exploitation
3. Environment step: Traffic light phase activated, vehicles processed, new arrivals generated
4. Reward calculation: Multi-component reward computed based on performance metrics
5. Q-table update: Bellman equation applied with boundary validation
6. State transition: Process repeats until episode termination (200 steps)

Training Policy:

- **Episode length:** 200 time steps per episode
- **Training duration:** 6,000 episodes with intermediate Q-table saves every 300 episodes

Performance Monitoring: Real-time visualization displays intersection state, active phases, queue lengths, and agent decisions, enabling qualitative assessment of learning progression and policy effectiveness.

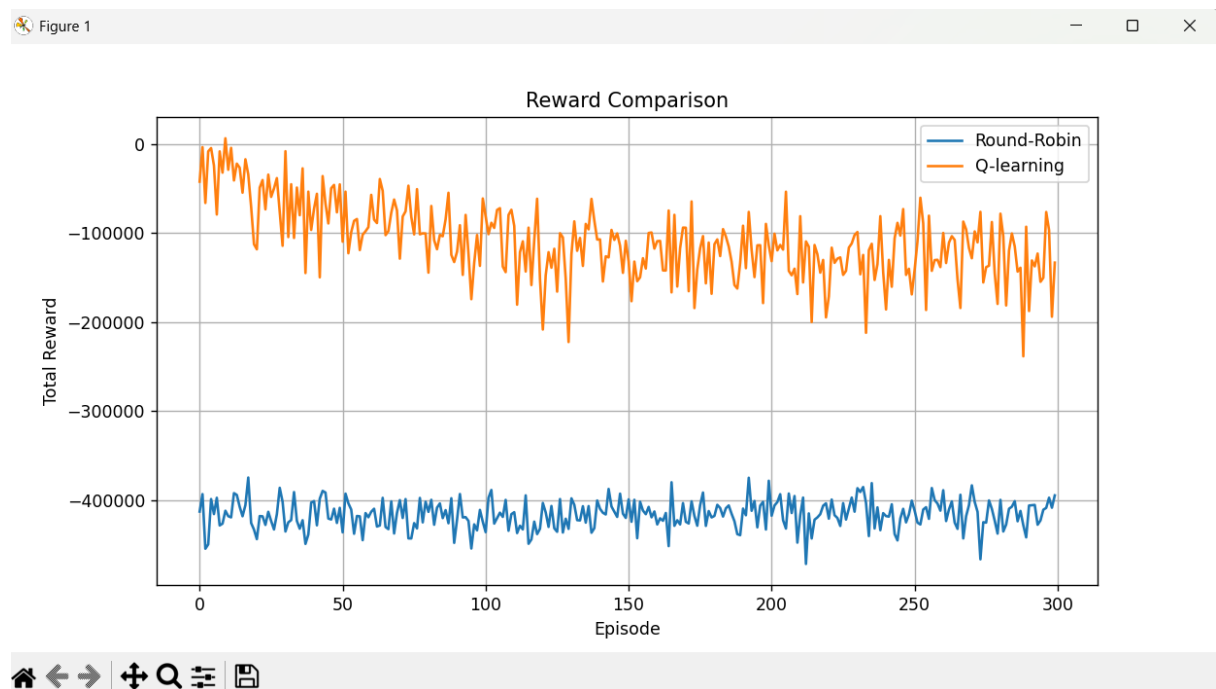
The training procedure enables systematic policy development through repeated environment interaction while maintaining detailed performance tracking for analysis and optimization.

4. Results

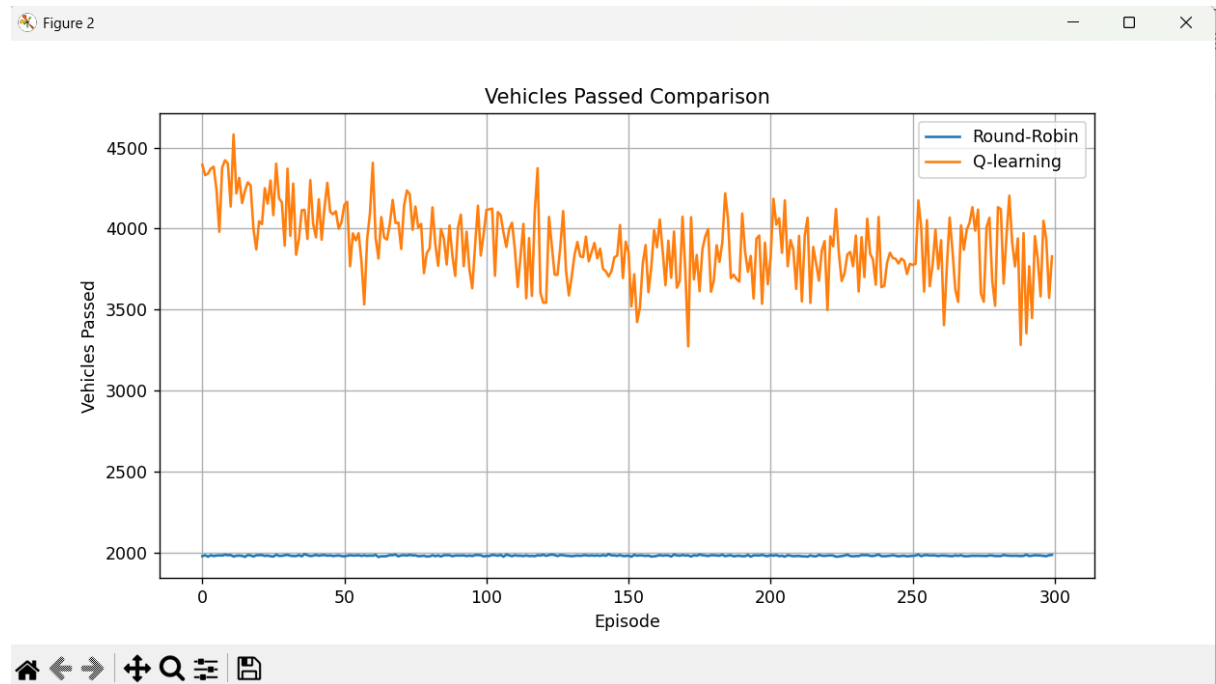
4.1. Comparative Results

As an initial benchmark, the baseline model (Round-Robin) was executed and compared against an untrained Q-learning agent to assess potential improvements from reinforcement learning. During this stage, three key metrics were collected:

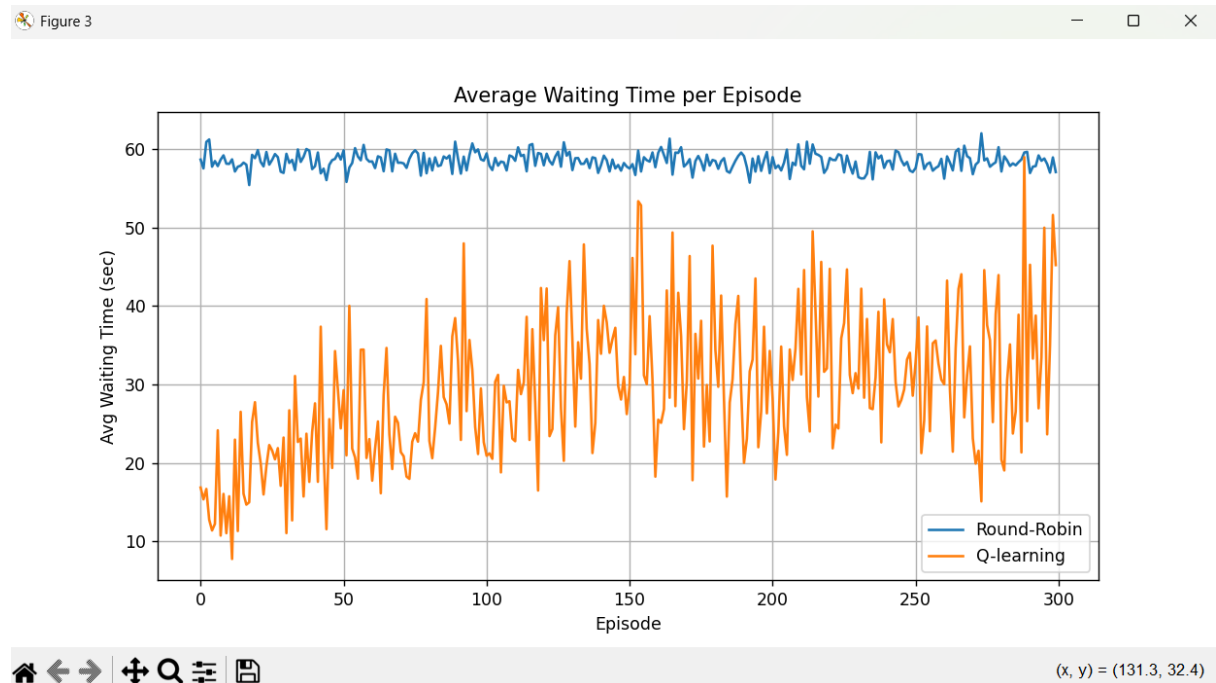
- Total Reward: Used to evaluate the overall effectiveness of the policy.



- Average Waiting Time: Measured to assess traffic efficiency and delays.



- Number of Passed Cars: Tracked to reflect throughput at the intersection.

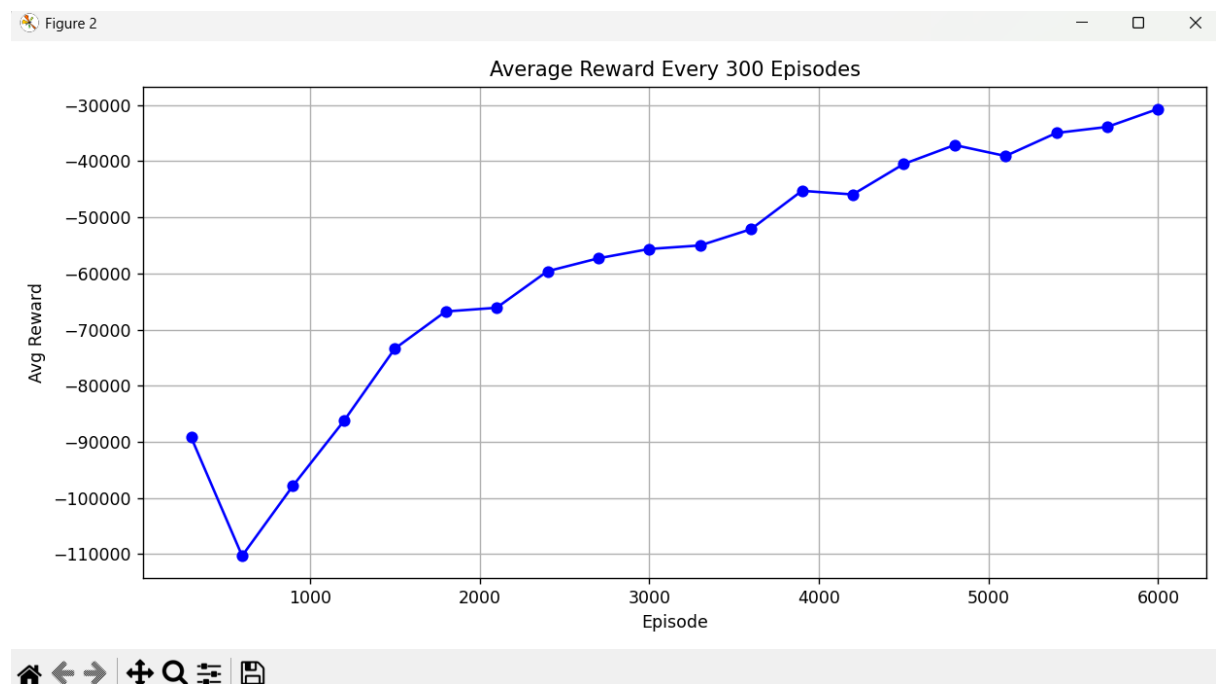


These results served as a reference point for evaluating the future performance of the learning-based model.

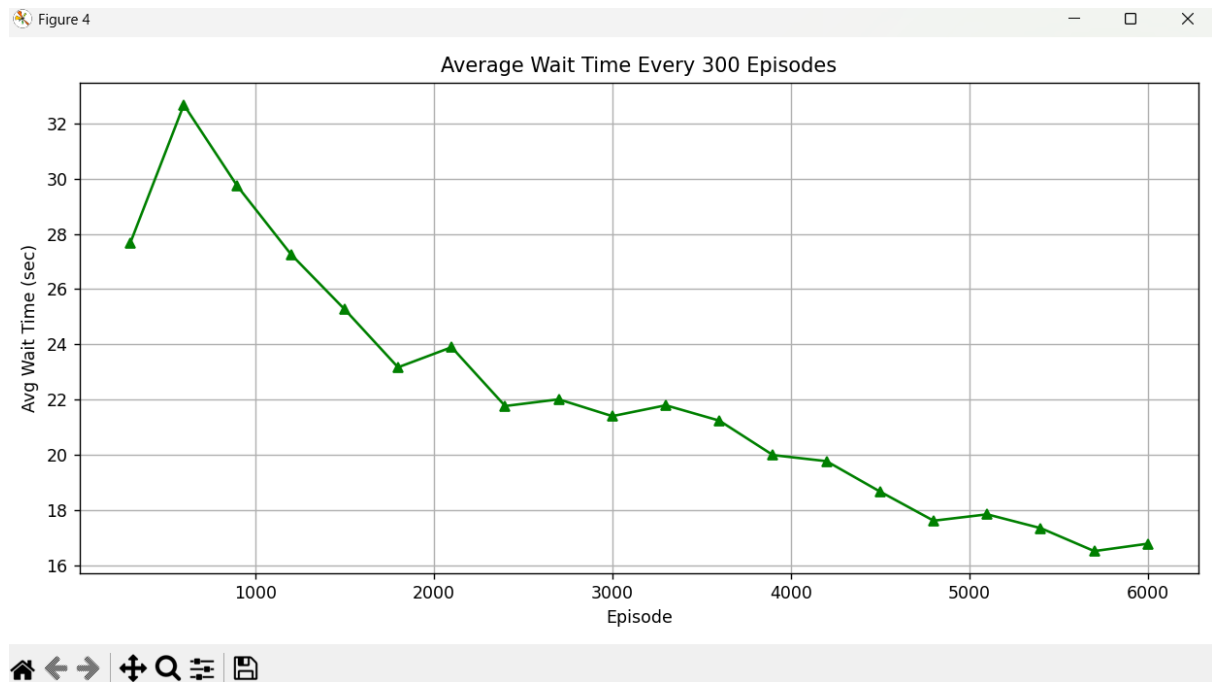


The next step was training the Q-learning agent over a total of 6000 episodes. Throughout the training process, performance metrics were recorded every 300 episodes to analyze the learning trends. The exploration rate (ϵ) was gradually decreased to encourage more exploitation over time, and the Q-table was continuously updated based on the agent's experiences. At the end of training, the final Q-table was saved for future use.

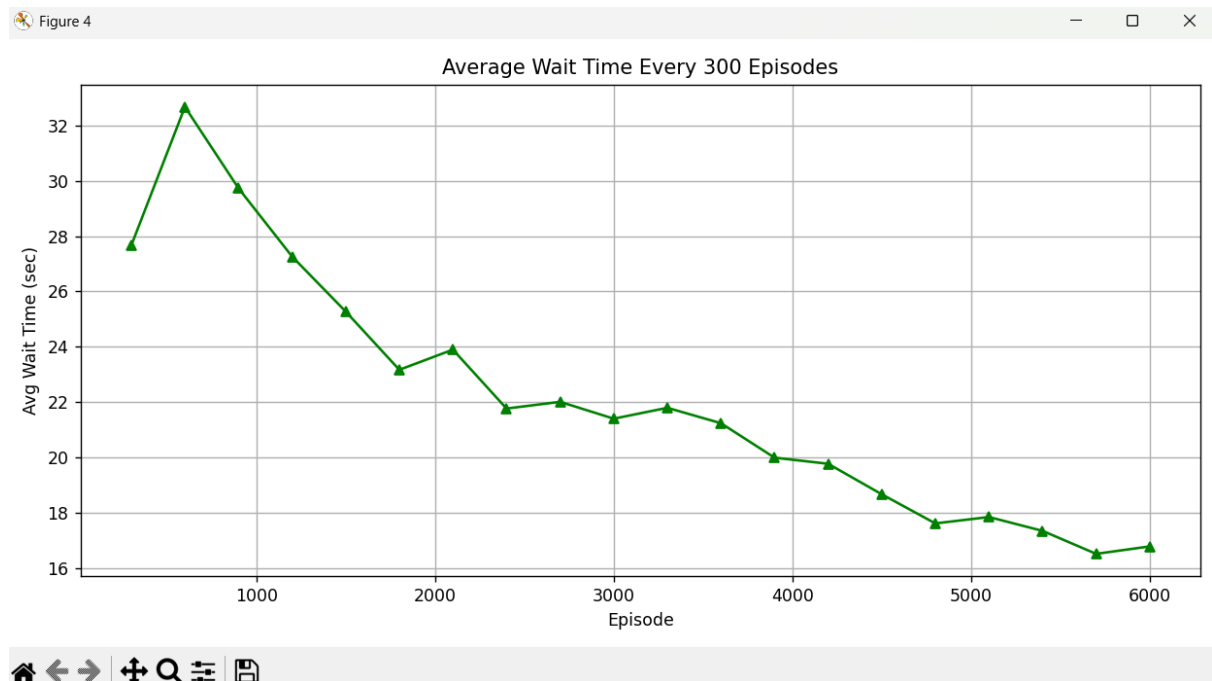
- Total Reward: Tracked periodically to observe improvements in the agent's decision-making.



- Average Waiting Time: Monitored to evaluate the agent's impact on reducing traffic congestion.



- Number of Passed Cars: Measured to assess throughput and intersection efficiency.



While running the episodes, we can see the following improvements:
the agent's reward and the number of passing cars are increasing, and the average waiting time is decreasing.

Performance Comparison Summary:

The evaluation graphs clearly show that the trained Q-learning agent outperforms the Round-Robin baseline across all three key metrics:

- Total Reward:

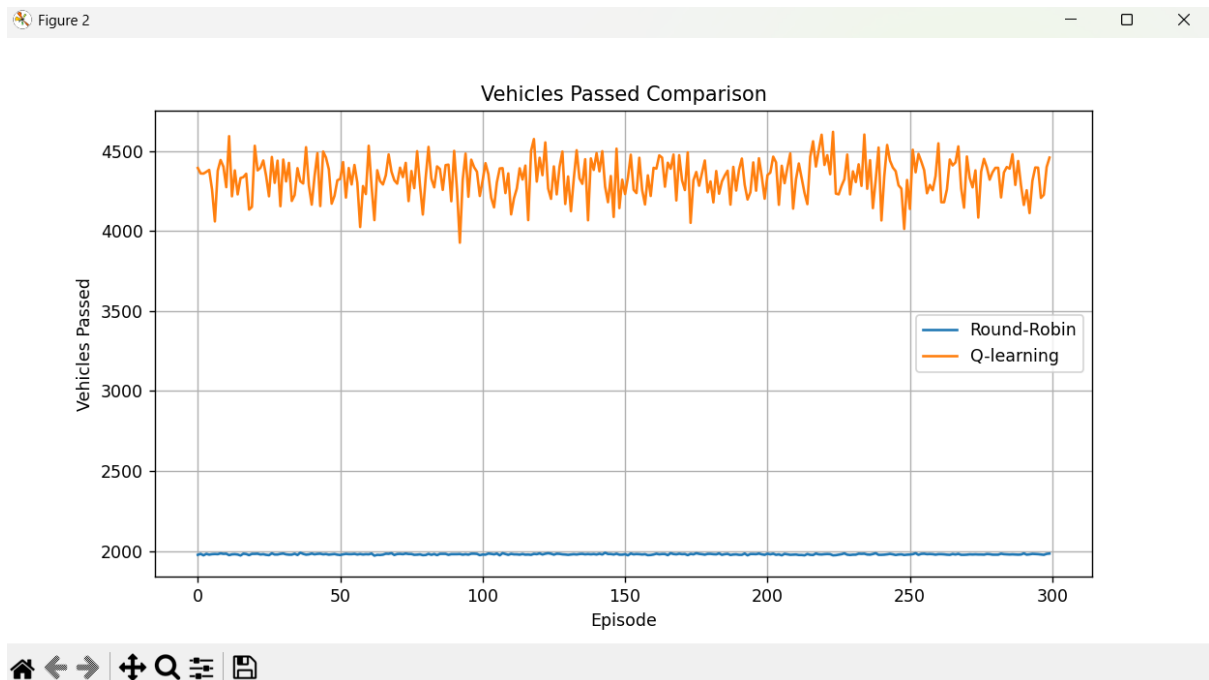
The Q-learning model achieved significantly higher rewards throughout the episodes, while the Round-Robin model remained at consistently lower reward levels. This indicates that the learning agent developed a more effective decision-making policy.



- Vehicles Passed:

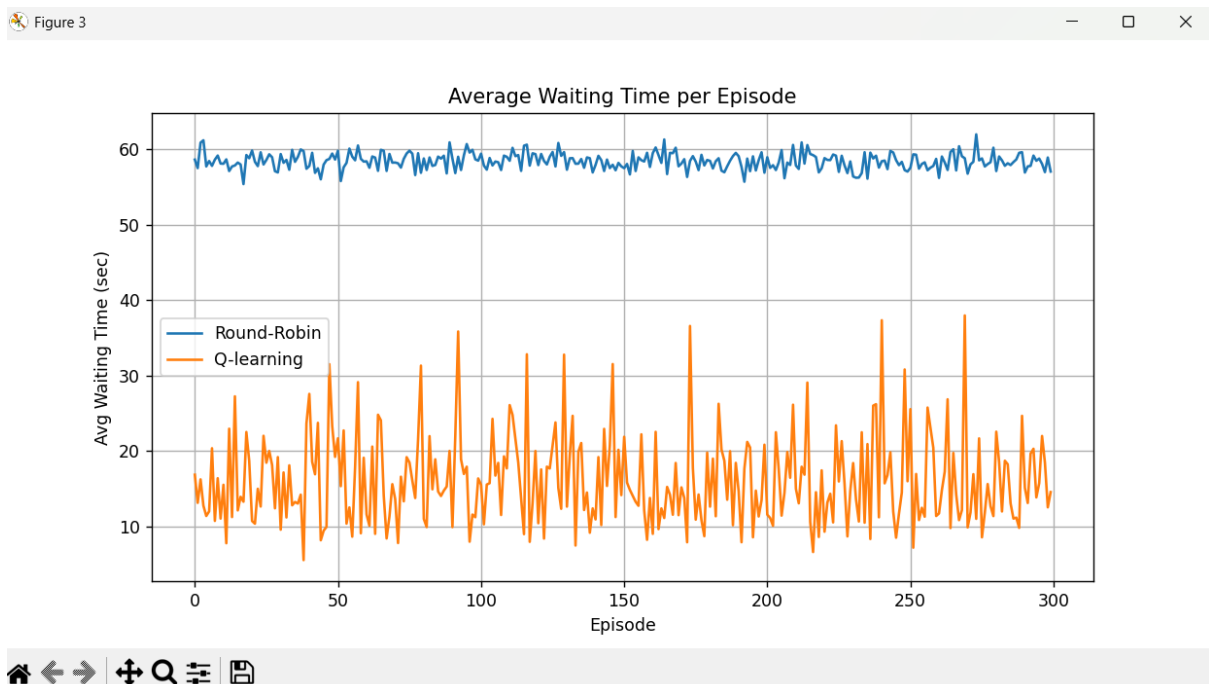
The number of vehicles that passed the intersection was consistently higher with the Q-learning agent. The Round-Robin policy maintained a low and stable throughput, whereas the Q-learning agent enabled a much more efficient traffic

flow.



- Average Waiting Time:

The Q-learning agent resulted in much shorter and more variable waiting times, while the Round-Robin model maintained a consistently high waiting time. This reflects better traffic management and reduced delays with the trained model.



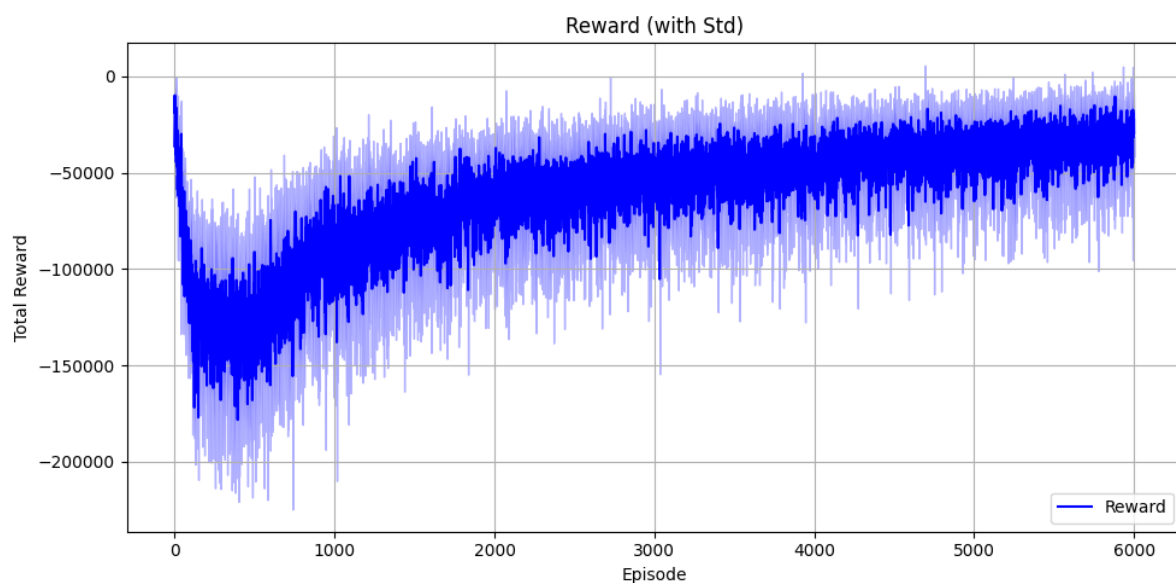


Overall, the trained Q-learning agent demonstrated clear improvements in performance, validating the effectiveness of reinforcement learning in optimizing traffic signal control.

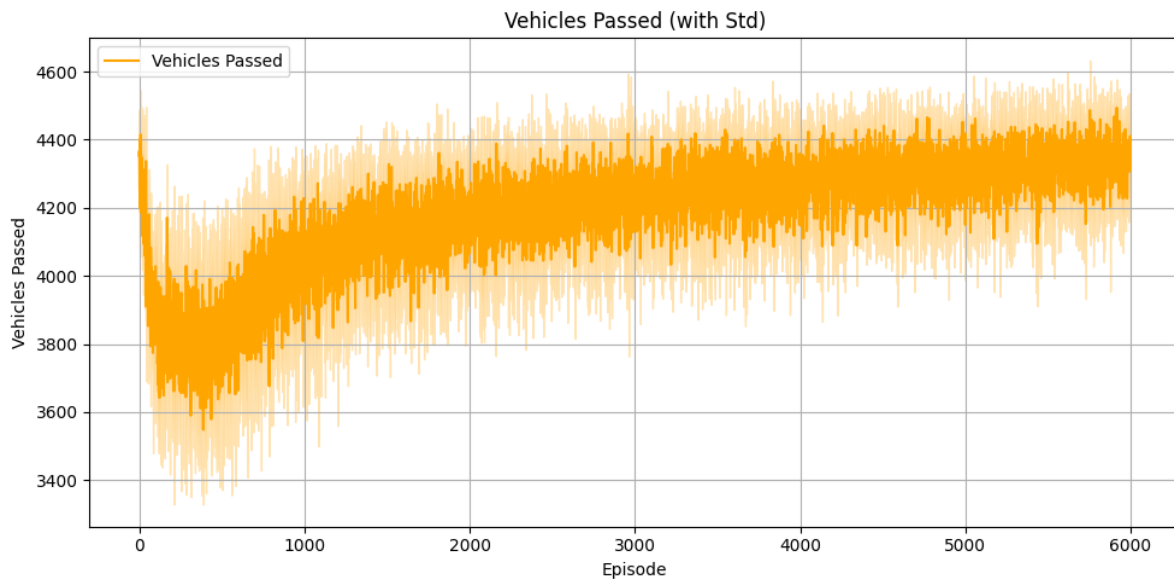
To assess the consistency and reliability of the Q-learning agent's performance, we conducted five independent training runs, each initialized with a different random seed. The agent was trained for 6,000 episodes in each run under identical traffic conditions. The resulting performance metrics were averaged across runs, and the standard deviation was computed to evaluate variability.

Three primary metrics were analyzed:

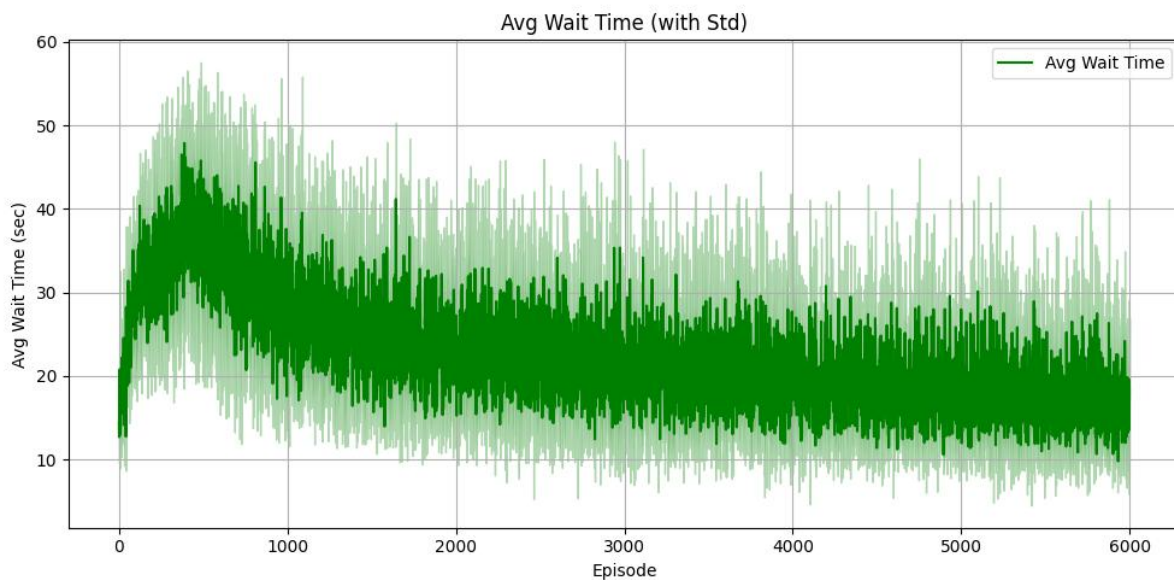
- **Total Reward** – indicating the overall learning effectiveness.



- **Vehicles Passed** – reflecting intersection throughput.



- **Average Wait Time** – measuring user experience and congestion levels.



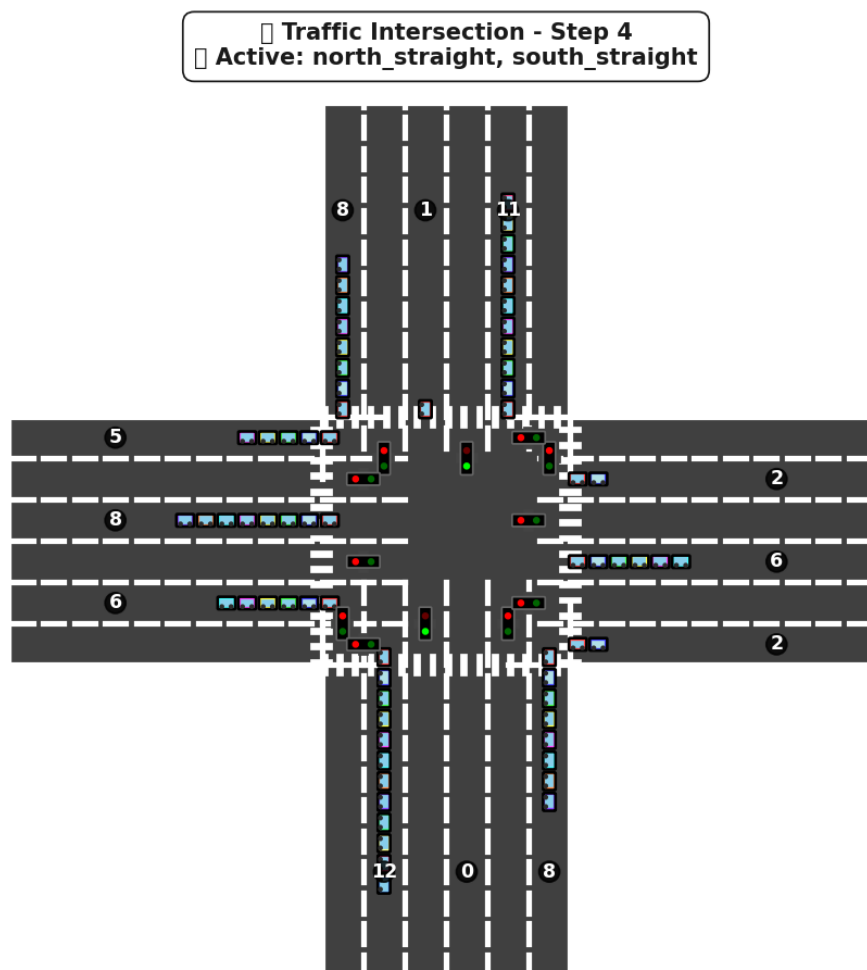
Each metric was plotted with its mean curve and a shaded area representing ± 1 standard deviation, as shown. The following summary presents the average standard deviation across all episodes:

- **Reward:** 21,272.13
- **Vehicles Passed:** 129.08
- **Average Wait Time:** 6.65 seconds

These results demonstrate that, despite initial fluctuations, the agent's performance stabilizes over time with reduced variance, indicating reliable convergence and learning stability.

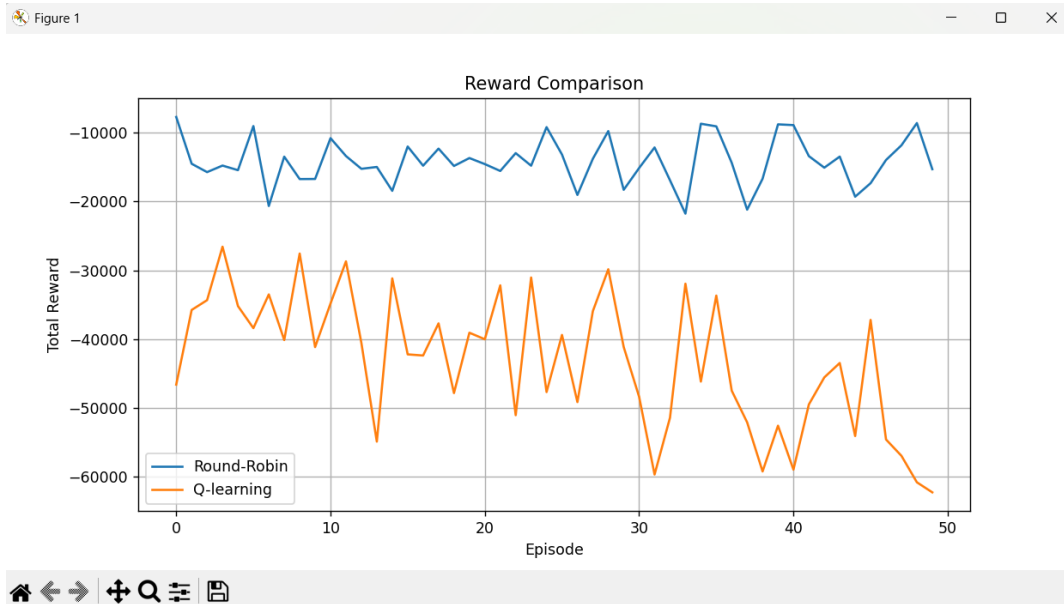
4.2. Visualization

Visualization of the simulated intersection at time step 4. The currently active phase is north_straight and south_straight, as indicated by the green traffic lights facing those directions. Vehicles are represented as colored rectangles, queued in their respective lanes. The white numbered circles at the edge of each lane display the real-time queue length. The graphical simulation enables intuitive tracking of traffic flow and congestion, providing visual validation of the agent's decision-making process.



5. Discussion

The initial implementation of our smart traffic light control system using a basic Q-learning setup yielded non optimal results:



The agent performed poorly in early training episodes, with high reward variability and overall performance consistently worse than the Round-Robin baseline. This outcome highlighted several limitations in the original model, including an overly rough reward function, limited state representation, and an inefficient action space that did not properly reflect real-world traffic dynamics. Therefore, we made a series of focused improvements aimed at enhancing both the learning stability and policy quality of the agent.

Through the development of our smart traffic light control system, several targeted modifications were introduced to improve the stability and performance of the Q-learning agent. These adjustments were motivated by the need to handle increasing traffic complexity while ensuring that the learning process converges toward efficient policies.

One of the key changes was the modification of queue length categories. Initially, each lane's queue was discretized into 4 categories based on rough thresholds. These bins were adjusted to cover a wider range of vehicle counts,



allowing the agent to better distinguish between moderate and severe congestion. This refinement improved the agent's sensitivity to varying traffic levels and led to more informed decision-making.

We also redesigned the reward function to capture a broader set of objectives. Beyond average waiting time and queue length penalties, we combined stronger penalties for overcrowding (queues > 35 cars) and severe congestion (queues > 45 cars), as well as throughput rewards and a light bonus for managing underloaded lanes. These changes promoted more balanced traffic flow across all approaches and discouraged neglecting certain directions.

To further improve the learning stability, the number of training episodes was increased to 300, and a more gradual epsilon decay factor ($\epsilon_{\text{decay}} = 0.995$) was applied. This allowed the agent to maintain a higher level of exploration over time, which helped reduce early-stage volatility and prevented premature convergence to suboptimal policies. Over multiple episodes, this led to smoother learning curves and a consistent rise in both reward scores and the number of vehicles processed.

Another important enhancement was the change in action selection granularity. Previously, the agent could select a number of cars to process from 0 to 20 in unit steps, which resulted in an unnecessarily large action space with limited interpretability. We replaced this with a discrete set of predefined vehicle batches: {0, 5, 10, 15, 20, 25, 30, 35}. This new structure strikes a balance between expressiveness and tractability, allowing the agent to regulate flow rates more effectively.



6. Conclusion

The development of our project highlighted both the potential and the challenges of applying reinforcement learning to real-world-inspired traffic control.

Through building a custom simulation of a four-way intersection and implementing a Q-learning agent, we explored how an autonomous system can learn to manage traffic lights based on dynamic conditions such as queue lengths and waiting times.

Our initial results clearly showed that a naive model—with limited reward shaping and overly simplistic discretization—was insufficient: the agent failed to outperform a basic Round-Robin controller. This led us to a series of focused improvements, including redesigning the reward function, redefining queue categories, expanding the range of vehicle discharge options, and adjusting the exploration policy to promote more stable learning.

These changes significantly improved the system's performance. The agent learned to make more efficient decisions, reducing average waiting time and increasing throughput over time. The visual simulation confirmed that the agent adapted its behavior in response to congestion patterns, favoring empty directions when needed and avoiding traffic buildup in high-density lanes.

Overall, our project demonstrates that reinforcement learning can form a strong basis for smart traffic light control when combined with thoughtful environment design, appropriate reward structures, and sufficient training scope.



References:

- [1] Bakker, B., Whiteson, S, Kester, L. and Groen, F. (2010). Traffic Light Control by Multiagent Reinforcement Learning Systems. *Springer-Verlag Berlin Heidelberg*, pp. 475–510.
- [2] Liang, X., Du, X., Wang, G. and Han, Z. (2018) Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks. *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*.
- [3] Wei, H., Yao, H., Zheng, G. and Li, Z. (2018). IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. Association for Computing Machinery. <https://doi.org/10.1145/3219819.3220096>
- [4] Wu, T., Zhou, P., Liu, K., Yuan, Y., Wang, X., Huang, H., and Wu, D. (2020). Multi-Agent Deep Reinforcement Learning for Urban Traffic Light Control in Vehicular Networks. *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*, VOL. 69, NO. 8.
- [5] Mousavi, S., Schukat, M., Howley, E. (2017). Traffic light control using deep policygradient and value-function-based reinforcement learning. *IET Intell. Transp. Syst.*, 2017, Vol. 11 Iss. 7, pp. 417-423
- [6] Mance, E., Stephanie, S. (1996). Reinforcement Learning: A Tutorial.
- [7] Li, Y. (2018) DEEP REINFORCEMENT LEARNING: AN OVERVIEW. <https://arxiv.org/abs/1810.06339>
- [8] Arulkumaran, K., Deisenroth, M., Brundage, M. & Bharath, A. (2017). Deep Reinforcement Learning - A brief survey. *IEEE Signal Processing Magazine* Vol 34, pp. 26-38
- [9] Gupta, J., Egorov, M., & Kochenderfer, M. (2017). Cooperative Multi-agent Control Using Deep Reinforcement Learning. *Autonomous Agents and Multiagent Systems* pp. 66-83.
- [10] Awan, A. (2022). An Introduction to Q-Learning: A Tutorial For Beginners. <https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial>