CS354N Project

Computational Intelligence Lab

Traffic Signal Control Using

Reinforcement Learning

Team 19

Team Members

Abhinav Kumar
Yatharth Gupta
Rohit Dhanotia

Problem Definition

Traffic congestion stands as a pervasive problem worldwide, leading to economic losses, increased pollution, and a general decline in the quality of life within urban environments. Traditional traffic signal control systems often rely on pre-timed sequences or simple sensor-based adjustments, which struggle to adapt effectively to the dynamic and unpredictable nature of real-world traffic patterns.

This project explores the application of Reinforcement Learning (RL) to develop intelligent traffic signal control systems. RL, a subset of machine learning, focuses on training agents to make optimal decisions through interaction with a complex environment. In traffic signal control, the RL agent learns to optimize traffic flow by adjusting signal timings in response to real-time traffic conditions.

Importance and Usefulness

The potential benefits of RL-based traffic signal control are vast:

- **Reduced Congestion:** Intelligent RL agents can learn to identify critical traffic patterns, optimizing signal sequences to minimize vehicle waiting times.
- **Enhanced Traffic Flow:** Improved flow leads to smoother traffic, potentially reducing accidents and improving travel times.
- **Environmental Benefits:** Better traffic management translates to reduced vehicle idling times, leading to decreased emissions.
- **Adaptability:** RL agents are capable of continuously learning and adapting to changing traffic conditions, providing a more robust solution than pre-programmed systems.

Methodology

This project utilizes the Simulation of Urban MObility (SUMO) environment, a leading open-source traffic simulator. SUMO provides a realistic platform to design and evaluate RL-based signal control strategies. We will explore and compare several cutting-edge RL techniques:

Deep Q-Network (DQN)

Deep Q-Network (DQN) is a powerful algorithm in the field of reinforcement learning. It combines the principles of deep neural networks with Q-learning, enabling agents to learn optimal policies in complex environments.

The DQN algorithm follows a deep neural network-based approach to learn and optimize action-value functions. The working process can be summarized as follows:

- **State Representation**: Convert the current state of the environment into a suitable numerical representation, such as raw pixel values or preprocessed features.
- **Neural Network Architecture**: Design a deep neural network, typically a convolutional neural network (CNN), that takes the state as input and outputs action-values for each possible action.
- **Experience Replay**: Store the agent’s experiences consisting of state, action, reward, and next state tuples in a replay memory buffer.
- **Q-Learning Update:** Sample mini-batches of experiences from the replay memory to update the neural network weights. The update is performed using the loss function derived from the Bellman equation, which minimizes the discrepancy between the predicted and target action-values.

- **Exploration and Exploitation**: Balance exploration and exploitation by selecting actions either greedily based on the current policy or stochastically to encourage exploration.
- **Target Network**: Use a separate target network with the same architecture as the main network to stabilize the learning process. Periodically update the target network by copying the weights from the main network.
- **Repeat Steps 1 to 6**: Interact with the environment, gather experiences, update the network, and refine the policy iteratively until convergence.

Advantage Actor Critic (A2C)

In the actor-critic framework, an agent (the “actor”) learns a policy to make decisions, and a value function (the “Critic”) evaluates the actions taken by the Actor. Simultaneously, the critic evaluates these actions by estimating their value or quality. This dual role allows the method to strike a balance between exploration and exploitation, leveraging the strengths of both policy and value functions.

Roles of Actor and Critic

- **Actor**: The actor makes decisions by selecting actions based on the current policy. Its responsibility lies in exploring the action space to maximize expected cumulative rewards. By continuously refining the policy, the actor adapts to the dynamic nature of the environment.
- **Critic**: The critic evaluates the actions taken by the actor. It estimates the value or quality of these actions by providing feedback on their performance. The critic’s role is pivotal in guiding the actor towards actions that lead to higher expected returns, contributing to the overall improvement of the learning process.

Policy gradients with the Advantage function:

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an algorithm in the field of reinforcement learning that trains a computer agent’s decision function to accomplish difficult tasks. PPO is classified as a policy gradient method for training an agent’s policy network.

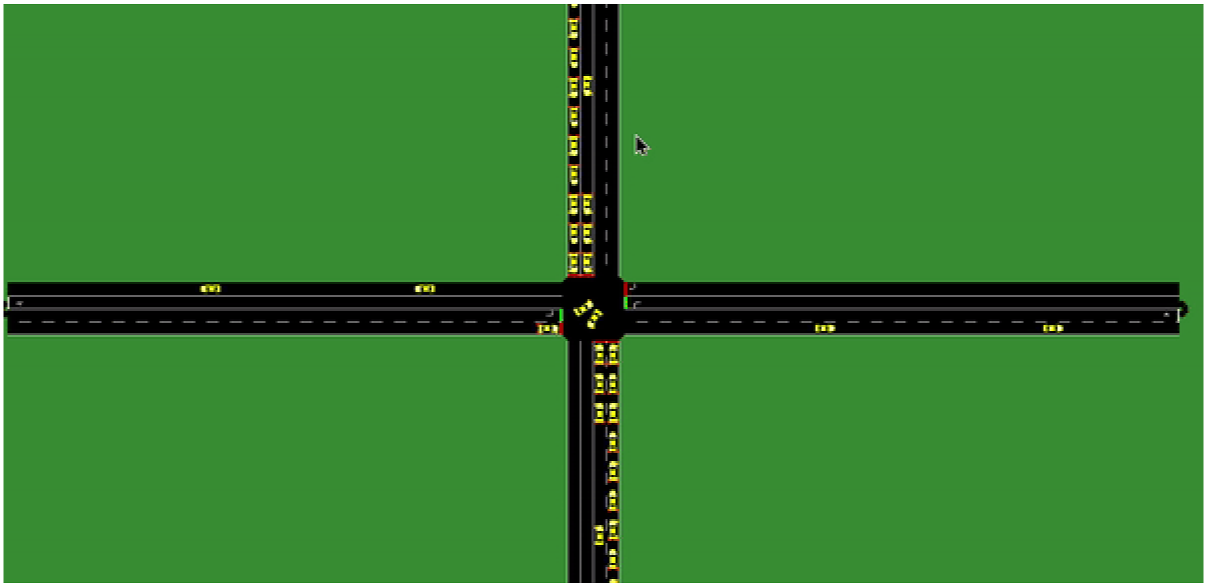
Here’s a brief overview of how PPO works:

- **Policy Gradient Methods**: PPO is based on policy gradient methods, which directly optimize the policy function that maps states to actions. This is in contrast to methods that estimate value functions.
- **Objective Function**: PPO aims to maximize the expected cumulative reward obtained from interacting with the environment. This is typically done by maximizing a surrogate objective function that approximates the policy improvement.
- **Clipped Surrogate Objective**: One of the key features of PPO is the clipped surrogate objective, which prevents large policy updates that could lead to catastrophic outcomes. By clipping the ratio between the probability of actions under the new policy and the old policy, PPO ensures that the policy update stays within a safe range.

- **Multiple Epochs and Mini-Batch Updates**: PPO typically involves multiple epochs of interaction with the environment, during which trajectories are collected. These trajectories are then used to compute the surrogate objective function, which is optimized using mini-batch updates.
- **Value Function Estimation**: PPO often incorporates value function estimation to reduce variance in the gradient estimates. This helps stabilize training and improve sample efficiency.
- **Parallelization**: PPO can be parallelized to accelerate training by collecting trajectories from multiple instances of the environment simultaneously.

Environment

We make a single intersection at first with four lanes, two for incoming and two for outgoing traffic each.



State and Action Space Representation

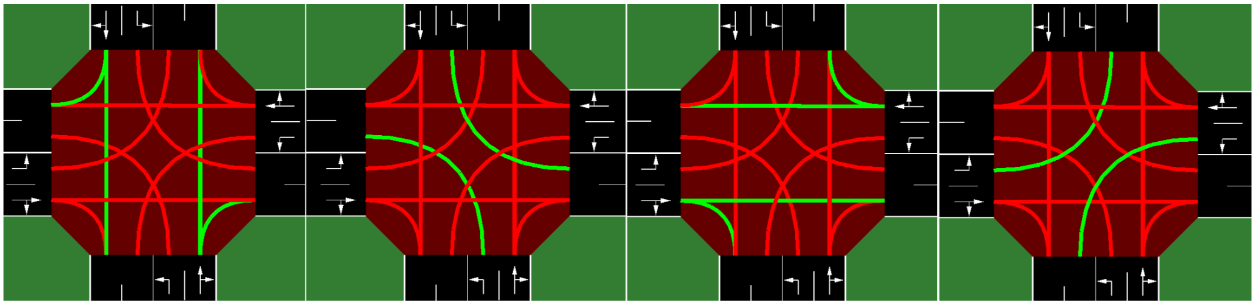
Observation:
obs = [phase_one_hot, min_green, lane_1_density,...,lane_n_density, lane_1_queue,...,lane_n_queue]

- phase_one_hot is a one-hot encoded vector indicating the current active green phase
- min_green is a binary variable indicating whether min_green seconds have already passed in the current phase
- lane_i_density is the number of vehicles in incoming lane i divided by the total capacity of the lane
- lane_i_queueis the number of queued (speed below 0.1 m/s) vehicles in incoming lane i divided by the total capacity of the lane

Action Space:

The action space is discrete. Every ‘delta_time’ seconds, each traffic signal agent can choose the next green phase configuration.

There are |A| = 4 discrete actions, corresponding to the following green phase configurations:



Everytime a phase change occurs, the next phase is preceded by a yellow phase lasting yellow_time seconds.

Reward:

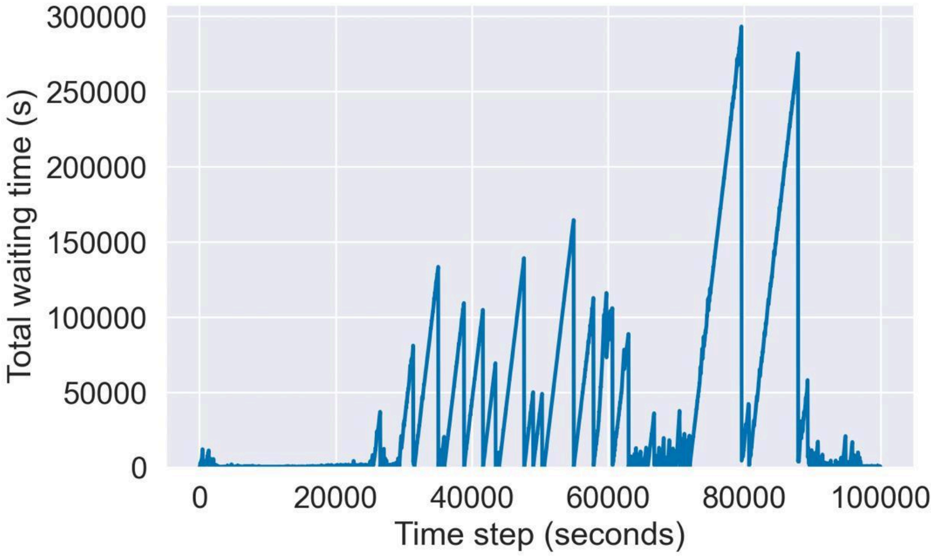
The default reward function is the change in cumulative vehicle delay:

$$\boxed{r_t = D_{a_t} - D_{a_{t+1}}}$$

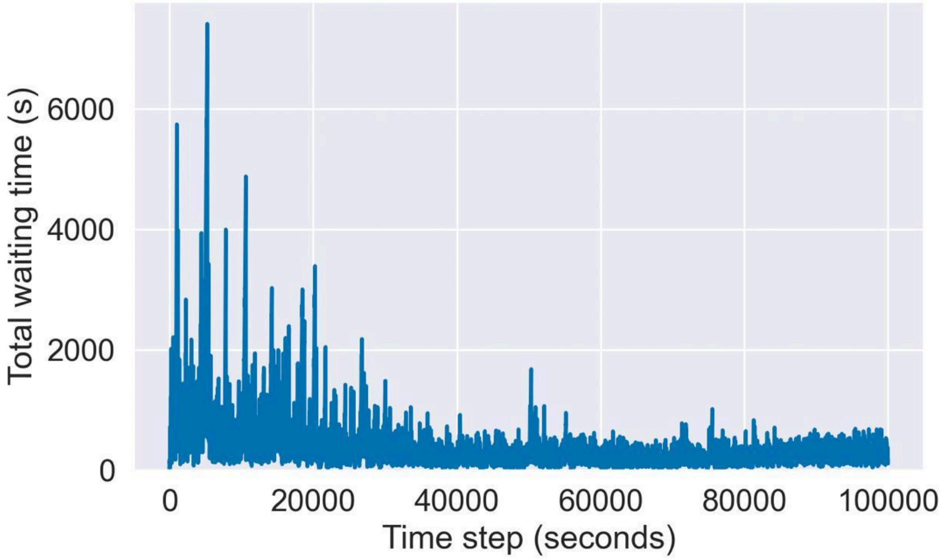
That is, the reward is how much the total delay (sum of the waiting times of all approaching vehicles) changed in relation to the previous time-step.

Comparison

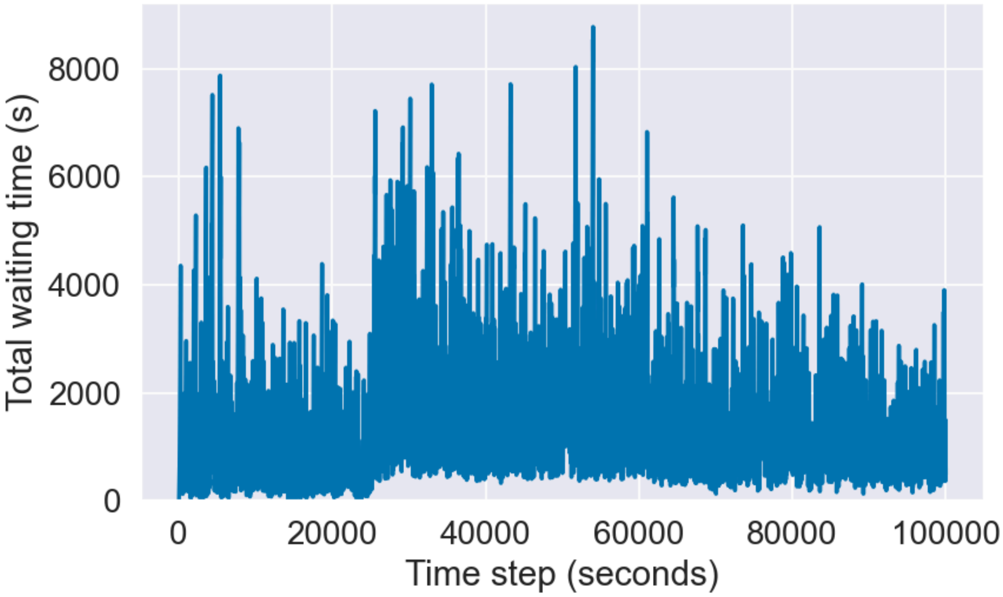
a) Deep Q-Networks (DQN)



b) Advantage Actor-Critic (A2C)



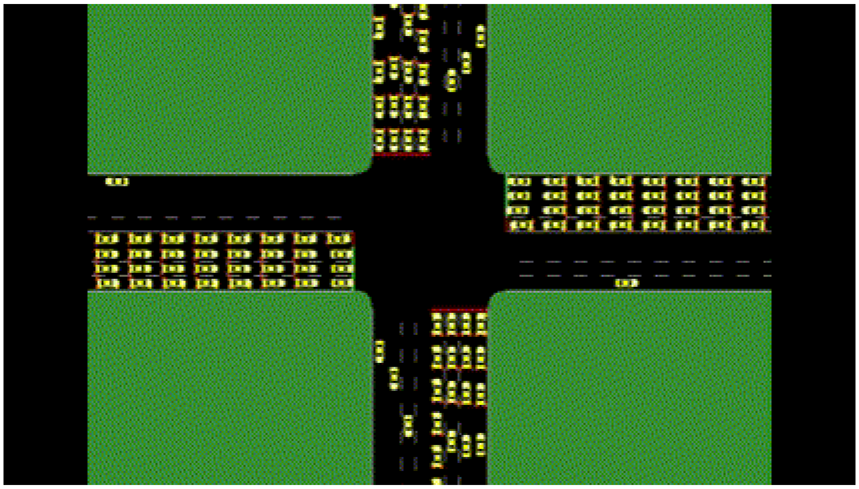
c) Proximal Policy Optimization (PPO)



More Environments

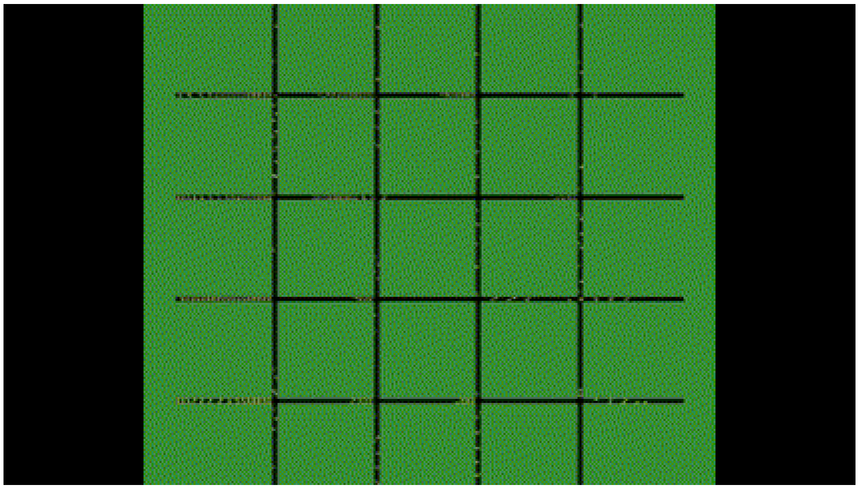
Big Intersection

We enhance the single intersection with eight lanes, four for incoming and four for outgoing traffic each.



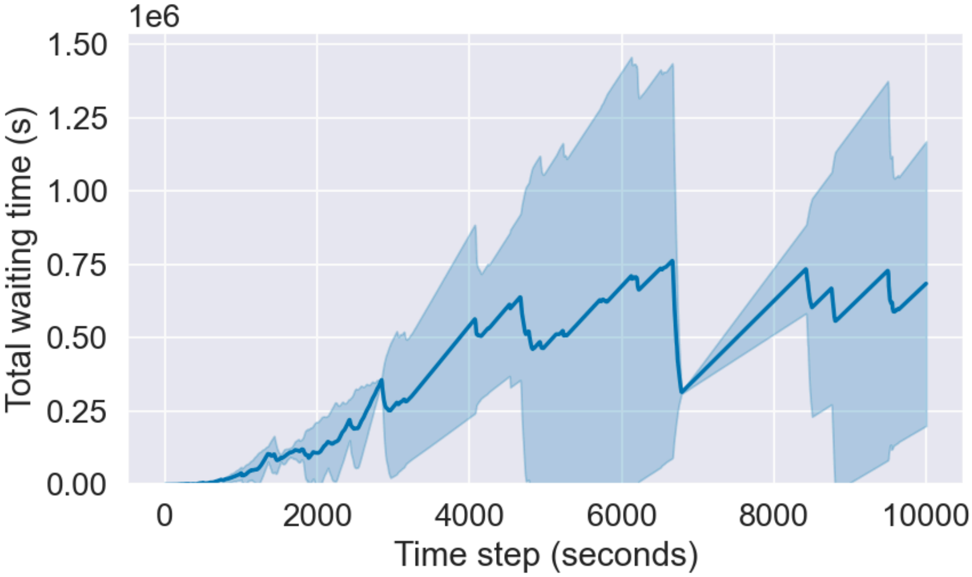
4X4 Grid

We create a city simulation with 16 intersections in 4x4 pattern and train agents to control the traffic for a city.

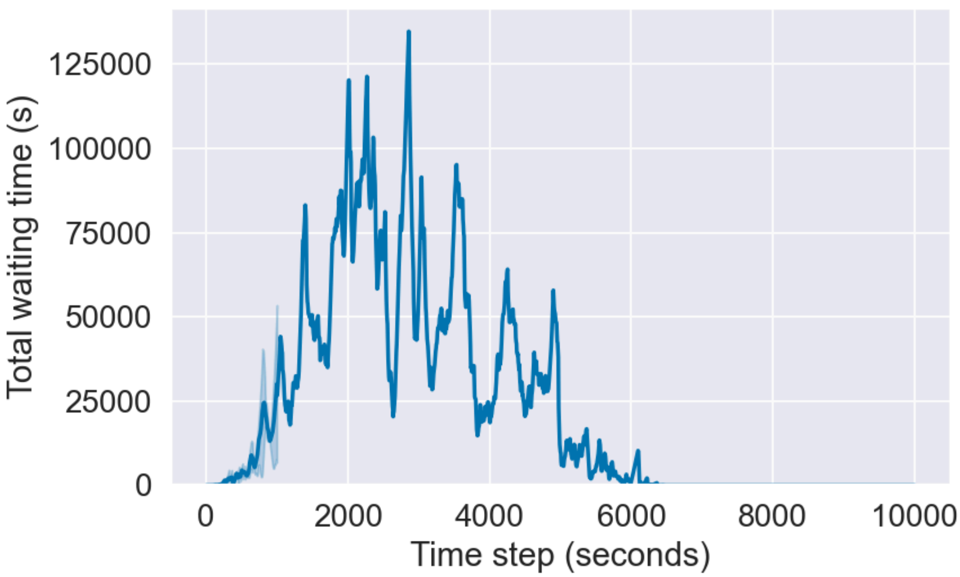


Results on Big Intersection

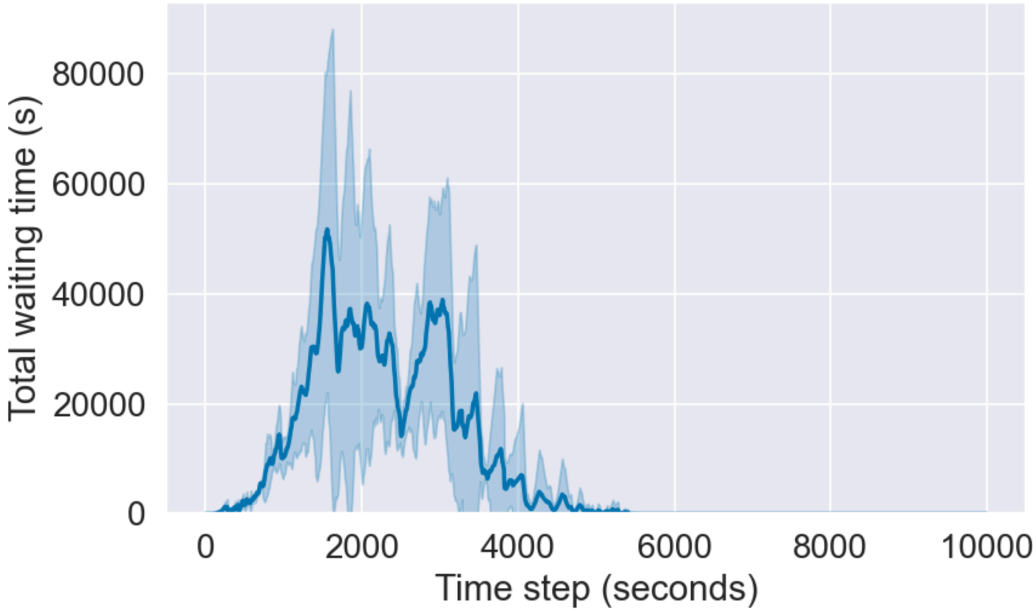
a) Deep Q-Network (DQN)



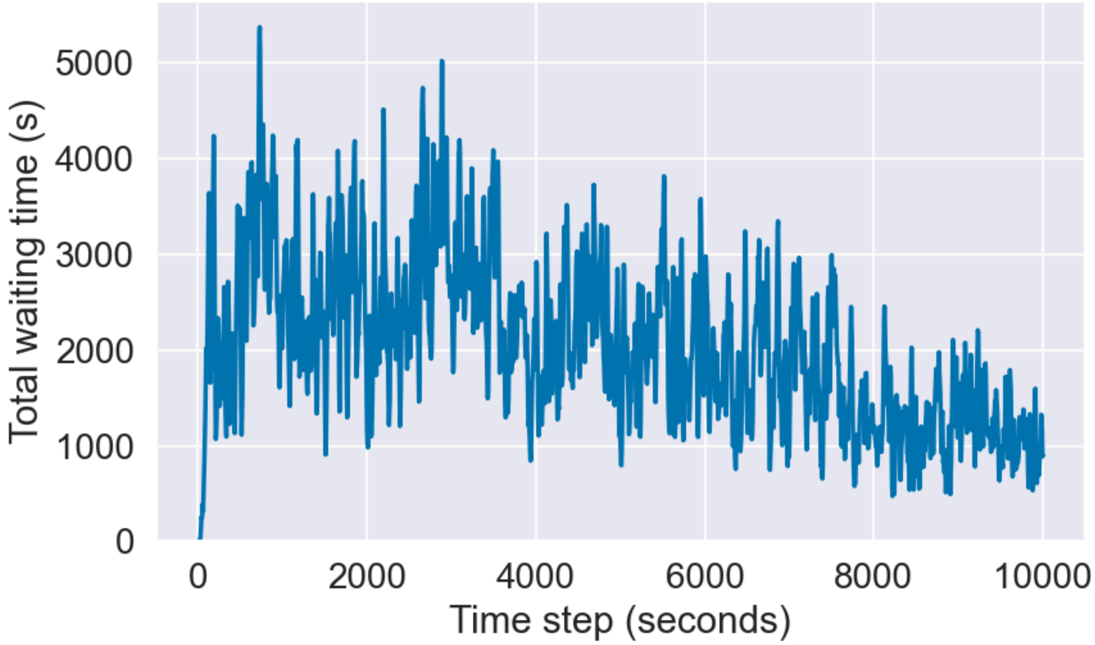
b) Proximal Policy Optimization (PPO)



c) Advantage Actor Critic (A2C)



4x4 Environment



Conclusion

We trained an agent for a traffic simulation reinforcement learning (RL) environment using the SUMO package. We trained agents with algorithms like Deep Q Networks (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO). Thorough hyperparameter tuning informed our final agent training on three environments (a simple intersection, a larger intersection, and a 4x4 city grid). A2C emerged as the superior algorithm, demonstrating rapid convergence and strong performance across all environments.

Through this project we learnt about Reinforcement Learning Algorithms, Modeling environments for creating RL agents and Reward Functions. We also got hands-on experience with training RL Agents and applying them for Real World Applications.