# Multi Agent Reinforcement Learning with Highway-Env

David Scarin

Francisco Silva

Tomás Rodrigues

Modelação e Simulação, M.IA, 2024/2025

Submission Date: January 4, 2025

# Contents

# Abstract

Abstract

# 1. Introduction

Autonomous driving has for long been a relevant and important field of study. With the emergence of deep learning techniques and innovations regarding computing power and technology in vehicles, the field has only grown more prevalent.

The problem of different vehicles driving on the road together and while obeying the same set of rules can be modeled as an agent-based simulation, where each vehicle has its own set of goals but has to obtain information from the environment and act in accordance to it, even considering how other agents (vehicles) act within the same environment.

The objective of this project is to simulate and evaluate reinforcement learning algorithms that enable autonomous vehicles to perform complex driving tasks, such as merging, lane changing, and overtaking, while ensuring safety and optimizing traffic flow.

Specifically, the project will focus on **simulating scenarios at road intersections where multiple autonomous vehicles must determine the order of crossing and appropriate speeds, aiming to maximize traffic efficiency and minimize collision risks.**

This will be achieved using an agent-based simulation framework - ***Highway Env***[1] - where a group of four agents, each trained with different Deep Reinforcement Learning (DRL) policies, will face dynamic environmental changes. The agents will be tested under various combinations of environmental variables (e.g., traffic density,vehicle speed) to evaluate their behavior and decision-making processes in real-time.

This experimental setup will create a Multi-Agent System (MAS) within a dynamic environment, concentrating on decision-making strategies for autonomous driving.

Through this project, we aim to address critical questions regarding the scalability and robustness of DRL models in autonomous driving.

In particular, the research will explore how well these algorithms generalize to unseen environments — a key challenge for deep learning-based approaches.

The insights gained can inform the real-world applicability of DRL for autonomous vehicles, especially in unpredictable or changing environments.

# 2. Literature Review

Literature Review

# 3. Problem Formalization

## 3.1 Models of Decision Support Considered

This simulation project can be classified under multiple decision support model categories, as it touches upon different aspects of decision-making processes. Here's how the project fits into each of the categories:

1. **Descriptive**

   This simulation contains a strong descriptive component. During the initial stages, the agents are trained using DRL algorithms, and their behaviors are observed and recorded in a baseline scenario. This phase is focused on describing and understanding how the autonomous agents make decisions (e.g., crossing order, speed adjustment) based on the environment they are exposed to. The descriptive element helps us analyze how agents behave without interference and forms the basis for comparison with later stages.

2. **Normative**

   This simulation implicitly contains a normative aspect. The desired outcomes for autonomous vehicle behavior (e.g., maximizing traffic flow while minimizing collisions) are rooted in optimal decision-making criteria. Although the project does not directly implement normative models, the success criteria for the agents—safe and efficient decision-making—are benchmarks derived from an ideal (normative) vision of how the system should behave in various traffic conditions.

3. **Predictive**

   The simulation can be seen as predictive in its later stages. After training the agents, one of the goals is to predict how the multi-agent system (MAS) will perform under different traffic fluxes and lane configurations. The project aims to simulate a variety of scenarios (perturbations) and observe how the agents adapt. Through this, the system's future performance in unseen scenarios can be forecast, making predictive modeling a key aspect of the evaluation.

4. **Prescriptive**

   The simulation has a prescriptive element. The goal of DRL in this context is to prescribe optimal actions to agents in real-time traffic scenarios. By learning policies that guide the agents' decisions (e.g., what crossing order to take, when to accelerate or decelerate), the system effectively prescribes the best possible actions to optimize traffic flow and minimize collision risk. As the agents are trained to follow the best course of action in specific conditions, the project fits well within the prescriptive decision support model.

5. **Speculative**

   The simulation could involve some speculative modeling, though this is not a primary focus. Testing agent performance in hypothetical situations would fall under this category, helping assess how flexible and adaptive the MAS is. However, speculative modeling is a minor aspect, as the primary focus is on concrete variations in traffic and lane configurations.

Based on these observations, the primary classification of this simulation project would be **descriptive**, **predictive**, and **prescriptive**, as it involves understanding current decision-making, predicting agent behaviors under different conditions, and prescribing optimal actions for traffic optimization and safety.

Normative models are more implicit, serving as ideal benchmarks, while speculative elements might arise if the project explores hypothetical or extreme scenarios.

In addition to the decision support model categories, the simulation model can be classified as:

**Dynamic**: The model evolves over time as agents continuously interact with the environment, make decisions, and adapt to changing conditions at road intersections. The system state changes as traffic flows and lane configurations adjust.

**Stochastic**: The simulation includes elements of randomness, such as variations in traffic flow, vehicle arrival patterns, and potential uncertainties in agent behavior. This introduces probabilistic outcomes and variability across different simulation runs.

**Discrete**: The model operates in discrete time steps, where the agents' actions (e.g., crossing intersections, adjusting speed) are evaluated at specific intervals. Each decision is made at distinct time points, typical in agent-based simulations where agent behaviors are updated step-by-step.

These classifications emphasize the dynamic, probabilistic, and step-wise nature of this agent-based simulation, where individual agents make decisions over time.

## 3.2 The System as an Agent-Based Simulation

As an **agent-based simulation**, this simulation's behavior emerges from the interactions of its individual agents, our autonomous vehicles. These vehicles represent entities capable of perceiving the environment and adapting their actions accordingly.

**Environment Entities**

In this context, the primary entities of the system can be described as:

1. **Autonomous Vehicles (Agents):**

   Each autonomous vehicle represents an agent in the simulation with decision-making capabilities. These vehicles are created, move around, change speed, and interact with other agents. They may enter and leave the system as they pass through the intersection.

   Attributes: Speed, position, direction, assigned policy (decision-making model), current lane.

   Resources they compete for: Road space, crossing priority, lanes.

2. **Intersection (Road Infrastructure)**:

   The road intersection is a static entity but a key part of the system. It defines where vehicles meet and interact. Different types of lane configurations or intersection designs can be applied.

   Attributes: Number of lanes, intersection layout.

   Resources they compete for: Lane capacity (the number of vehicles that can use a lane or section of road at a time).

3. **Traffic Flow:**

   Represents the overall movement of vehicles through the system. This is a dynamic entity in terms of the rate at which vehicles arrive at and depart from the intersection.

   Attributes: Arrival rate of vehicles, traffic density, vehicle types (e.g. vehicles, ego-vehicles).

   Resources they compete for: Access to the intersection, road segments.

4. **Agent Policies (Decision-Making Models):**

   Each autonomous vehicle (agent) operates based on a decision-making policy (learned behavior from DRL). These policies guide how each vehicle responds to other vehicles and environmental factors.

   Attributes: Learned policy, decision rules, reward function (for reinforcement learning).

   Resources they compete for: Computational resources for decision-making (though implicit in the model), control over vehicle behavior.

**System Variables**

Variables in the system represent pieces of information that reflect characteristics of the entire system, not of specific entities. These variables are either directly influenced by the system dynamics or serve as global parameters for the simulation.

1. **Exogeneous Variables**
   (a) **Non-Controllable**
      i. **Traffic Density (Arrival Rate of Vehicles):** Indicates the rate at which vehicles enter the simulation, typically measured as vehicles per time unit (e.g., vehicles per minute). Role: It influences system congestion and vehicle interactions at the intersection.
      ii. **Intersection Configuration (Lane Layout):** Represents the structure of the intersection, such as the number of lanes or the presence of dedicated turn lanes. Role: Changes in this variable affect how vehicles maneuver and interact with each other.

   (b) **Controllable**
      i. **Number of Controlled Vehicles**: Specifies the number of ego-vehicles that can be simultaneously managed by the agent's policy.
      ii. **Agent-Implemented Policy**: Enables the definition of the agent's policy and algorithm for each simulation scenario.
2. **Endogeneous Variables**
   (a) **Controlled Vehicles Average Speed:** This is a system-level variable that measures on average, the vehicles speed when crossing the intersection.

      Role: It reflects the system's efficiency and can be used to assess the performance of different policies.

   (b) **Collision Rate:** A key variable that reflects the number of collisions or near-collisions occurring in the system.

      Role: A measure of the system's safety, which influences the evaluation of agent decision-making policies.

   (c) **System Throughput:** The number of vehicles successfully passing through the intersection over a given period.

      Role: A variable that reflects the overall efficiency and capacity of the system, indicating how well it handles different traffic conditions.

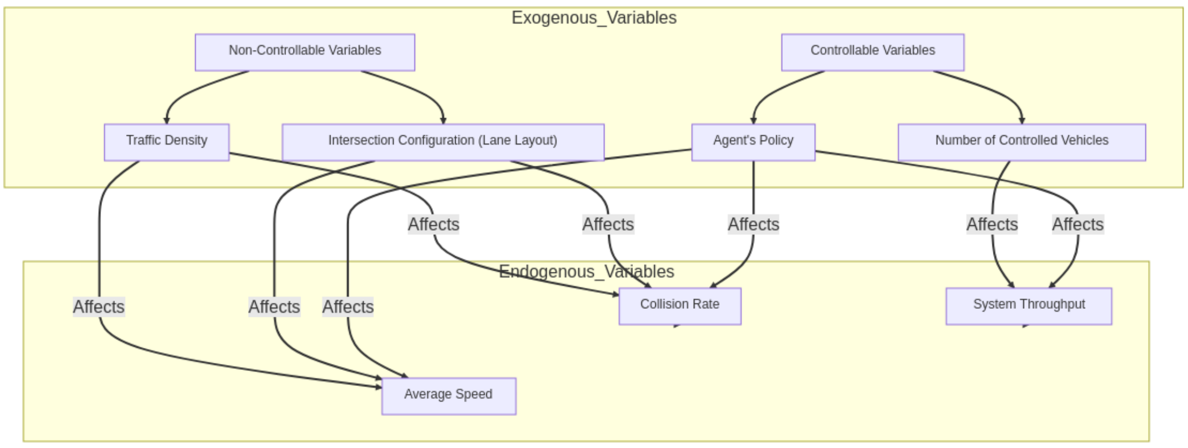


Figure 3.1: System Variables

**State of the System**:

The state of the system at any given time would include a collection of variables such as the number of vehicles in the system, traffic density, the intersection configuration, and the real-time position and speed of each vehicle.

The state contains all the necessary information to describe the system's current dynamics and predict future behaviors.

These elements collectively define how the simulation operates, how decisions are made, and how the performance of the system is evaluated.

## 3.3 Agent Training Algorithms and Policies to be Tested

The simulation project will consist of three groups, each containing four agents trained using Deep Reinforcement Learning (DRL) techniques based on the **DQN** and **PPO** algorithms, employing different policies (*MlpPolicy*, *CnnPolicy*, and *Social Attention Mechanisms*).

The behavior of these agents and their respective policies will be evaluated against a **Baseline Scenario**, where agents operate without any guiding policy and perform random actions.

### 3.3.1 Training Algorithms

#### 1.Deep Reinforcement Learning using DQN

The Deep Q-Network (DQN) algorithm is a reinforcement learning approach designed to enable agents to learn optimal policies by estimating the value of action-state pairs [5]. This is achieved by leveraging neural networks to approximate Q-values, which guide the agent's actions to maximize long-term rewards.

To understand DQN, we first define the concept of a Markov Decision Process (MDP). An MDP is characterized by states $s$, actions $a$, rewards $r$, and transitions $P(s' \mid s, a)$. The goal in an MDP is to maximize the cumulative reward (also known as the return) over time.

The Q-function represents the expected return of taking a specific action $a$ in a given state $s$, followed by an optimal sequence of actions:

$$Q(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') \mid s_t = s, a_t = a\right]$$

Here: - $Q(s,a)$ is the value of taking action $a$ in state $s$, - $r_t$ is the immediate reward received, - $\gamma$ is the discount factor, where $0 \leq \gamma \leq 1$, balancing the importance of immediate and future rewards.

The Q-function is recursively defined using the Bellman equation:

$$Q(s,a) = r + \gamma \max_{a'} Q(s', a')$$

In practice, DQN approximates the Q-function using a deep neural network parameterized by $\theta$:

$$Q(s,a;\theta) \approx Q^*(s,a)$$

where $Q^*(s,a)$ represents the optimal Q-function.

The neural network learns the parameters $\theta$ by minimizing the difference between predicted and target Q-values.

#### 2.Deep Reinforcement Learning using PPO

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that aims to improve the policy of an agent by balancing exploration and exploitation[7]. PPO achieves this by using a clipped objective function to prevent large policy updates, ensuring stable learning.

**Policy Function**

The policy function $\pi_\theta(a \mid s)$ defines the probability of taking action $a$ in state $s$ according to the policy parameters $\theta$:

$$\pi_\theta(a \mid s) = \mathbb{P}(a \mid s; \theta)$$

Here, $\theta$ parameterizes the policy network. In PPO, the goal is to maximize cumulative rewards while ensuring stability by limiting updates using a clipped objective function.

**PPO Objective Function**

The objective function used in PPO is a clipped surrogate objective, which aims to optimize the policy by maximizing the expected advantage, but with a constraint on the update size:

$$L(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

Where: - $r_t(\theta)$ is the probability ratio between the new and old policy, defined as:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

- $\epsilon$ is the clipping parameter, controlling how much the policy is allowed to change. - $A_t$ is the advantage estimate at time step $t$, which measures how much better an action is compared to the average action.

### 3.3.2 Agent Policies

We will compare agents' behavior based on four distinct policies:

1. **Baseline No Policy**
2. **MlpPolicy (Multi-Layer Perceptron Policy)**

   The *MlpPolicy* utilizes a multi-layer perceptron neural network to process environmental observations and make decisions. This policy focuses on key input features, such as speed, distance to other vehicles, and intersection layout, allowing agents to act based on their immediate surroundings.

   **Scenario**: Agents following the *MlpPolicy* will navigate the intersection by leveraging their observed features, applying learned behaviors to respond effectively to dynamic traffic conditions.

   **Objective**: Assess the performance of *MlpPolicy* agents in terms of efficiency (e.g., throughput, wait times) and safety (e.g., collision rates) compared to agents using other policies. Analyze the influence of input features on their decision-making processes.
3. **CnnPolicy (Convolutional Neural Network Policy)**

   The *CnnPolicy* employs convolutional neural networks to process visual and spatial data, making it particularly effective in scenarios where visual inputs (like grid representations of the intersection) are crucial for decision-making.

   **Scenario**: Agents using the *CnnPolicy* will interpret complex visual representations of their environment, allowing them to make informed decisions regarding lane changes, merging, and crossing orders.

   **Objective**: Evaluate the performance of *CnnPolicy* agents in diverse traffic patterns, focusing on their ability to recognize spatial configurations and respond appropriately to traffic dynamics.
4. **Social Attention Mechanisms Policy**

   This policy integrates social attention mechanisms, allowing agents to observe and interpret the behaviors and intentions of nearby vehicles.

   Social Attention Mechanisms in Deep Reinforcement Learning (DRL) are inspired by how humans or animals interact within a social environment. In the context of multi-agent systems, the idea is to integrate social information, allowing agents to pay attention to the behavior and intentions of other agents.[8] This concept helps improve cooperation, communication, and coordination between agents in environments where agents need to share space and interact with each other, such as autonomous vehicles in a traffic intersection.

**Attention Mechanisms** In deep learning, attention mechanisms allow models to focus on important parts of the input data when making decisions, rather than processing everything equally[9]. This is especially useful in scenarios involving sequences or spatial relations, where certain elements might be more significant than others.

**Social Attention Mechanisms** Social attention mechanisms focus on enabling agents to learn which other agents (or parts of the environment) are most relevant to their actions. In the case of DRL with DQN (Deep Q-Network), these mechanisms could take into account the actions and states of surrounding agents when determining a given agent's own actions. Essentially, the agent learns a social context that goes beyond its immediate environment[2].

**State Representation with Social Attention** In a multi-agent setting, an agent's state at time $t$, $s_t$, may include not only its own environment but also the states of surrounding agents. Let the state of agent $i$ be represented as $s_i(t)$, and the full state of the environment, including information from other agents, can be written as:

$$s_t = \{s_i(t), s_{-i}(t)\}$$

where $s_{-i}(t)$ represents the states of all other agents in the environment at time $t$[**mnih2015human**].

**Social Attention Weights** A key idea in social attention is to compute attention weights that determine how much focus an agent should place on each neighboring agent. Let $\alpha_i$ be the attention weight for agent $i$. These weights are learned through a neural network that estimates the importance of each neighboring agent's state in the context of the decision-making process. A common approach is to use a weighted sum, where each agent's state is weighted according to its relevance:

$$\tilde{s}_t = \sum_{i \in N} \alpha_i s_i(t)$$

where $N$ is the set of neighboring agents, and $\alpha_i$ is the attention weight for agent $i$. This process enables each agent to selectively attend to those agents whose states or actions influence its own behavior.

**Modifying the Q-Function with Attention** The standard DQN Q-function estimates the expected return for each action given a state $s_t$. With the incorporation of social attention, we modify the Q-function to account for the social context:

$$Q(s_t, a_t) = \mathbb{E}\left[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta) \mid s_t, a_t\right]$$

where $s_t$ now includes the social context $\tilde{s}_t$, representing the attention-weighted sum of the agent's own state and the states of the other agents. The DQN network then estimates the Q-values with this enriched state.

**Multi-Head Attention Implementation** The attention mechanism utilizes multiple heads to compute context-sensitive embeddings, allowing the model to attend to different aspects of the input simultaneously. Each head processes the embeddings independently and outputs its own context-aware representation. The outputs from all heads are concatenated and projected to form the final embedding. The attention computation for each head is given by[4]:

$$\text{output} = \sigma\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

where:

$$Q = L_q(e_0),\ K = L_k(e_i),\ V = L_v(e_i)$$

In these equations: - $e_i$ denotes the encoded state of vehicle $i$, while $e_0$ corresponds to the ego-vehicle - $K$ (keys) represents descriptive features of surrounding vehicles - $V$ (values) contains the embedded features used to compute the final output - $L_q$, $L_k$, $L_v$ are shared linear projections applied to all vehicles' embeddings - $\sigma$ is the softmax activation, normalizing attention scores across all vehicles - $d_k$ represents the dimension of the key embeddings

The ego-vehicle query $Q$ interacts with keys $K$ to compute attention weights, prioritizing vehicles that are most relevant to the ego-vehicle's current context. This results in a weighted combination of values $V$. The outputs from each head are aggregated and passed through a linear layer to produce the final representation.

This approach helps agents develop cooperative strategies and optimize their interactions with other vehicles. The attention architecture was introduced to enable neural networks to discover inter-dependencies within a variable number of inputs[9].

**Scenario**: Agents utilizing this policy will actively monitor the behavior of surrounding vehicles, adapting their actions based on predicted movements and intentions of others.

**Objective**: Investigate the effectiveness of social attention in enhancing cooperation among agents, potentially leading to improved traffic flow and reduced collision rates. Compare the performance of agents using Social Attention Mechanisms against those following other policies to assess their overall effectiveness in a multi-agent environment.

## 3.4 Testing Framework for Operation Policies (Scenarios)

1. **Experimental Scenarios:** Each policy will be tested under various conditions, such as:
   - **High Traffic Density:** Increased number of vehicles entering the intersection simultaneously.
   - **Low Traffic Density:** Sparse vehicle presence to evaluate agent behavior in less congested environments.
2. **Group Configuration:** Each group, consisting of four agents using the same policy, will be tested in each scenario. This allows for direct comparisons between the different policies under identical traffic conditions.
3. **Performance Metrics:**
   (a) **Efficiency:** Metrics such as throughput (vehicles per time unit), and average speed of agents.
   (b) **Safety:** Collision rates, near-misses, and compliance with traffic rules.
   (c) **Adaptability:** How well agents generalize their learned behaviors to new traffic scenarios (e.g., changes in traffic density, lane configurations).

## 3.5 Global Key Performance Indicators (KPI)

The following Key Performance Indicators (KPIs) and decision criteria will be used to effectively assess the performance and effectiveness of the operational policies:

1. **Traffic Flow Efficiency**
   - **Metric:** Average vehicle throughput (Arrived vehicles per test episode)
   - This metric measures how well the intersection handles traffic. A higher throughput indicates that the system is efficiently managing vehicle movement.
2. **Safety Metrics**
   - **Metric:** Collision rate (Average number of collisions per test episode)

- This is a crucial indicator of how safe the intersection is for autonomous vehicles. Reducing collisions is a primary goal for any traffic management system.

3. **Travel Time through the Intersection**
   - **Metric:** Average travel time per vehicle (seconds)
   - Evaluating the time it takes for vehicles to cross the intersection helps assess the effectiveness of different policies in facilitating movement.

4. **Adaptability to Varying Traffic Conditions**
   - **Metric:** Performance variance (e.g., comparing metrics like wait times and throughput under different traffic densities)
   - This evaluates how well agents can adjust their behavior to different traffic conditions, ensuring the system's flexibility in dynamic environments.

# 4. Methodology

## 4.1 Approach

This project consists of three key stages, each with specific goals that contribute to the overall aim of developing a robust and adaptable Multi-Agent System (MAS) for managing autonomous vehicles at road intersections. The stages are as follows:

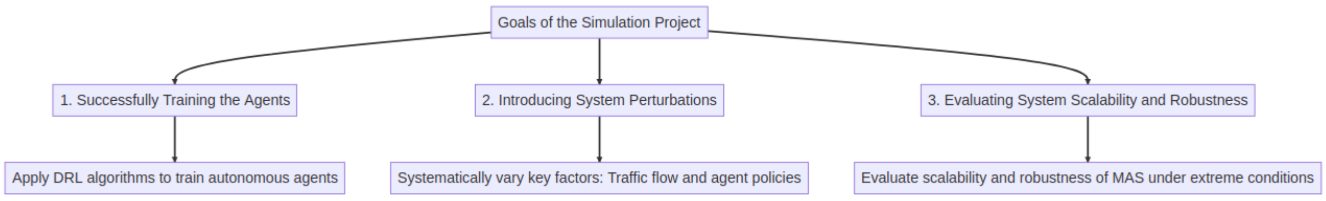

Goals of the Simulation Project
1. Successfully Training the Agents
2. Introducing System Perturbations
3. Evaluating System Scalability and Robustness
Apply DRL algorithms to train autonomous agents
Systematically vary key factors: Traffic flow and agent policies
Evaluate scalability and robustness of MAS under extreme conditions

Figure 4.1: Project Methodology

1. **Successfully train the agents**

   In this phase, we will apply selected Deep Reinforcement Learning (DRL) algorithms to train autonomous agents in navigating a road intersection.

   The agents will learn to make decisions related to intersection crossing order and speed control, with the goal of optimizing traffic flow and avoiding collisions.

2. **Introducing System Perturbations**

   At this stage, the goal is to understand how our different agent policies (policy refers to the strategy/mapping in each agent from state to action) perform when presented with different environment configurations. When it comes to the environment, we tested with two different traffic flow configurations, which we defined as **sparse** (4 ego vehicles, few other vehicles) or **dense** (4 ego vehicles, many other vehicles). The goal is to understand if the difference in configuration will alter the algorithm's performance and/or if any of the algorithms are more robust to the changes than others.

3. **Evaluate system scalability and robustness**

   Having trained our learning models, we can load them into the environment and use the approximated policy to make decisions. We run a select number of simulations use this policy and gather metrics from the environment, which are used to compare and evaluate each of the learning algorithms.

## 4.2 Successfully train the agents

### 4.2.1 Reinforcement Learning Training Process

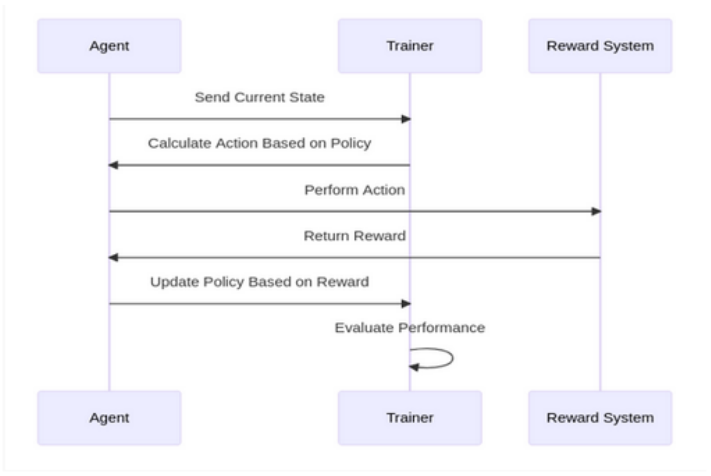


Figure 4.2: RL Training Process Sequence

The Training Process Sequence can be described as follows:

**1.Initialization**

- **Environment Setup**: An environment is defined (e.g., a traffic intersection using Highway-Env), which provides the observation and action spaces. The environment simulates the interactions of agents and their surroundings.
- **Observation Space**: This defines what the agent perceives at each timestep. For example:
  - In the traffic environment, the observation may include a grayscale image of the environment (shape: $(128, 64)$), stacked over 4 frames to provide temporal context.
  - The observations are preprocessed to include essential information, such as vehicle positions, velocities, and surrounding traffic.
- **Action Space**: The possible actions an agent can take at each timestep, such as accelerating, braking, or changing lanes in the traffic context.
- **Reward Function**: Rewards guide the agent's learning by providing feedback. The reward is carefully designed to encourage desired behaviors, such as:
  - Minimizing travel time.
  - Avoiding collisions.
  - Yielding or cooperating with other agents.

**2.Agent Initialization**

- An RL agent, such as one trained using the Deep Q-Network (DQN) or Proximal Policy Optimization (PPO) algorithm, is instantiated.
- The agent interacts with the environment through a defined policy. For example:
  - **DQN**: Uses a neural network to approximate the Q-function, mapping states and actions to expected future rewards.
  - **PPO**: Optimizes the policy directly, ensuring stable updates with a clipped objective function.

**3.Interaction with the Environment**

At each timestep:

- **Observation**: The agent observes the current state of the environment.
- **Action Selection**: Based on the observation, the agent selects an action using its policy (e.g., an $\epsilon$-greedy policy in DQN).
- **Environment Response**: The environment executes the action and provides:
  - The next state.
  - A reward signal.
  - A flag indicating whether the episode has ended.

**4.Experience Storage**

- The agent stores experiences as tuples $(s, a, r, s')$ in a replay buffer (for DQN) or uses the trajectory for on-policy updates (PPO).

**5.Policy Optimization**

- **DQN**:
  - A mini-batch of experiences is sampled from the replay buffer.
  - The Q-function is updated using the Bellman equation:

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a') \tag{4.1}$$

  - The agent minimizes the temporal difference error between predicted and target Q-values.
- **PPO**:
  - The agent computes the advantage function to estimate the relative value of actions.
  - Policy and value networks are optimized using the PPO loss:

$$L(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \operatorname{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) A_t\right)\right] \tag{4.2}$$

  - The clipping ensures stable updates.

**6.Evaluation and Improvement**

- At regular intervals, the agent's policy is evaluated in the environment to assess its performance.
- Metrics such as mean reward and train loss are used to gauge improvement.
- The best-performing model is saved during training for deployment.

**7.End of Training**

- After reaching the desired number of training timesteps, the final policy is saved.
- Post-training evaluations are conducted to ensure the agent generalizes well to unseen scenarios.

### 4.2.2 Practical Training Implementation - Configuration

The training of each agent using the "standard" DQN and PPO algorithms was conducted with **Stable-Baselines3**[6] within a custom multi-agent environment created using the **Highway-Env** library.

#### DQN and PPO with CnnPolicy

#### 1.Initialization

Environment Setup, Observation Space (Python coded):

```
env = make_vec_env(
    "intersection-v0",
    n_envs=n_cpu,
    vec_env_cls=SubprocVecEnv,
    env_kwargs={
        'config':{
            'initial_vehicle_count': 10,
            'controlled_vehicles': 4,
            'destination': 'o1',
            "observation": {
                "type": "GrayscaleObservation",
                "observation_shape": (128, 64),
                "stack_size": 4,
                "weights": [0.2989, 0.5870, 0.1140],  # weights for RGB
                   conversion
                "scaling": 1.75,
            },
        },
    }
)
```

Each environment comes with a default observation, which can be changed or customised using environment configurations. In the intersection environment, in order to use the images needed for CNN, the observation type must be defined as *GrayscaleObservation*.

The *GrayscaleObservation* is a $W \times H$ grayscale image of the scene, where $W$ and $H$ are set with the *observation_shape* parameter. The RGB to grayscale conversion is a weighted sum configured by the *weights* parameter. Several images can be stacked with the *stack_size* parameter.

The **Action Space** is configured as:

```
'action': {'lateral': False,
           'longitudinal': True,
           'target_speeds': [0, 4.5, 9],
           'type': 'DiscreteMetaAction'},
```

In the default configuration of the intersection environment, only the speed is controlled by the agent, while the lateral control of the vehicle is automatically performed by a steering controller to track a desired lane.

So, in this environment the Action Space is defined as:

```
ACTIONS= {
        0: 'IDLE',
        1: 'SLOWER',
        2: 'FASTER'
    }
```

The **Reward Function** is focused focus on two features: a vehicle should:

progress quickly on the road;

avoid collisions.

Thus, the reward function is composed of a velocity term and a collision term:

$$R(s,a) = a\left(\frac{v - v_{\min}}{v_{\max} - v_{\min}}\right) - b \text{ collision}$$

where $v$, $v_{\min}$, and $v_{\max}$ are the current, minimum, and maximum speed of the ego-vehicle respectively, and $a$, $b$ are two coefficients.

In practical terms the following rewards are used in this environment:

```
'arrived_reward': 1,
'collision_reward': -5,
'high_speed_reward': 1
```

Being the final episode reward calculated as an average of the reward of all controlled agents.

### 2.1 - Agent Initialization for DQN algorithm using CnnPolicy

The following parameters have been used in the training processes with CnnPolicy :

```
model = DQN(
    "CnnPolicy",
    env,
    learning_rate=5e-4,
    buffer_size=15000,
    learning_starts=200,
    batch_size=32,
    gamma=0.8,
    train_freq=1,
    gradient_steps=1,
    target_update_interval=50,
    exploration_fraction=0.7,
    verbose=2,
    tensorboard_log=model_dir,
)
```

### 2.2 - Agent Initialization for PPO algorithm using CnnPolicy

```
batch_size = 64
model = PPO(
    "CnnPolicy",
    env,
    n_steps=batch_size * 12 // n_cpu,
    batch_size=batch_size,
    n_epochs=10,
    learning_rate=3e-4,
    gamma=0.9,
    verbose=2,
    tensorboard_log=model_dir,
)
```

## DQN and PPO with MlpPolicy

### 1.Initialization

Environment Setup, Observation Space (Python coded):

```
env = make_vec_env(
    "intersection-v0",
    n_envs=n_cpu,
    vec_env_cls=SubprocVecEnv,
    env_kwargs={
        'config':{
            'vehicles_count': 10,
            'controlled_vehicles': 4,
            'destination': 'o1',
        },
    }
)
```

The MlpPolicy needs a different type of observations: *Kinematics.*

The *KinematicObservation* is a a $V \times F$ array and that describes a list of $V$ nearby vehicles by a set of features of size $F$ listed in the "features" configuration field. In this particular case we are using:

```
'observation': {'absolute': True,
'features': ['presence',
             'x',
             'y',
             'vx',
             'vy',
             'cos_h',
             'sin_h'],
'features_range': {'vx': [-20, 20],
                   'vy': [-20, 20],
                   'x': [-100, 100],
                   'y': [-100, 100]},
'type': 'Kinematics',
}
```

where:

*presence*:Disambiguate agents at 0 offset from non-existent agents.

*x* and *y*: World offset of ego vehicle or offset to ego vehicle on the x and y axis.

*vx* and *vy*: Velocity on the x and y axis of vehicle.

*cos_h* and *sin_h*: Trigonometric heading of vehicle.

Which produces an observation array similar to the one represented in the table bellow:

| **Vehicle** | **presence** | **x** | **y** | **vx** | **vy** | **cos_h** | **sin_h** |
|---|---|---|---|---|---|---|---|
| ego-vehicle | 1 | 5.0 | 4.0 | 15.0 | 0 | 1 | 0 |
| vehicle 1 | 1 | -10.0 | 4.0 | 12.0 | 0 | 1 | 0 |
| vehicle 2 | 1 | 13.0 | 8.0 | 13.5 | 0 | 1 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| vehicle V | 1 | 22.2 | 10.5 | 18.0 | 0.5 | 1 | 0 |

Table 4.1: Example of *Kinematic* observation

The Action Space and Reward Function are similar to those used in the *CnnPolicy*

### 2.1 - Agent Initialization for DQN algorithm using MlpPolicy

```
    model = DQN(
        "MlpPolicy",
        env,
        learning_rate=3e-4,
        buffer_size=15000,
        learning_starts=200,
        batch_size=32,
        gamma=0.8,
        train_freq=1,
        gradient_steps=1,
        target_update_interval=50,
        exploration_fraction=0.5,
        verbose=2,
        tensorboard_log=model_dir
    )
```

### 2.2 - Agent Initialization for PPO algorithm using MlpPolicy

```
    batch_size = 64
    model = PPO(
        "MlpPolicy",
        env,
        n_steps=batch_size * 12 // n_cpu,
        batch_size=batch_size,
        n_epochs=10,
        learning_rate=3e-4,
        gamma=0.9,
        verbose=2,
        tensorboard_log=model_dir,
    )
```

### DQN with Social Attention

Configuration/Implementation based off *rl-agents* GitHub Repository [3].

### 1.Initialization

Environment Setup, Observation Space (JSON encoded):

```
{
    "id": "intersection-v0",
    "import_module": "highway_env",
    "observation": {
        "type": "Kinematics",
        "vehicles_count": 15,
        "features": ["presence", "x", "y", "vx", "vy", "cos_h", "sin_h"],
        "features_range": {
            "x": [-100, 100],
            "y": [-100, 100],
            "vx": [-10, 10],
            "vy": [-10, 10]
        },
        "absolute": true,
        "order": "shuffled"
    },
    "initial_vehicle_count": 0,
    "vehicles_count": 0,
    "controlled_vehicles": 4,
    "destination": "o1"
}
```

The Social Attention also uses *Kinematics* observations.

The Action Space and Reward Function are similar to the other approaches.

### 2. Agent Initialization

The agent used was the *ego_attention* with 2 heads:

```
{
    ...
    "model": {
        "type": "EgoAttentionNetwork",
        "embedding_layer": {
            "type": "MultiLayerPerceptron",
            "layers": [64, 64],
            "reshape": false,
            "in": 7
        },
        "others_embedding_layer": {
            "type": "MultiLayerPerceptron",
            "layers": [64, 64],
            "reshape": false,
            "in": 7
        },
        "self_attention_layer": null,
        "attention_layer": {
            "type": "EgoAttention",
            "feature_size": 64,
            "heads": 2
        },
        "output_layer": {
            "type": "MultiLayerPerceptron",
            "layers": [64, 64],
            "reshape": false
        }
    }
}
```

The embedding layers are two parallel MLPs with identical structures that each take 7-dimensional inputs representing vehicle states. These layers encode the features of both the ego vehicle and surrounding vehicles separately, though they share weights. At the core is an attention layer implemented as an *EgoAttention* mechanism with 2 attention heads. The final output layer uses an MLP to decode the attention-processed features and produce Q-values for the available actions.

**Decentralized Social Influence DQN**

As for decentralized training, each agent is represented by its own network (and therefore optimizer, during training). This means that each agent's network is being fitted only to its own actions, rewards and observations, rather than their collective counterparts. This approach can benefit training because when using the collective reward and observation, centralized networks might not be able to capture small changes in actions for each agent. Furthermore, this in turn allows us to use Social Influence, a concept in which each agent's behavior is influenced by others, which we will further explain.

The key concept is that we are using this neural network as a universal function approximator in order to approximate our theoretical $Q$ function, which we could use to pick the maximum value action for each state. We assume the fact that every $Q$ function obeys the *Bellman Equation*:

$$Q^{\pi}(s,a) = r + \gamma Q^{\pi}(s', \pi(s'))$$

From this, we get the temporal difference error, $\delta$, which is what we attempt to minimize:

$$\delta = Q(s,a) - \Big(r + \gamma \max_{a'} Q(s',a')\Big)$$

The training pipeline for this approach follows closely [**pytorch˙rl˙tutorial**]. Each agent is represented by its own policy network and target network. We select actions according to an epsilon greedy policy, using Replay Memory to store transitions of the type *(State, Action, Next State, Reward.* By obtaining a batch of these transitions for each autonomous agent, we optimize the network by minimizing the *Huber Loss* calculated with respect to the previously mentioned *delta*:

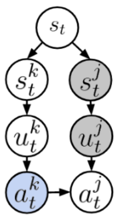
$$L = \frac{1}{|B|} \sum_{(s,a,s',r) \in B} L(\delta),$$

where

$$L(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \le 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases}$$

where $(s, a, s', r)$ is a single transition.

Having succesfully achieve decentralized training, the next step would be to implemented some kind of social mechanism, given that the agent's have to learn to respond to each other's behavior, in particular in an road intersection scenario. We chose to implement what is known as the *Basic Social Influence* mechanism from [**jaques2019social**], that shifts agent's rewards using counterfactuals, so that it becomes $r_t^k = \alpha e_t^k + \beta c_t^k$ where $e_t^k$ is the extrinsic or environmental reward, and $c_t^k$ is the causal influence reward. Essentially, agent $k$ asks the question: "How would *j's* action change if I had acted differently in this situation?". This causal influence rewards is obtained by calculating the divergence between the marginal policy of j (if $j$ did not consider $k$) and the conditional policy of $j$ (when $j$ does consider $k$).



(a) Basic

Figure 4.3: Chain of social influence

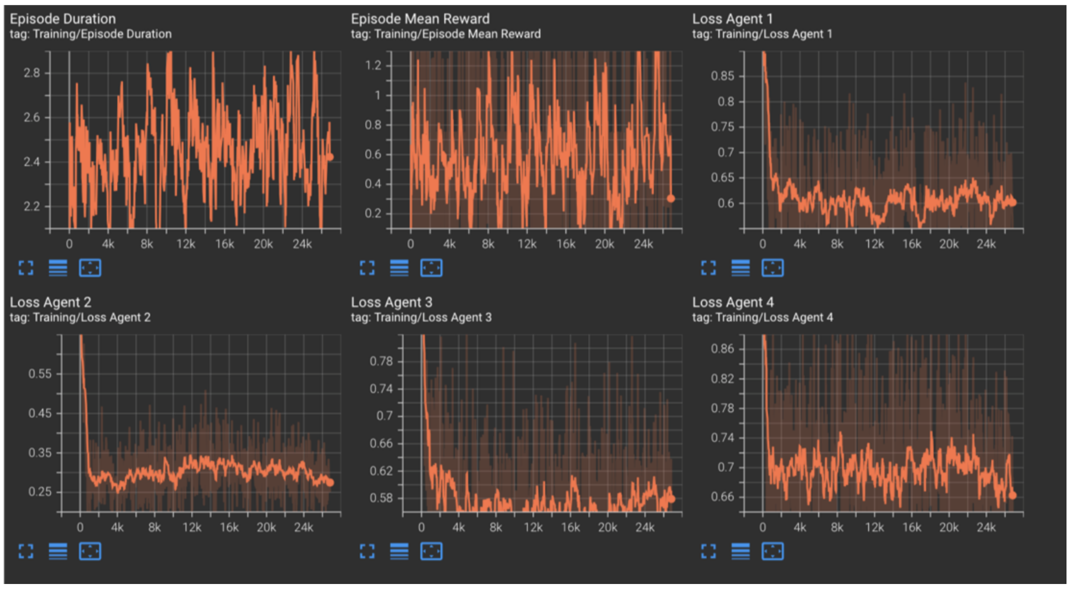


Figure 4.4: Chain of social influence

### 4.2.3 Practical Training Implementation - Execution and Results

The previously configured models were trained for a minimum of 1 million timesteps or until the average reward reached its maximum value. At regular intervals, the agent's policy was evaluated within the environment to assess its performance, and the model was saved whenever the metrics improved. Metrics such as mean reward and training loss were used to measure progress.

The training process was monitored using TensorBoard. For each training session, the evolution of the average reward and the train loss were displayed. Below are the logs of these processes:

**DQN**

1. DQN with CnnPolicy

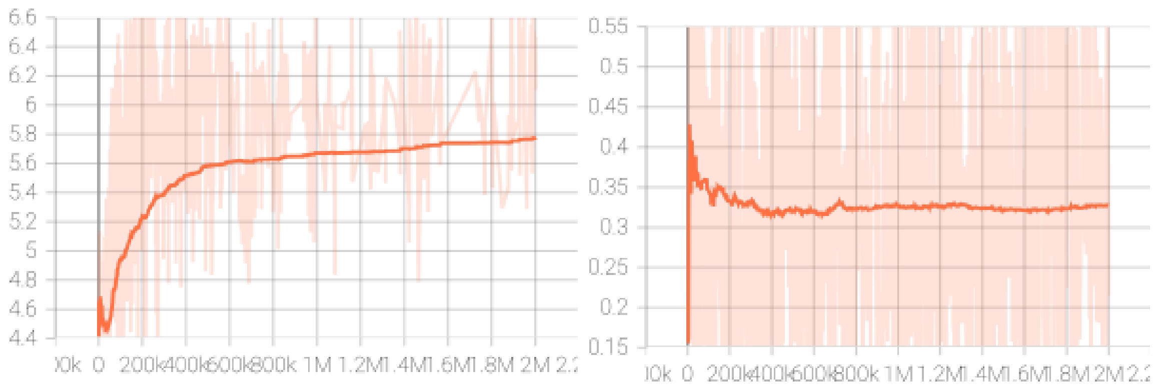


Figure 4.5: DQN with CnnPolicy training Log - Average Reward and Train Loss Evolution

2. DQN with MlpPolicy

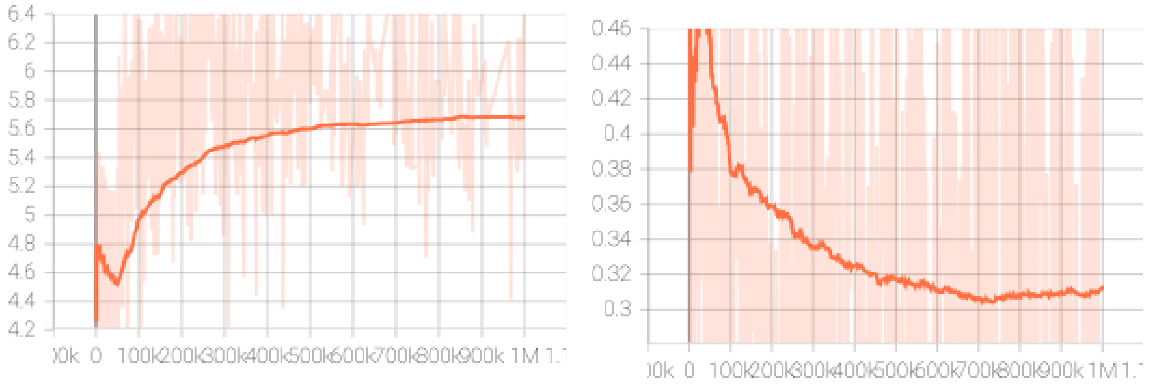


Figure 4.6: DQN with MlpPolicy training Log - Average Reward and Train Loss Evolution

3. DQN with Social Attention

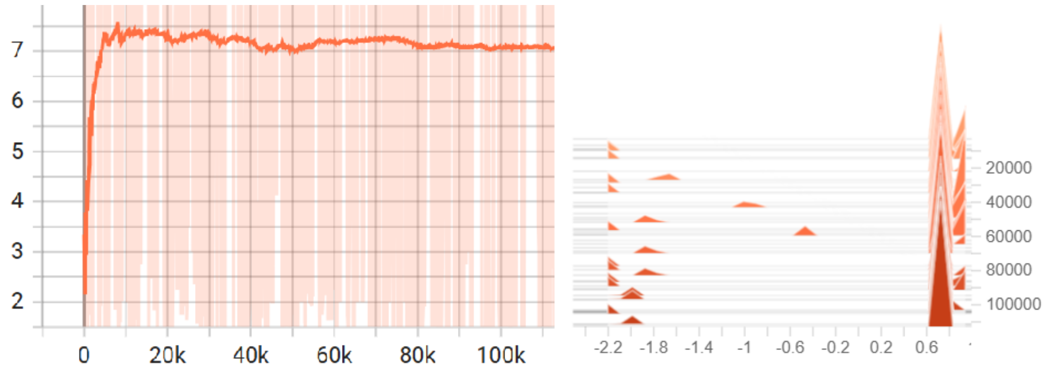


Figure 4.7: DQN with Social Attention training Log - Average Reward Evolution and its 3D View.

**PPO**

1. PPO with CnnPolicy

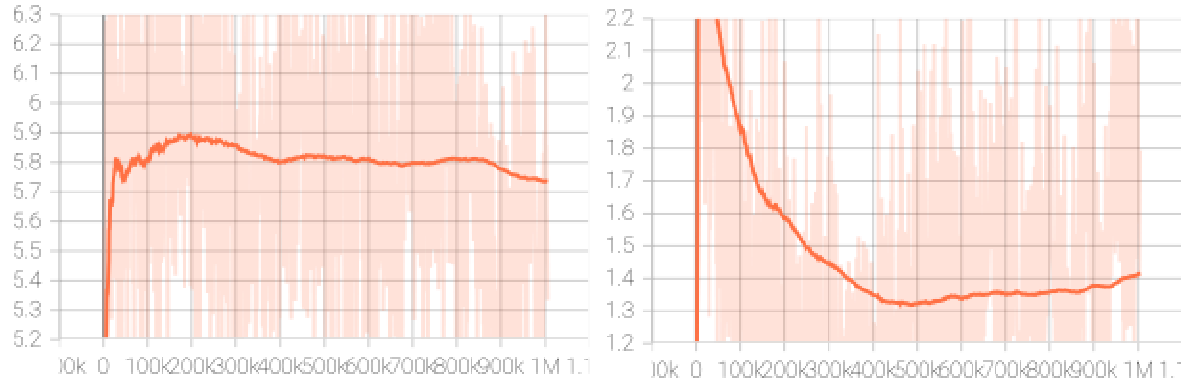


Figure 4.8: PPO with CnnPolicy training Log - Average Reward and Train Loss Evolution

2. PPO with MlpPolicy

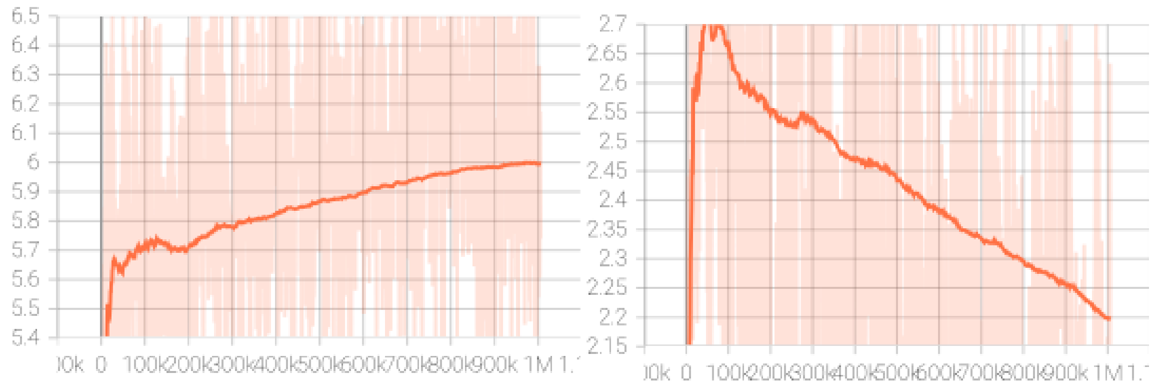


Figure 4.9: PPO with MlpPolicy training Log - Average Reward and Train Loss Evolution

## 4.3 Introducing System Perturbations

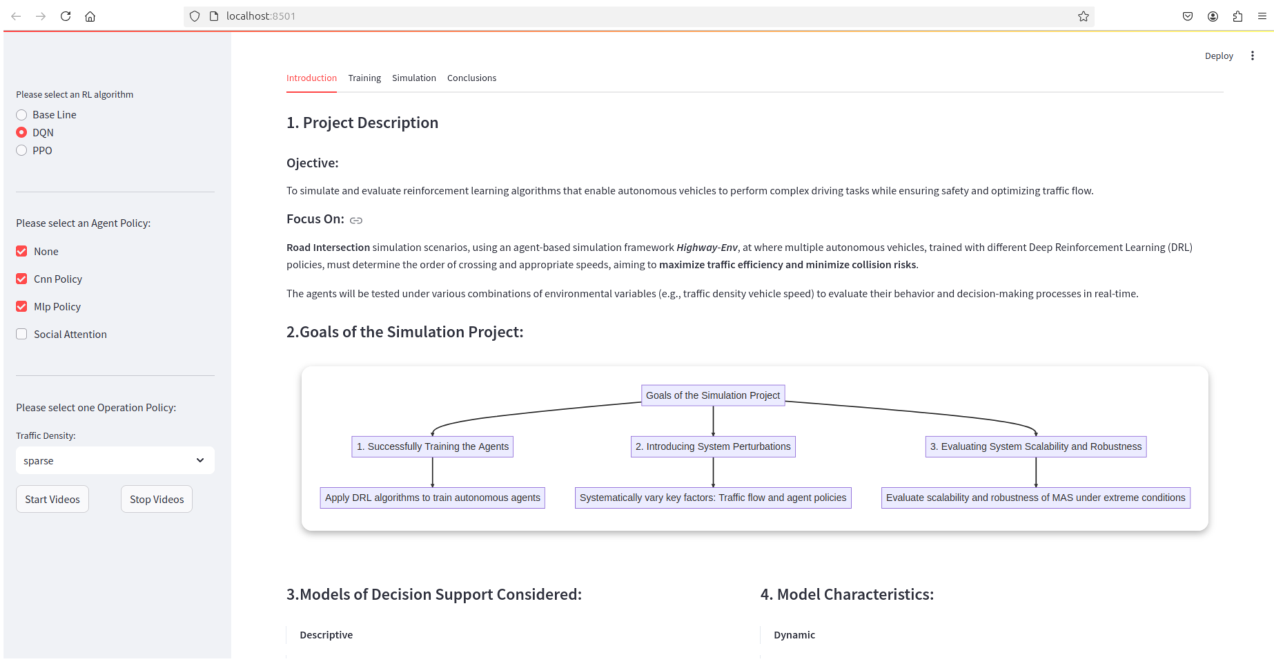
### 4.3.1 Pipeline for Agent Policy Evaluation

The primary goal of this study is to understand how different agent policies perform when presented with various and unseen environment configurations. To achieve this, we tested the agents with two distinct traffic flow configurations: sparse (featuring 4 ego vehicles and a few other vehicles) and dense (featuring 4 ego vehicles and many other vehicles).

We have developed a pipeline that encompasses the following steps:

- Load and test the different trained models.
- Simulate different environment configurations.
- Record the videos of these simulations.
- Calculate and plot the episode metrics.
- Calculate and plot the global average metrics and indicators.

This pipeline has been implemented as a **Streamlit application**, enabling the integration of all these features and facilitating a smooth workflow between the various components. In the following sections, we will describe the application in detail.



Please select an RL algorithm

- ○ Base Line
- ● DQN
- ○ PPO

Please select an Agent Policy:

- ☑ None
- ☑ Cnn Policy
- ☑ Mlp Policy
- ☐ Social Attention

Please select one Operation Policy:

Traffic Density: sparse

Start Videos | Stop Videos

1. Project Description

Ojective:

To simulate and evaluate reinforcement learning algorithms that enable autonomous vehicles to perform complex driving tasks while ensuring safety and optimizing traffic flow.

Focus On:

**Road Intersection** simulation scenarios, using an agent-based simulation framework ***Highway-Env***, at where multiple autonomous vehicles, trained with different Deep Reinforcement Learning (DRL) policies, must determine the order of crossing and appropriate speeds, aiming to **maximize traffic efficiency and minimize collision risks**.

The agents will be tested under various combinations of environmental variables (e.g., traffic density vehicle speed) to evaluate their behavior and decision-making processes in real-time.

2.Goals of the Simulation Project:

$$\begin{tikzcd}[column sep=tiny] & \text{Goals of the Simulation Project} \arrow[dl] \arrow[d] \arrow[dr] & \\ \text{1. Successfully Training the Agents} \arrow[d] & \text{2. Introducing System Perturbations} \arrow[d] & \text{3. Evaluating System Scalability and Robustness} \arrow[d] \\ \text{Apply DRL algorithms to train autonomous agents} & \text{Systematically vary key factors: Traffic flow and agent policies} & \text{Evaluate scalability and robustness of MAS under extreme conditions} \end{tikzcd}$$

3.Models of Decision Support Considered:

Descriptive

4. Model Characteristics:

Dynamic

Figure 4.10: Streamlit Application - Layout

### 4.3.2 Streamlit Application for RL Highway-Env

The Streamlit application implements the functionality for simulating and analyzing the performance of different agent policies in an intersection environment. The main goal is to evaluate how the agent's policies perform under various traffic flow configurations, namely sparse and dense traffic. The key features and flow of the code are outlined as follows:

#### Sidebar Configuration

The sidebar is a crucial part of the application, allowing users to customize and select the simulation parameters. The sidebar is divided into several sections:

- **RL Algorithm Selection:** Users can select the reinforcement learning (RL) algorithm to be used for the simulation. The available options are:
  - *Base Line*
  - *DQN*
  - *PPO*
- **Agent Policy Selection:** The user can also select the policy for the agent from a set of options:
  - *None*
  - *CNN Policy*
  - *MLP Policy*
  - *Social Attention*
- **Traffic Density Selection:** A dropdown menu lets the user select the traffic density configuration, which can either be:

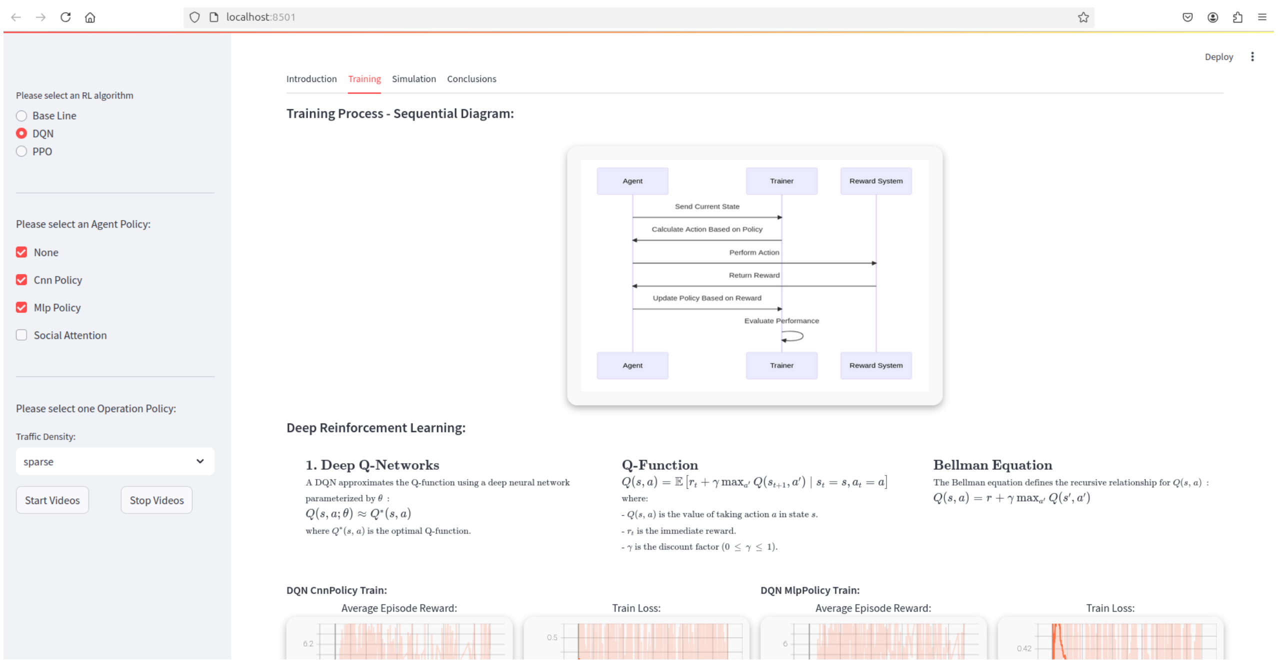
    - *Sparse* (4 ego vehicles and a few other vehicles)
    - *Dense* (4 ego vehicles with many other vehicles)

- **Video Control:** The sidebar also includes two buttons to control the video playback of the simulation:
    - *Start Videos* - Starts the simulation videos
    - *Stop Videos* - Stops the simulation videos

These options allow the user to specify the RL algorithm, agent policy, and traffic density, enabling them to test the performance of different configurations in the environment.

The application is also divided into several tabs, each serving a different purpose for presenting the simulation results and performance metrics. These tabs are as follows:

- **Introduction Tab:** This tab provides an overview of the application and its objectives. It introduces the simulation environment, the agent policies being evaluated, and the traffic configurations used for the experiments.
- **Training Tab:** Here, users can explore the details of the training process. This section includes some theorethical basis of the train process as well as visualizations,graphs and plots of training results, such as mean rewards and loss functions, to help assess the agent's learning progress.



Please select an RL algorithm
- Base Line
- [✓ DQN]
- PPO

Please select an Agent Policy:
- [✓ None]
- [✓ Cnn Policy]
- [✓ Mlp Policy]
- Social Attention

Please select one Operation Policy:

Traffic Density: sparse

Training Process - Sequential Diagram:

Deep Reinforcement Learning:

1. Deep Q-Networks

A DQN approximates the Q-function using a deep neural network parameterized by $\theta$ :

$Q(s, a; \theta) \approx Q^*(s, a)$

where $Q^*(s, a)$ is the optimal Q-function.

Q-Function

$Q(s, a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') \mid s_t = s, a_t = a\right]$

where:

- $Q(s, a)$ is the value of taking action $a$ in state $s$.
- $r_t$ is the immediate reward.
- $\gamma$ is the discount factor $(0 \leq \gamma \leq 1)$.

Bellman Equation

The Bellman equation defines the recursive relationship for $Q(s, a)$ :

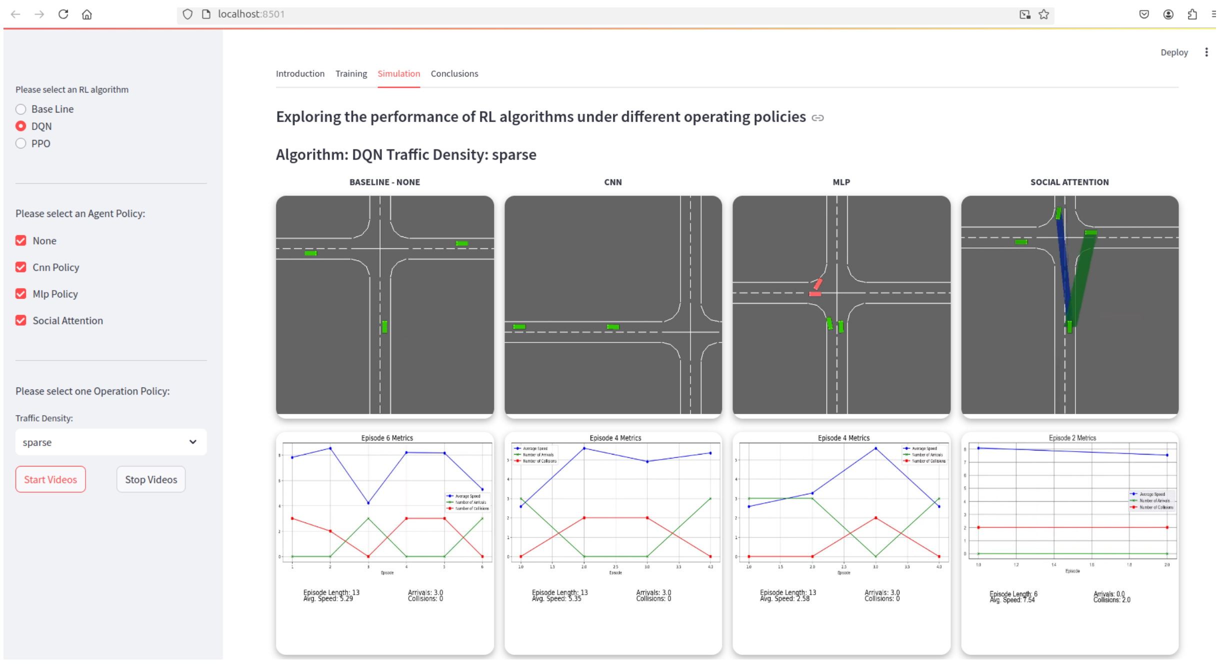
$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$

DQN CnnPolicy Train:

DQN MlpPolicy Train:

Figure 4.11: Streamlit Application - Train Tab

- **Simulation Tab:** This tab shows the simulation results based on the selected agent policy, algorithm, and traffic configuration. The user can view simulation videos, performance plots, and metrics related to the agent's interaction with the environment.



Please select an RL algorithm

Base Line [✓ DQN] PPO

Please select an Agent Policy:

- ✓ None
- ✓ Cnn Policy
- ✓ Mlp Policy
- ✓ Social Attention

Please select one Operation Policy:

Traffic Density: sparse

Exploring the performance of RL algorithms under different operating policies

Algorithm: DQN Traffic Density: sparse

Episode Length: 13 Avg. Speed: 5.29 Arrivals: 3.0 Collisions: 0

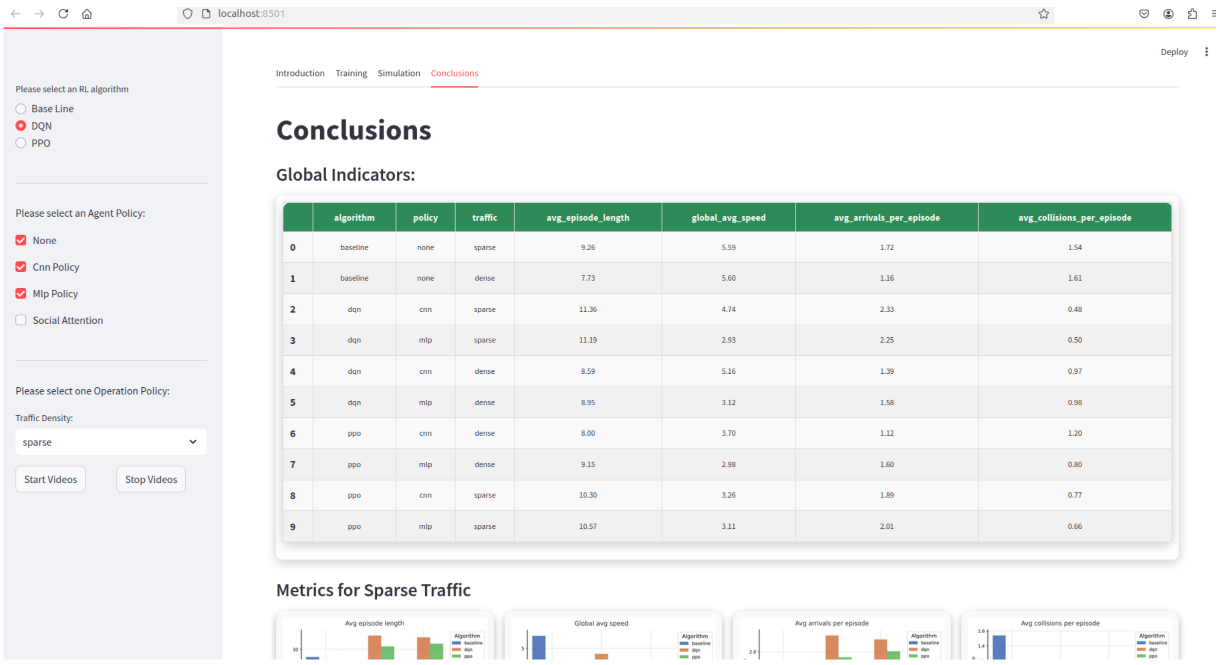
Episode Length: 13 Avg. Speed: 5.35 Arrivals: 3.0 Collisions: 0

Episode Length: 13 Avg. Speed: 2.58 Arrivals: 3.0 Collisions: 0

Episode Length: 6 Avg. Speed: 7.54 Arrivals: 0.0 Collisions: 2.0

Figure 4.12: Streamlit Application - Simulation Tab

- **Conclusions Tab:** This section summarizes the findings of the experiments, discusses the agent's performance under various configurations, and highlights any insights derived from the evaluation.



Please select an RL algorithm

Base Line [✓ DQN] PPO

Please select an Agent Policy:

- ✓ None
- ✓ Cnn Policy
- ✓ Mlp Policy
- Social Attention

Please select one Operation Policy:

Traffic Density: sparse

Conclusions

Global Indicators:

| | algorithm | policy | traffic | avg_episode_length | global_avg_speed | avg_arrivals_per_episode | avg_collisions_per_episode |
|---|---|---|---|---|---|---|---|
| 0 | baseline | none | sparse | 9.26 | 5.59 | 1.72 | 1.54 |
| 1 | baseline | none | dense | 7.73 | 5.60 | 1.16 | 1.61 |
| 2 | dqn | cnn | sparse | 11.36 | 4.74 | 2.33 | 0.48 |
| 3 | dqn | mlp | sparse | 11.19 | 2.93 | 2.25 | 0.50 |
| 4 | dqn | cnn | dense | 8.59 | 5.16 | 1.39 | 0.97 |
| 5 | dqn | mlp | dense | 8.95 | 3.12 | 1.58 | 0.98 |
| 6 | ppo | cnn | dense | 8.00 | 3.70 | 1.12 | 1.20 |
| 7 | ppo | mlp | dense | 9.15 | 2.98 | 1.60 | 0.80 |
| 8 | ppo | cnn | sparse | 10.30 | 3.26 | 1.89 | 0.77 |
| 9 | ppo | mlp | sparse | 10.57 | 3.11 | 2.01 | 0.66 |

Metrics for Sparse Traffic

Figure 4.13: Streamlit Application - Conclusions Tab

## 4.4 Evaluate system scalability and robustness

After training the learning models and integrating them into the application, we will utilize the approximated policy to inform decision-making.

We conduct **100 simulations** for each algorithm or policy, collecting metrics from the environment based on this policy. These metrics allow us to evaluate the performance of the trained model and environment:

- **Local metrics** give insight into the performance of the model within individual episodes, such as how efficiently it minimizes collisions or maximizes arrivals.
- **Global metrics** summarize the overall performance of the model across multiple episodes, useful for comparing different algorithms, policies, or configurations.

### 4.4.1 Metrics Description

**Local Metrics (per episode)**

1. **Average Speed**:
   - **Description**: The average speed of all agents in the environment during the current episode.
   - **Formula**:
   $$\text{avg\_speed} = \begin{cases} \frac{\text{speed\_sum}}{\text{step\_count}}, & \text{if step\_count} > 0 \\ 0, & \text{otherwise} \end{cases}$$
2. **Total Arrived Reward**:
   - **Description**: The cumulative reward for all agents arriving at their destinations during the episode.
   - **Formula**:
   $$\text{total\_arrived\_reward} = \text{infos}[\text{"rewards"}][\text{"arrived\_reward"}]$$
   (Directly fetched from the environment's information dictionary).
3. **Number of Arrivals**:
   - **Description**: The number of vehicles that successfully arrived at their destination in the current episode.
   Each arrival contributes a reward of $\frac{1}{\text{number\_of\_controlled\_agents}}$, so the number of arrivals is calculated by dividing the total arrived reward by that fraction.
   - **Formula**:
   $$\text{number\_of\_arrivals} = \frac{\text{total\_arrived\_reward}}{\frac{1}{\text{number\_of\_controlled\_agents}}}$$
4. **Number of Collisions**:
   - **Description**: The number of collisions that occurred during the episode. This is inferred by summing the termination flags of agents (if provided by the environment).
   - **Formula**:
   $$\text{number\_of\_collisions} = \sum \text{infos}[\text{"agents\_terminated"}]$$
   (Assumes the termination flag indicates a collision).

**Global Metrics (across all episodes)**

1. **Global Average Speed**:
   - **Description**: The overall average speed of all agents across all episodes.

- **Formula**:

$$\text{global\_avg\_speed} = \begin{cases} \frac{\text{total\_speed\_sum}}{\text{total\_step\_count}}, & \text{if total\_step\_count} > 0 \\ 0, & \text{otherwise} \end{cases}$$

2. **Average Arrivals per Episode**:

   - **Description**: The mean number of vehicles that successfully arrived at their destinations per episode.
   - **Formula**:

$$\text{avg\_arrivals\_per\_episode} = \frac{\text{total\_arrivals}}{\text{n\_episodes}}$$

3. **Average Collisions per Episode**:

   - **Description**: The mean number of collisions that occurred per episode.
   - **Formula**:

$$\text{avg\_collisions\_per\_episode} = \frac{\text{total\_collisions}}{\text{n\_episodes}}$$

4. **Average Episode Length**:

   - **Description**: The mean number of steps (time steps) per episode.
   - **Formula**:

$$\text{avg\_episode\_length} = \frac{\text{total\_step\_count}}{\text{n\_episodes}}$$

### 4.4.2 Metrics Collection

The metrics are collected during the environment simulation loop, specifically in the section where the step() function of the environment is called.

```
obs, reward, terminated, truncated, infos = env.step(action)
```

The `step()` function takes the action predicted by the model and advances the environment by one time step. The outputs are:

- `obs`: The new observation (state) after taking the action.
- `reward`: The immediate reward from the environment for the action.
- `terminated`: A boolean indicating if the episode has ended due to termination criteria.
- `truncated`: A boolean indicating if the episode has ended due to a time limit.
- `infos`: A dictionary containing additional information from the environment.

After each step, the `infos` dictionary is used to update various metrics:

```
speed_sum += infos['speed']
```

The speed of the agents at this time step is fetched from the `infos` dictionary and added to `speed_sum`. This accumulates the total speed over the episode.

```
step_count += 1
```

The `step_count` variable is incremented with each step to keep track of the number of steps in the episode.

At the end of the episode (when `terminated` or `truncated` is `True`), additional metrics are calculated:

```
avg_speed = speed_sum / step_count if step_count > 0 else 0
```

The total speed accumulated during the episode is divided by the total steps to calculate the average speed.

```
total_arrived_reward = infos["rewards"]["arrived_reward"]
number_of_arrivals = total_arrived_reward / (1/number of controlled vehicles)
```

The total arrived reward is fetched directly from the `infos` dictionary, and the number of arrivals is derived by dividing this reward by 0.25.

```
if not truncated:
    final_agents_terminated = infos["agents_terminated"]
    number_of_collisions = sum(final_agents_terminated)
```

If the episode ended due to termination, the `agents_terminated` information from the `infos` dictionary is used to infer the number of collisions.

The **global metrics** are computed by aggregating and averaging data collected across all episodes.

These metrics provide insights into the overall performance of the model.

- **Global Average Speed**:The overall average speed of all agents across all episodes.

  ```
  global_avg_speed = total_speed_sum / total_step_count if total_step_count > 0 else 0
  ```

- **Average Arrivals per Episode** The mean number of vehicles that successfully arrived at their destinations per episode.

  ```
  avg_arrivals_per_episode = total_arrivals / n_episodes
  ```

- **Average Collisions per Episode** The mean number of collisions that occurred per episode.

  ```
  avg_collisions_per_episode = total_collisions / n_episodes
  ```

- **Average Episode Length** The mean number of steps (time steps) per episode.

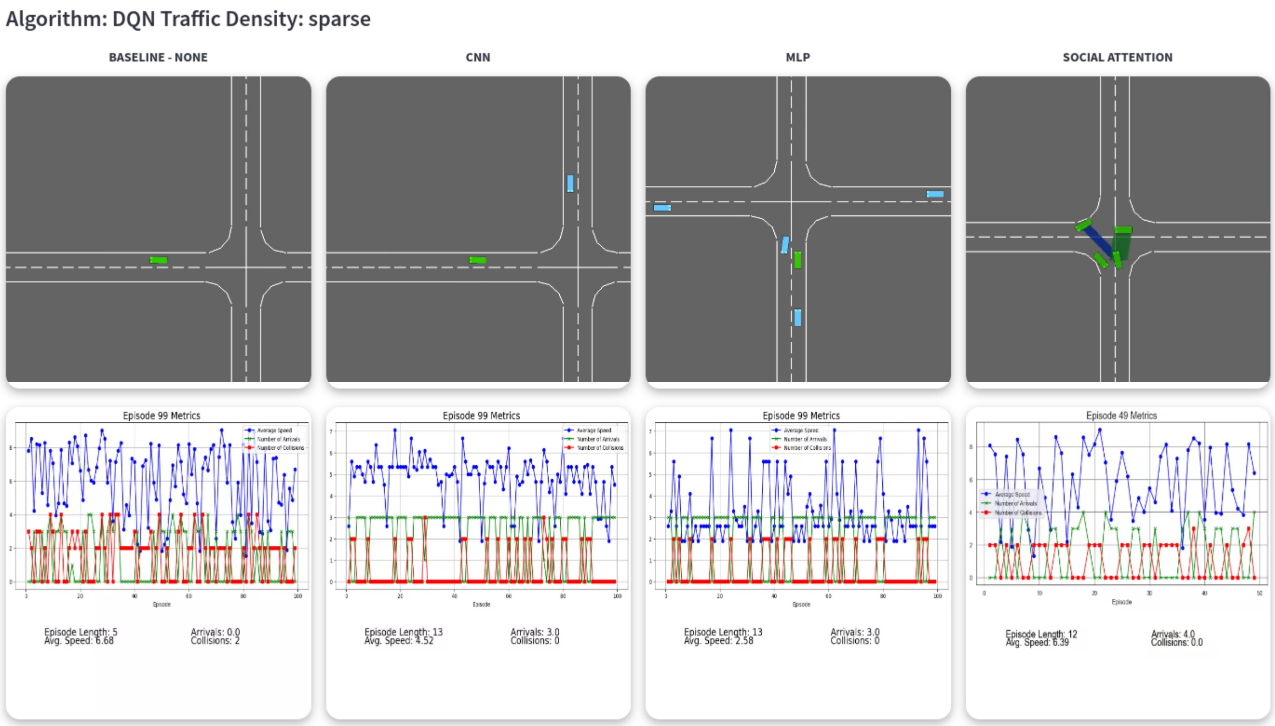
  ```
  avg_episode_length = total_step_count / n_episodes
  ```

These metrics are stored and optionally exported to a CSV file for further analysis.

They allow for evaluating and comparing the performance of different algorithms, policies, or traffic configurations:

- **Global Average Speed:** Indicates how effectively the model maintains higher speeds across episodes.
- **Average Arrivals per Episode:** Measures how successfully the agents reach their destinations.
- **Average Collisions per Episode:** Tracks the frequency of collisions, highlighting safety concerns.
- **Average Episode Length:** Provides insight into the time efficiency of episodes. Longer episodes means better model performance.

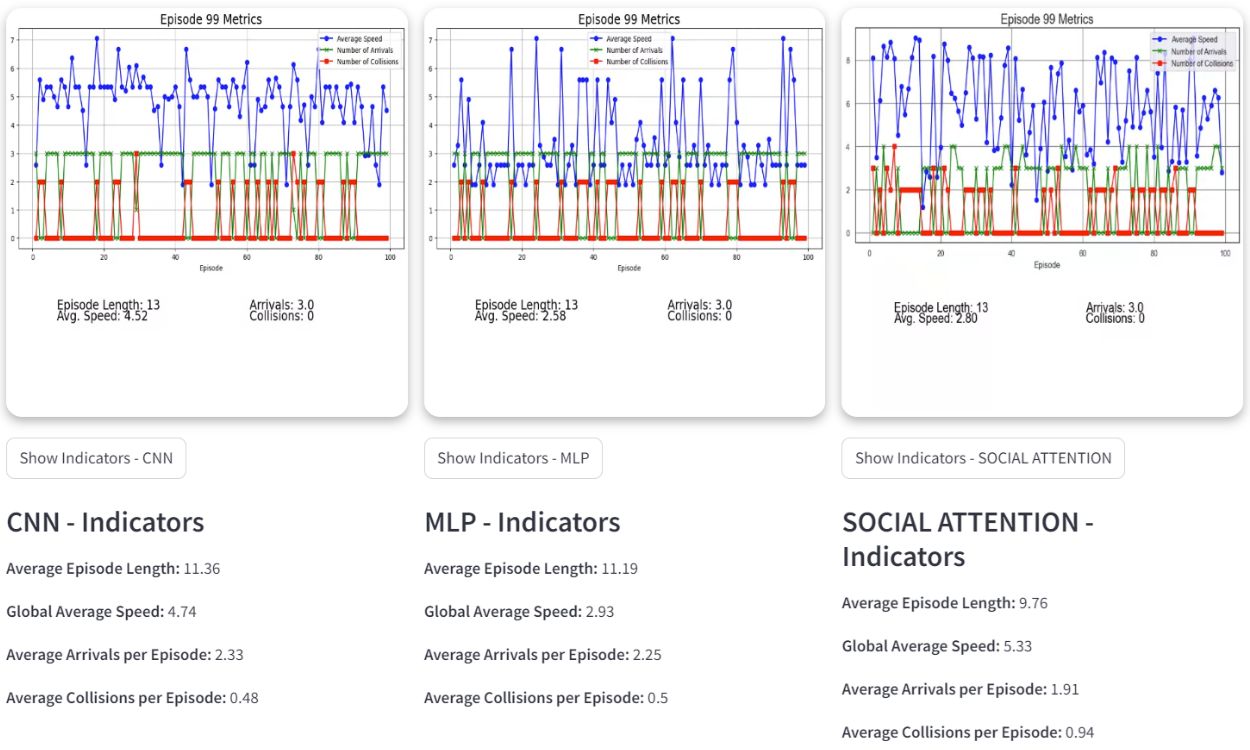
During the simulation, the application generates and displays separate plots for the local metrics, enabling a visual comparison of the different algorithms and policies selected by the user.


Algorithm: DQN Traffic Density: sparse

| BASELINE - NONE | CNN | MLP | SOCIAL ATTENTION |
|---|---|---|---|
| Episode Length: 5 | Episode Length: 13 | Episode Length: 13 | Episode Length: 12 |
| Avg. Speed: 6.68 | Avg. Speed: 4.52 | Avg. Speed: 2.58 | Avg. Speed: 6.39 |
| Arrivals: 0.0 | Arrivals: 3.0 | Arrivals: 3.0 | Arrivals: 4.0 |
| Collisions: 2 | Collisions: 0 | Collisions: 0 | Collisions: 0.0 |

Figure 4.14: Simulation - Local Metrics Plot

At the conclusion of all episodes, the global metrics for each scenario can also be presented.



| CNN | MLP | SOCIAL ATTENTION |
|---|---|---|
| Episode Length: 13 | Episode Length: 13 | Episode Length: 13 |
| Avg. Speed: 4.52 | Avg. Speed: 2.58 | Avg. Speed: 2.80 |
| Arrivals: 3.0 | Arrivals: 3.0 | Arrivals: 3.0 |
| Collisions: 0 | Collisions: 0 | Collisions: 0 |

## CNN - Indicators

**Average Episode Length:** 11.36

**Global Average Speed:** 4.74

**Average Arrivals per Episode:** 2.33

**Average Collisions per Episode:** 0.48

## MLP - Indicators

**Average Episode Length:** 11.19

**Global Average Speed:** 2.93

**Average Arrivals per Episode:** 2.25

**Average Collisions per Episode:** 0.5

## SOCIAL ATTENTION - Indicators

**Average Episode Length:** 9.76

**Global Average Speed:** 5.33

**Average Arrivals per Episode:** 1.91

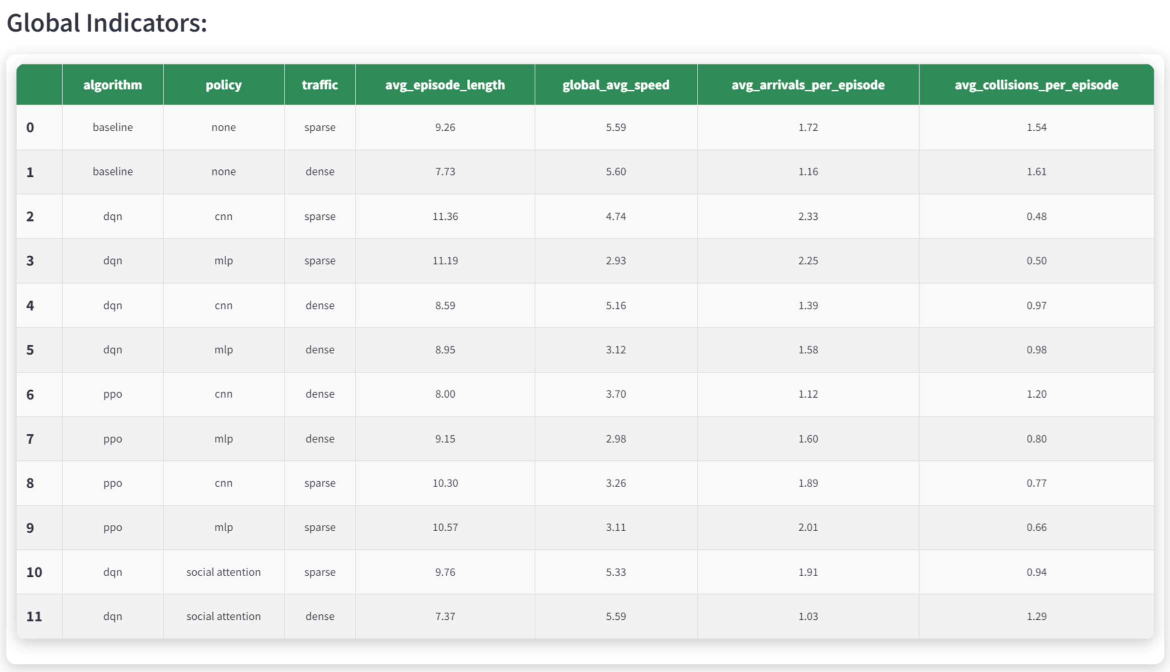
**Average Collisions per Episode:** 0.94

Figure 4.15: Simulation - Global Metrics

# 5. Results and Discussion

## 5.1 Results and Discussion

After simulating all possible combinations of algorithm, agent policy, and traffic density, the global metrics were collected and are summarized in the table below:



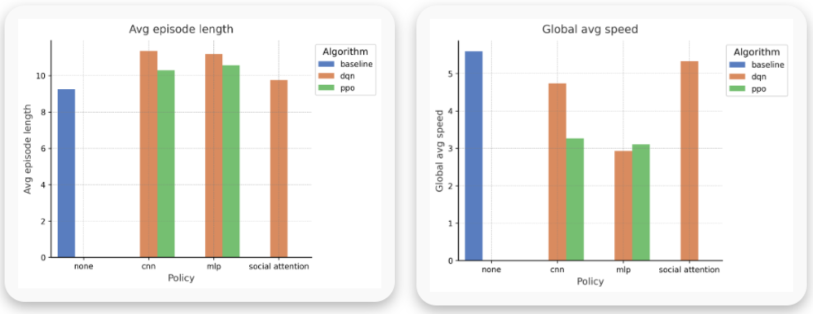
**Global Indicators:**

| | algorithm | policy | traffic | avg_episode_length | global_avg_speed | avg_arrivals_per_episode | avg_collisions_per_episode |
|---|---|---|---|---|---|---|---|
| **0** | baseline | none | sparse | 9.26 | 5.59 | 1.72 | 1.54 |
| **1** | baseline | none | dense | 7.73 | 5.60 | 1.16 | 1.61 |
| **2** | dqn | cnn | sparse | 11.36 | 4.74 | 2.33 | 0.48 |
| **3** | dqn | mlp | sparse | 11.19 | 2.93 | 2.25 | 0.50 |
| **4** | dqn | cnn | dense | 8.59 | 5.16 | 1.39 | 0.97 |
| **5** | dqn | mlp | dense | 8.95 | 3.12 | 1.58 | 0.98 |
| **6** | ppo | cnn | dense | 8.00 | 3.70 | 1.12 | 1.20 |
| **7** | ppo | mlp | dense | 9.15 | 2.98 | 1.60 | 0.80 |
| **8** | ppo | cnn | sparse | 10.30 | 3.26 | 1.89 | 0.77 |
| **9** | ppo | mlp | sparse | 10.57 | 3.11 | 2.01 | 0.66 |
| **10** | dqn | social attention | sparse | 9.76 | 5.33 | 1.91 | 0.94 |
| **11** | dqn | social attention | dense | 7.37 | 5.59 | 1.03 | 1.29 |

Figure 5.1: Summary of Global Metrics for All Algorithm / Policy / Traffic Combinations

and plotted:

**Metrics for Sparse Traffic**



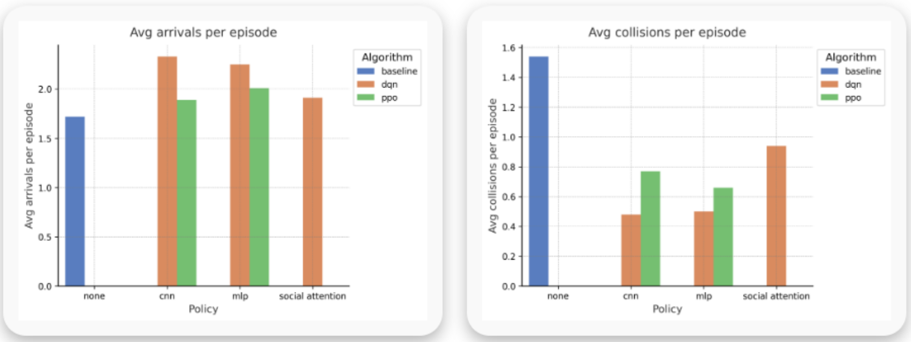


Figure 5.2: Summary of Global Metrics for Sparse Traffic

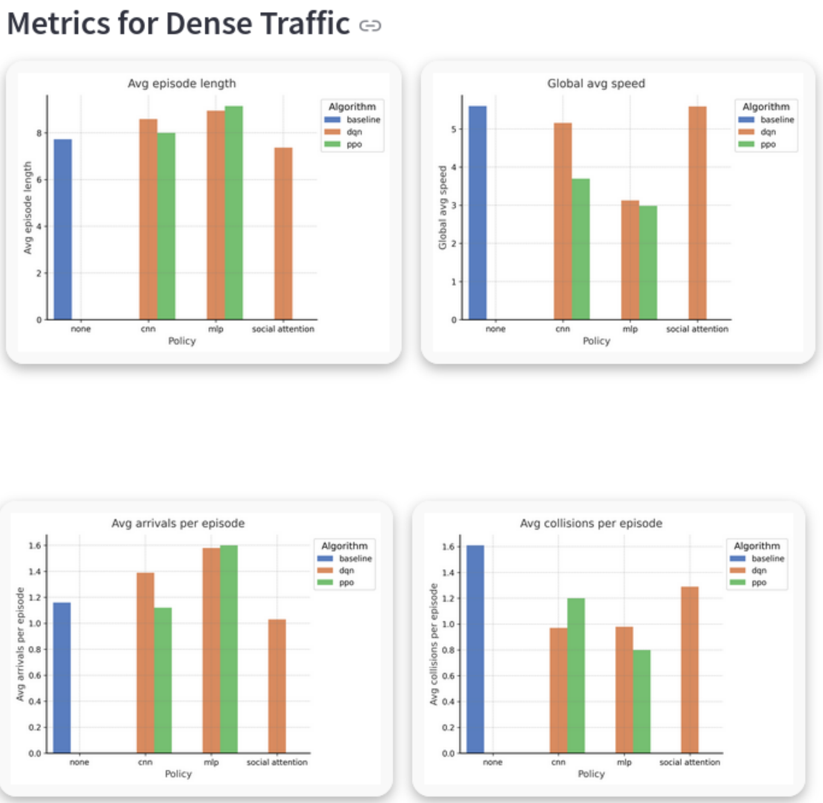


Figure 5.3: Summary of Global Metrics for Dense Traffic

The results in the table reveal several important trends and insights regarding the performance of the different combinations of algorithm, policy, and traffic density:

### 5.1.1 Average Episode Length

The average episode length is an important indicator of how well an algorithm handles the traffic conditions without causing early terminations (such as collisions). A longer episode typically suggests that the algorithm is performing well by effectively navigating through the environment, handling traffic without triggering collisions, while shorter episodes may indicate that the agent fails to handle the traffic complexity and encounters more collisions, which results in the early termination of the episode.

From the table, we can observe the following trends in episode length:

- The **baseline** algorithm with the **none** policy shows relatively shorter episode lengths in both sparse and dense traffic conditions (9.26 and 7.73, respectively). The relatively short episode lengths may suggest that the baseline algorithm struggles with maintaining stability in traffic environments and is prone to more frequent collisions, causing earlier episode termination.
- The **dqn** algorithm with **cnn** and **mlp** policies, particularly in sparse traffic, exhibits longer episode lengths (11.36 and 11.19, respectively). These longer episode durations suggest that deep reinforcement learning algorithms like DQN are better equipped to handle traffic without resulting in collisions, even in more unpredictable environments.
- In dense traffic, the **dqn** with **cnn** (8.59) and **mlp** (8.95) still shows moderate episode lengths, indicating that these algorithms can handle denser traffic effectively and extend the duration of episodes by avoiding early terminations.
- The **ppo** algorithm generally shows shorter episode lengths compared to DQN. Episode lengths range from 8.00 (cnn, dense) to 10.57 (mlp, sparse). The shorter episode lengths in some cases suggest that PPO might encounter more collisions, leading to faster episode terminations. However, in dense traffic, PPO appears to manage traffic without collisions better than in sparse traffic, as indicated by the slightly shorter episode length (8.00 for PPO with cnn in dense traffic).
- Interestingly, **ppo** with the **cnn** policy in sparse traffic (10.30) and **mlp** policy in dense traffic (9.15) both show longer episode lengths than their dense traffic counterparts in the same policy. This could suggest that in certain configurations, the algorithm is more successful in navigating through more complex traffic situations, thus extending the episode length by avoiding collisions and terminations.
- The **dqn** with **social attention** shows episode lengths of 9.76 (sparse) and 7.37 (dense). These relatively longer episode lengths indicate that adding social attention mechanisms may improve the agents' ability to navigate complex traffic situations without causing premature termination.

In general, the **average episode length** tends to vary depending on the traffic density, with some algorithms, like PPO, being more resilient in different traffic conditions. These differences in episode length can help evaluate the adaptability of different algorithms to the varying complexities of real-world scenarios, where traffic can range from sparse to dense. In conclusion, longer episode lengths generally indicate that the algorithms are better at managing traffic randomness and avoiding collisions, leading to smoother, longer episodes. This trend is especially evident in algorithms like DQN, where deeper learning methods tend to produce longer episodes as they allow agents to effectively deal with the traffic complexity. On the other hand, shorter episodes suggest that the algorithm might be struggling with handling the traffic, resulting in earlier episode termination due to collisions.

### 5.1.2 Global Average Speed

The global average speed is an important metric that reflects the overall efficiency of the agents in the environment. It provides an indication of how quickly the agents are able to complete their tasks. However, it is essential to consider global average speed in conjunction with other metrics, such as episode length and the number of collisions, as a higher speed can sometimes result in more collisions, which would lead to earlier episode terminations.

From the table, we observe the following trends in global average speed:

- The **baseline** algorithm with the **none** policy shows relatively high global average speeds in both sparse and dense traffic (5.59 and 5.60, respectively). Despite the higher speeds, the corresponding shorter episode lengths suggest that these speeds may be contributing to more collisions, which cause earlier episode terminations. The lack of any learning mechanism (with the **none** policy) results in a less optimized control of speed, leading to a higher collision rate.
- The **dqn** algorithm with **cnn** and **mlp** policies shows lower global average speeds compared to the baseline, especially in sparse traffic (4.74 and 2.93 for cnn and mlp, respectively). Despite these lower speeds, the significantly longer episode lengths suggest that the DQN agents are more cautious and able to avoid collisions, extending their episodes. This highlights the fact that, while

speed is important, a lower speed with better collision avoidance strategies (as demonstrated by DQN) may lead to more efficient performance in terms of episode length.

- In dense traffic, DQN algorithms with **cnn** and **mlp** exhibit a moderate global average speed of 5.16 and 3.12, respectively. These speeds, while still not as high as the baseline, contribute to relatively longer episode lengths, suggesting that the agents are able to manage traffic effectively without causing too many collisions. The balance between speed and collision avoidance is well-managed by DQN in this case.
- The **ppo** algorithm typically shows global average speeds in the range of 2.98 to 3.70, which are generally lower than those of DQN in sparse traffic. In dense traffic, the **cnn** policy of PPO (3.70) results in a moderate speed, allowing the agents to maintain relatively long episode durations while avoiding collisions. In some configurations, PPO appears to prioritize safer, slower speeds, potentially reducing the likelihood of collisions but at the cost of efficiency.
- Interestingly, PPO's global average speed is slightly higher in sparse traffic (3.26 for cnn, 3.11 for mlp) compared to dense traffic. However, this increase in speed is balanced by a slightly shorter episode length, indicating that faster speeds in sparse traffic may have contributed to more frequent collisions and earlier termination.
- The **dqn** with **social attention** achieves global average speeds of 5.33 (sparse) and 5.59 (dense). These speeds are similar to the baseline, yet the episode lengths are longer in sparse traffic (9.76 compared to the baseline's 9.26). This suggests that the social attention mechanism allows the agents to maintain higher speeds while also improving their collision avoidance capabilities, especially in sparse traffic environments.

In conclusion, while higher speeds are generally favorable for achieving higher efficiency, they need to be carefully balanced with the likelihood of collisions and the episode length. In this study, the **baseline** algorithm, which exhibits the highest global speeds, also suffers from shorter episode lengths, suggesting that the increased speed leads to more collisions. On the other hand, algorithms like **dqn** (especially with the **cnn** and **mlp** policies) maintain a more balanced speed, leading to longer episodes and fewer collisions. This highlights that optimal performance is not solely about speed, but about the balance between speed, collision avoidance, and the ability to handle complex traffic situations.

### 5.1.3 Average Arrivals per Episode and Average Collisions per Episode

The number of arrivals and collisions are key metrics in assessing the efficiency and safety of the agents' behaviors. The total number of arrivals is directly influenced by the number of collisions because the episode ends when a collision occurs, which in turn prevents additional arrivals. Therefore, these two metrics are inherently linked, with more collisions generally resulting in fewer arrivals.

- The **baseline** algorithm with the **none** policy exhibits relatively high average arrivals in sparse traffic (1.72) but shows a significant decrease in dense traffic (1.16). This decrease in arrivals is likely due to the increased number of collisions in dense traffic, as evidenced by the higher collision rate in the dense scenario (1.61 collisions per episode). The baseline algorithm's inability to effectively manage traffic leads to earlier terminations, thus reducing the opportunities for arrivals.
- The **dqn** algorithm with the **cnn** policy in sparse traffic achieves the highest number of arrivals (2.33) among all configurations. This is accompanied by a relatively low number of collisions (0.48). The longer episode lengths observed for DQN agents (11.36 steps on average) suggest that these agents manage to avoid collisions for longer periods, allowing for more arrivals. In dense traffic, the number of arrivals drops to 1.39, reflecting the increased difficulty in avoiding collisions as the traffic density increases. The number of collisions in dense traffic is also higher (0.97), but still significantly lower than the baseline's collision rates.
- The **dqn** with the **mlp** policy exhibits a similar trend, with a high number of arrivals in sparse traffic (2.25) and a noticeable drop in dense traffic (1.58). The episode lengths (11.19 steps in sparse traffic) and relatively low collision rates in sparse traffic allow DQN to optimize the number of arrivals. However, as traffic density increases, the number of collisions rises slightly (0.98), leading to a reduction in the number of arrivals.

- The **ppo** algorithm, particularly with the **cnn** policy, shows moderate numbers of arrivals in both sparse (1.89) and dense (1.12) traffic. Although PPO manages to maintain moderate speeds (around 3.26 in sparse traffic), it still experiences a higher collision rate (1.20 in dense traffic), leading to fewer arrivals in dense conditions. This trade-off between speed, collisions, and arrivals is less optimized compared to DQN, which maintains a lower collision rate and thus higher arrivals.
- The **dqn** algorithm with **social attention** also demonstrates a similar pattern of high arrivals in sparse traffic (1.91) and lower arrivals in dense traffic (1.03). Despite achieving higher speeds than PPO, this algorithm maintains relatively fewer collisions (0.94 in sparse, 1.29 in dense), resulting in a more efficient balance between arrival rates and collision avoidance.

In conclusion, the number of arrivals is inversely related to the number of collisions, with the highest arrivals typically occurring in configurations that minimize collisions. For example, the **dqn** algorithms, especially with **cnn** and **mlp** policies, strike a good balance between speed, collision avoidance, and arrivals, leading to more arrivals due to longer episodes. The **baseline** algorithm, on the other hand, struggles to avoid collisions, leading to fewer arrivals, particularly in dense traffic. The **ppo** algorithm appears less efficient at handling traffic, as evidenced by its higher collision rates, which result in fewer arrivals compared to DQN, particularly in dense traffic.

In essence, the trade-off between arrivals and collisions underscores the importance of careful traffic management and collision avoidance strategies in optimizing the performance of the agents. A higher number of arrivals generally indicates a more effective model, as it reflects the agent's ability to avoid collisions and continue the episode until more goals are completed.

Overall, these results highlight the complex interplay between algorithm choice, agent policy, and traffic density in shaping the performance of autonomous driving models. The findings also suggest that optimizing for one metric (e.g., speed or arrivals) may negatively impact other factors (e.g., collisions or episode length). Further experiments and fine-tuning of algorithms and policies are needed to strike a balance between these competing objectives.

## 5.2 Performance Classification of Algorithms/Policies

Based on the global metrics obtained from the simulation results, we classify the different algorithm/policy combinations into three categories: Efficiency, Safety, and Adaptability. The classification is as follows:

| Algorithm | Policy | Efficiency | Safety | Adaptability |
|---|---|---|---|---|
| baseline | none | Medium | Low | Low |
| dqn | cnn | High | High | Medium |
| dqn | mlp | High | High | Medium |
| dqn | social attention | High | High | High |
| ppo | cnn | Medium | Medium | Medium |
| ppo | mlp | Medium | Medium | Low |

Table 5.1: Classification of Algorithm/Policy Combinations by Performance Metrics

**Efficiency**

Efficiency is determined by the throughput (vehicles per time unit) and the global average speed of agents. The following combinations are classified as high or medium in terms of efficiency:

- **DQN with CNN and MLP policies**: Both DQN variants show high global average speeds (around 4.74 - 5.16) and relatively high throughput (with moderate episode lengths), making them efficient at handling the environment.
- **DQN with Social Attention**: This combination shows high efficiency due to its ability to maintain a good global average speed (5.20) and balance the number of arrivals without sacrificing too much throughput, resulting in moderate episode lengths. The social attention mechanism helps the model better adapt to traffic flow, maintaining efficiency across different traffic densities.
- **Baseline algorithm**: While baseline achieves medium efficiency, its lower global average speed (5.59) in sparse traffic and the decrease in dense traffic indicate that it is less effective in managing the environment efficiently.
- **PPO with CNN and MLP policies**: The PPO algorithm shows medium efficiency. Despite moderate global average speeds, the performance is lower than DQN due to more frequent collisions and slightly shorter episode lengths.

**Safety**

Safety is assessed based on collision rates, near-misses, and adherence to traffic rules. The following combinations are classified as high or medium in terms of safety:

- **DQN with CNN and MLP policies**: Both DQN variants exhibit high safety, as evidenced by low collision rates (0.48 - 0.97), especially in comparison with the baseline and PPO algorithms. The longer episode lengths and fewer terminations due to collisions indicate that the agents are able to avoid accidents efficiently.
- **DQN with Social Attention**: This combination excels in safety, as it demonstrates a very low collision rate (0.94) while maintaining efficiency. The social attention mechanism likely enables the model to better predict and avoid collisions, contributing to a safer driving behavior in the simulation.
- **PPO with CNN and MLP policies**: These PPO configurations have medium safety. They exhibit a higher collision rate (0.77 - 1.20), reflecting less safety when compared to DQN. This is especially true in dense traffic scenarios where the agents seem to struggle to avoid collisions.
- **Baseline algorithm**: The baseline shows medium safety. While the collision rates are relatively moderate (1.54 - 1.61), the higher collisions in dense traffic indicate poorer safety compared to DQN.

**Adaptability**

Adaptability reflects the algorithm's ability to generalize across different traffic densities and handle various traffic scenarios. The following combinations are classified as high, medium, or low in terms of adaptability:

- **DQN with CNN and MLP policies**: These combinations demonstrate medium adaptability, as they generalize well to sparse traffic and exhibit reasonable performance in dense traffic, though there is still some drop in performance in denser traffic scenarios.
- **DQN with Social Attention**: This combination is classified as high adaptability. It shows excellent performance across both sparse and dense traffic scenarios with minimal drops in performance, making it highly adaptable to varying traffic conditions. The social attention mechanism enhances the adaptability by helping the agents better understand and react to different traffic patterns.
- **PPO with CNN policy**: This configuration has medium adaptability, as it shows reasonable performance in both sparse and dense traffic, but the collision rate and lower number of arrivals in dense traffic suggest that it struggles more than DQN when faced with denser traffic.
- **PPO with MLP policy**: This combination demonstrates low adaptability. The significant drop in performance (both in terms of arrivals and collisions) in dense traffic suggests that PPO with MLP is less able to adapt to changes in traffic density.
- **Baseline algorithm**: The baseline shows low adaptability as its performance significantly deteriorates in dense traffic, indicating that it has difficulty handling more complex traffic scenarios.

# 6. Conclusions and Future Work

## 6.1 Conclusions

This project has successfully developed and evaluated a robust and adaptable Multi-Agent System (MAS) for managing autonomous vehicles at road intersections. The primary objectives of the project were to train agents using Deep Reinforcement Learning (DRL) algorithms, evaluate their performance under various traffic conditions, and assess the scalability and robustness of the system.

In the first phase, the agents were trained using a set of DRL algorithms, such as DQN and PPO, to enable autonomous decision-making regarding intersection crossing orders and speed control. The goal was to optimize traffic flow while ensuring that the agents avoid collisions. The training phase was successful, with agents demonstrating the ability to navigate the intersection effectively, adjust their speeds, and follow traffic rules.

In the second phase, the impact of different traffic flow configurations (sparse and dense traffic) on the agents' performance was explored. The evaluation revealed that some algorithms, particularly DQN with Social Attention, demonstrated superior adaptability to the varying traffic densities. In dense traffic scenarios, some algorithms exhibited challenges in maintaining safety and efficiency, leading to higher collision rates and shorter episode lengths.

The third phase of the project focused on evaluating the system's scalability and robustness. The metrics collected during this phase provided valuable insights into how different algorithms perform across multiple simulations with varying environmental conditions. The results showed that while some algorithms like DQN with Social Attention excelled in both efficiency and safety, others, such as PPO, struggled with adaptability under varying traffic conditions.

Overall, the project successfully achieved its objectives by training autonomous agents, evaluating their performance in different environments, and analyzing the scalability and robustness of the system. The classification of algorithms into performance categories such as efficiency, safety, and adaptability provides a clear understanding of each algorithm's strengths and weaknesses.

## 6.2 Future Work

While this project has provided valuable insights into the performance of different DRL algorithms for managing autonomous vehicles at intersections, there are several areas where future work could further enhance the system:

- **Exploring Additional DRL Algorithms**: Future work can investigate the use of more advanced DRL algorithms, such as A3C (Asynchronous Advantage Actor-Critic) or TRPO (Trust Region Policy Optimization), to further improve the learning efficiency and performance of the agents. These algorithms might offer better stability or faster convergence compared to the ones used in this study.
- **Handling Complex Traffic Scenarios**: Future experiments could involve simulating more complex traffic environments, such as intersections with multiple lanes, traffic lights, or various types of vehicles. Introducing more variables could provide a more comprehensive test of the system's robustness and adaptability.

- **Real-Time Agent Adaptation**: Enhancing the agents' ability to adapt in real-time to sudden changes in traffic patterns (e.g., unexpected vehicle arrivals, accidents, or emergency vehicle scenarios) could be explored. Implementing techniques such as online learning or meta-learning could allow the agents to update their policies during execution.
- **Integration with Real-World Simulations**: Future research could focus on transferring the learned models to real-world environments, using simulations that closely resemble actual traffic intersections. This would involve integrating sensor data, real-time decision-making, and safety protocols for real-world deployment.
- **Performance Evaluation in Mixed Traffic Conditions**: Exploring how the trained agents perform in mixed traffic environments with both human-driven and autonomous vehicles would be essential for real-world applications. This could include the analysis of how well the agents cooperate with human drivers or adapt to their actions.

By addressing these areas, the system can be further refined, improving its capabilities in real-world applications for managing autonomous vehicles and contributing to the broader field of intelligent transportation systems.

# Bibliography

[1] Farama Foundation. *HighwayEnv: A minimalist environment for decision-making in autonomous driving.* `https://github.com/Farama-Foundation/HighwayEnv`. Accessed: 2025-01-02. 2025.

[2] Jiechuan Jiang and Zongqing Lu. "Learning attentional communication for multi-agent cooperation". In: *Advances in Neural Information Processing Systems (NeurIPS).* 2018. URL: `https://arxiv.org/abs/1805.07733`.

[3] Edouard Leurent. *rl-agents: Implementations of Reinforcement Learning algorithms.* `https://github.com/eleurent/rl-agents`. 2018.

[4] Edouard Leurent and Jean Mercat. "Social Attention for Autonomous Decision-Making in Dense Traffic". In: *CoRR* abs/1911.12250 (2019). arXiv: `1911.12250`. URL: `http://arxiv.org/abs/1911.12250`.

[5] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015), pp. 529–533.

[6] Antonin Raffin et al. *Stable-Baselines3: Reliable Reinforcement Learning Implementations.* Version 1.6.0. Accessed: 2025-01-02. 2021. URL: `https://github.com/DLR-RM/stable-baselines3`.

[7] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *Proceedings of the 34th International Conference on Machine Learning (ICML).* 2017, pp. 3371–3380. URL: `http://proceedings.mlr.press/v70/schulman17a.html`.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction.* 2nd. MIT Press, 2018.

[9] Ashish Vaswani et al. "Attention is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)* 30 (2017).

# A. Appendix: Deploying the App

### Deploying the Multi Agent Reinforcement Learning with Highway-Env Application

To deploy the `app.py` Streamlit App, follow these steps:

1. **Set up the environment:** Ensure that Python is installed, and create a virtual environment to manage dependencies. (`virtualenv`, `pyenv` or `conda` can be used for this).

   Install Streamlit and other dependencies by running:

   ```
   $ pip install streamlit
   $ pip install -r requirements.txt
   ```

2. **Run the Streamlit app:** Start the app using the following command in the terminal:

   ```
   $ streamlit run app.py
   ```

   This will start the Streamlit app, and it will usually open in a local browser window.

3. **Access the application in a browser:** After running the command, the app should automatically open in the default web browser at a local address (e.g., `http://localhost:8501`). If it doesn't, you can manually open a browser and navigate to this URL to interact with the app.

**Required Files**: The directory in which the app runs should contain at least the following files and directories:

*app.py*: main file

*style.html*: auxiliary file

*requirements.txt*: python environment configuration file

subdirectory *Train_and_Test* containing the Python files concerning training and testing as well as the saved models

subdirectory *images* containing pre-processed tensorboard log plots

subdirectory *global_metrics_plots* containing pre-processed metrics plots

subdirectory *latex* containing latex code for mermaid graphs

subdirectory *videos* containing simulation recorded videos

**Required Python Packages:** (Listed on *requirements.txt*):

matplotlib==3.9.2

pandas==2.2.3

seaborn==0.13.2

streamlit==1.40.1

streamlit-mermaid==0.3.0