Integrating Vision-Language Models with Reinforcement Learning for Human-Aligned Decision-Making of Autonomous Vehicles

Erfan Doroudian

A Thesis

in

The Department

of

Mechanical, Industrial and Aerospace Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Mechanical Engineering) at

Concordia University

Montréal, Québec, Canada

November 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Erfan Doroudian**

Entitled: **Integrating Vision-Language Models with Reinforcement Learning for Human-Aligned Decision-Making of Autonomous Vehicles**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Mechanical Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Youmin Zhang

_____ External Examiner
Dr. Arash Mohammadi

_____ Examiner
Dr. Youmin Zhang

_____ Supervisor
Dr. Hamid Taghavifar

Approved by _____
Dr. Muthukumaran Packirisamy, Chair
Department of Mechanical, Industrial and Aerospace Engineering

_____ 2025

_____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Integrating Vision-Language Models with Reinforcement Learning for Human-Aligned Decision-Making of Autonomous Vehicles

Erfan Doroudian

This thesis focuses on developing a new approach for improving the decision-making process for autonomous vehicles (AVs) in complex urban driving scenarios, particularly at unsignalized intersections, using the reinforcement learning (RL) framework. One of the primary difficulties in RL environments is designing a suitable reward model, which can often be challenging to achieve manually due to the complexity of the interactions and the driving scenarios. To address this challenge, this work utilizes Vision-Language Models (VLMs), particularly CLIP (Contrastive Language-Image Pretraining), to build an additional reward model based on visual and textual cues. CLIP's ability to align image and text embeddings provides unique features for translating human-like instructions into reward signals to guide the AV's decision-making process. We apply two RL algorithms, Proximal Policy Optimization (PPO) and Deep Q-Network (DQN), to train an agent in complex unsignalized intersection environments. The performance of these algorithms is compared with and without the CLIP-based reward model, which highlights the impact of CLIP on the agent's ability to learn and optimize its behavior in a way that aligns with desired driving actions. This study's results show VLMs' capabilities in improving RL-based decision-making in autonomous driving. We utilize the *Highway-env* simulation package, built on the OpenAI Gym framework, to test and validate the effectiveness of the proposed framework. Our simulation experiments indicate the framework's effectiveness in optimizing traffic flow, minimizing collisions, and balancing both individual and collective benefits among road users. The results highlight the potential of integrating VLMs to provide human-aligned instructions, which could guide autonomous vehicle actions toward safer and more socially acceptable behaviors and eventually promote AVs' safe and trustable deployment in future intelligent transportation systems.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, Dr. Hamid Taghavifar, for his unwavering support and invaluable guidance and for being a fantastic mentor throughout my M.Sc. student career. I have learned great things from him. The way he teaches, how he interacts with his students, and how he is interested in and dedicated to his research. I can't recall ever knocking on his office door, and he didn't greet me with a smile. I am very grateful for his enthusiasm about our research and willingness to meet me, even on his busiest day. He never limited me in my research and was always open to new ideas. I was very lucky to have this opportunity to work with him.

I would also like to thank my colleagues from the LACITS lab, Neda Shamaeian, Amir Afsharinejad, and Victor Rasidescu, for their invaluable feedback and always being willing to answer my questions. I thank my close friends, Arsalan, Ardalan Aeini, and Shahab Nabavi. They have consistently encouraged and supported me. Their friendship has been a source of strength and has made this journey unforgettable.

Last but not least, I sincerely thank my parents and brother for their constant support, love, and many sacrifices. I am grateful to everyone who has been part of this significant milestone in my life.

List of Acronyms

RL Reinforcement Learning

AV Autonomous Vehicle

PPO Proximal Policy Optimization

DQN Deep Q-Network

CLIP Contrastive Language-Image Pretraining

VLM Vision Language Models

LLM Large Language Model

NLP Natural Language Processing

IDM Intelligent Driver Model

CNN Convolutional Neural Network

ViT Vision Transformer

RNN Recurrent Neural Network

LSTM Long Short-Term Memory Network

FFN Feed-Forward Network

ReLU Rectified Linear Unit

PE Positional Encoding

MDP Markov Decision Process

BCE Binary Cross-Entropy

MSE Mean Squared Error

SGD Stochastic Gradient Descent

WHO World Health Organization

SAE Society of Automobile Engineers

IL Imitation Learning

SAC Soft Actor-Critic

DRL Deep Reinforcement Learning

TD Temporal Difference

RLHF Reinforcement Learning from Human Feedback

Contents

List of Figures

List of Tables

Publications

Some of the ideas, materials and figures in this thesis have appeared previously in the following publications and submitted manuscripts:

(1) **Erfan Doroudian**, Hamid Taghavifar. *Balancing Individual and Collective Interests: Integrating Transformer-Based PPO and Social Intelligence for Autonomous Vehicles at Unsignalized Intersections, IEEE Intelligent Transportation Systems Conference 2024, 24-27th September 2024, Edmonton, Canada.*

(2) **Erfan Doroudian**, Hamid Taghavifar. *CLIP-RLDrive: Human-Aligned Autonomous Driving via CLIP-Based Reward Shaping in Reinforcement Learning.* arXiv:2412.16201v1, 17th December 2024. [Online]. Available: https://arxiv.org/abs/2412.16201

To everyone living far from their loved ones.

Maybe in another universe, we never left home.

Chapter 1

Introduction

1.1 Background

The prospect of autonomous vehicles (AVs) has offered massive potential and benefits for creating efficient, safe, and intelligent transportation systems. The growing interest in the commercialization and deployment of AVs arises from ongoing challenges in the existing transportation systems, such as traffic congestion, road safety, and economic and environmental concerns associated with them. Despite many efforts to reduce road accidents and the resulting deaths, the number of casualties worldwide has significantly increased to over 1.35 million deaths, according to the World Health Organization (WHO) report in 2018 [1]. Table 1.1 summarizes the key human behaviors that contribute to fatal crashes on the road. This table also emphasizes how each of the driving behaviors contributes to the risk of death and serious injury.

In 2014, the Society of Automobile Engineers (SAE) introduced a document called SAE-J3016 to establish consistent terminology for the autonomous driving industry [2]. This document classifies Levels of Driving Automation on a scale of 0 to 5. As shown in Fig 1.1, the higher degrees of automation are linked to the higher SAE automation levels. Although AV technology has progressed over the past couple of decades, most systems today are still at SAE Level 2. The implication is that available and commercial products can mostly deal with aspects such as steering and controlling speed, but the driver always needs to stay alert and ready to take over. A few examples would be Tesla's Autopilot and GM's Super Cruise.

On the other hand, level 3 automation requires the vehicle to handle driving completely in certain situations, so the driver doesn't need to stay focused constantly. There are a few systems starting to offer such products, such as Mercedes-Benz's Drive Pilot and Honda's Sensing Elite in Japan. Transition to level 4 and beyond is much more challenging from technical, regulation, ethical, and legal aspects.

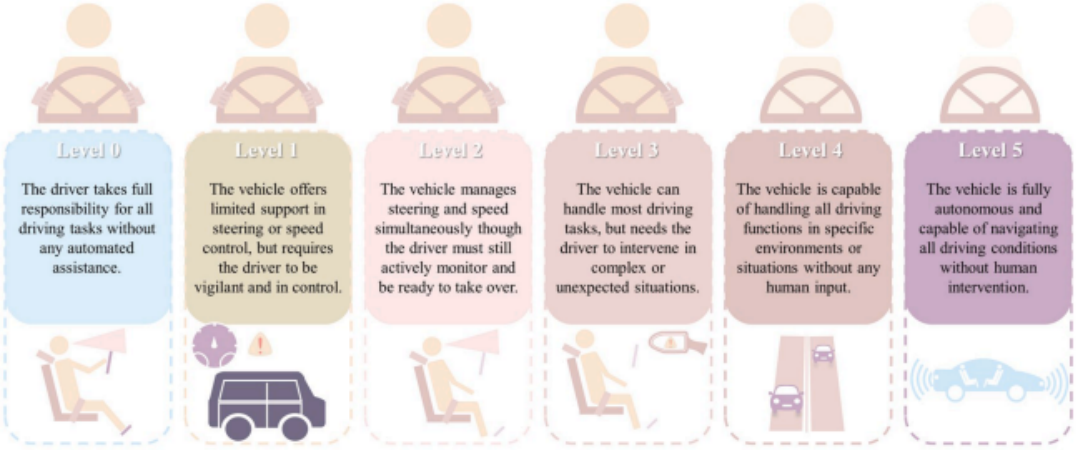


Figure 1.1: Levels of Autonomy as Defined by SAE [3].

One of the mentioned technical challenges is the vehicle's ability to make complex and real-time decisions in dynamic environments. This decision-making process in AVs depends on how perception, planning, and control systems are integrated. Figure 1.2 illustrates the standard blocks of an AV system, which includes sensing elements to actuation components. A vehicle must interpret the data from its surroundings using sensors such as LiDAR, radar, and cameras to obtain a complete geometric understanding of its environment. Subsequently, the vehicle is capable of making logical and safe decisions about navigation, obstacle avoidance, and interactions with other road users. Several important aspects of decision-making in AVs can be explained in the following categories:

(1) **Perception and Environmental Awareness:** The perception step provides the essential information for decision-making processes to the AV by continuously analyzing sensor data to detect objects, pedestrians, and other road users.

(2) **Path Planning and Navigation:** Planning a safe path after the complete perception of the environment is the next step. This step needs real-time decisions to perform smooth lane

changes, safe overtaking, and adherence to traffic regulations. The planning constituent is responsible for immediate actions and long-term goals, such as reaching the destination efficiently through the shortes possible pass.

(3) **Interaction with Human-Driven Vehicles:** In mixed-traffic environments, AVs must anticipate and react to human drivers' unpredictable behavior. This aspect of decision-making is especially challenging, as AVs need to interpret the intentions of other road users and act to avoid conflicts while maintaining traffic flow.

(4) **Handling Uncertainty:** AVs operate in complex environments with high levels of uncertainty. Factors such as adverse weather, sensor inaccuracies, or unforeseen road situations are suggestive of the need to make adaptable decisions concerning the changing driving conditions.

Table 1.1: Risky Driving Behaviors Reported For Drivers Involved In Fatal Crashes [1].

Risk Factor	**Details**
Speeding	Higher speeds increase crash risk and severity, especially for pedestrians and in side impacts.
Driving Under the Influence	Alcohol and drugs greatly increases crash risks; amphetamines raise fatal crash risks by five times.
Non-use of Safety Measures	Helmets, seatbelts, and child restraints reduce death and injury risks by significant percentages.
Distracted Driving	Using mobile phones while driving increases crash risk fourfold, with minimal safety difference between hands-free and handheld devices.

Recent progress in RL, deep learning, and neural networks have led to better decision-making abilities of AVs, particularly in complex and crowded urban environments. By learning from past experiences and simulations, AVs can refine their decision-making strategies and better balance safety with efficiency. Despite this, decision-making is still one of the most difficult aspects of autonomous driving, especially in unpredictable scenarios where human judgment plays a critical role. The future intelligent transportation systems involve vehicles with varying levels of autonomy, including human drivers and artificial drivers. Therefore, it is crucial to incorporate human-like thinking models into decision-making. This approach will improve the interpretability of the decisions and allow for the co-existence of multi-level autonomy on the roads.

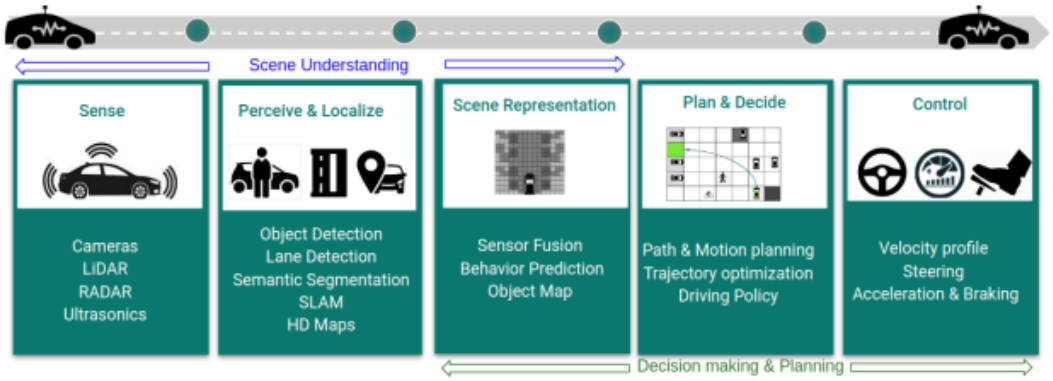


Figure 1.2: Standard components in a modern autonomous driving systems pipeline listing the various tasks [4].

AVs employ various decision-making methods to navigate complex environments safely and efficiently. These methods integrate advanced algorithms, machine learning techniques, and real-time data processing to handle dynamic road conditions and interactions with human drivers. One of the more straightforward methods is rule-based decision-making. This approach follows pre-defined rules and conditions to make decisions. For example, if the vehicle detects a red light, it automatically stops. While rule-based systems are highly reliable in structured environments like highways or controlled intersections, they can struggle in unstructured or unpredictable scenarios where fixed rules may not apply effectively. Another common approach is optimization-based decision-making. Here, the vehicle continuously evaluates multiple possible actions and selects the one that minimizes a particular cost function. This cost function could account for safety, efficiency, and passenger comfort. By balancing these trade-offs, the vehicle can make real-time decisions that ensure safety while achieving long-term goals, such as minimizing fuel consumption or reaching the destination quickly. Machine learning, particularly deep learning, and RL have huge potential to be employed in AV decision-making, especially in more complex environments. Machine learning models are trained on vast amounts of driving data, learning patterns from real-world experiences and simulations. RL, for instance, enables a vehicle to improve its decision-making through trial and error, learning from rewards or penalties assigned to its actions.

An RL agent can be modeled as a Markov Decision Process (MDP). In this framework, the agent operates within the environment by performing actions for which it receives both observations and

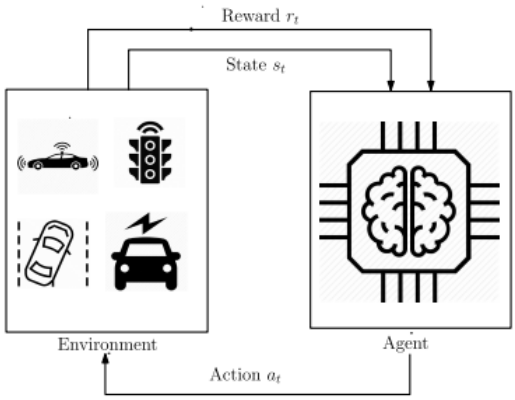


Figure 1.3: A graphical decomposition of the different components of an RL algorithm [6].

rewards. At any given time step, the agent is presented with a certain state from the overall set of possible states $s_t \in S$. Based on the current state; the agent then chooses an action $a_t \in A$ from the set of available actions. Selecting any action is based on the agent's behavior also called the policy. It tells the agent which actions should be selected for each possible state. As a consequence of each action, the agent receives reward $r_t \in R$ and observes the next state $s_{t+1} \in S$ [5]. (see Fig 1.3). Imitation Learning (IL) is another machine learning technique used in decision-making. In this method, AVs learn to make decisions by mimicking the behavior of human drivers. Large datasets of human driving actions in various scenarios are used to train the vehicle, teaching it how to handle similar situations. Although this method is effective, its success depends heavily on the quality and diversity of the data. It may struggle in rare or extreme cases not well-represented in the training data.

Various RL methods with promising applications for AVs can include deep Q-learning, Soft Actor-Critic (SAC), and Proximal Policy Optimization (PPO). In this approach, the AV learns from experience by optimizing a reward function that evaluates its performance in various traffic scenarios. While RL algorithms can optimize the reward function, they might not ensure safety and ethical behavior as they overlook potential risks and consequences [4]. This gap emphasizes the urge to integrate sophisticated decision-making algorithms with human-centered risk assessment and social-behavioral cues is crucial for attaining this objective. RL methods are often sample-inefficient. To reduce interaction time with the environment, reward shaping is commonly used.

This involves designing reward functions that provide the agent with intermediate rewards for making progress toward the goal. However, creating appropriate shaping rewards is known to be difficult and time-consuming [7].

One potential solution for addressing the limitations of traditional reinforcement learning is RL from Human Feedback (RLHF). RLHF enables autonomous agents to learn from predefined reward functions and human-provided feedback, incorporating a more nuanced understanding of safety, ethics, and human behavior. Training the agent with real-time human feedback makes it possible to shape its decision-making process in ways that better align with social expectations and ethical standards, thereby reducing the likelihood of unsafe or undesirable behavior. Furthermore, recent advancements in Large Language Models (LLMs) offer a new opportunity for improving RL frameworks. LLMs can be used to generate context-aware instructions or explanations that provide AVs with additional guidance in complex traffic scenarios. These models can interpret human preferences, predict consequences, and even explain the rationale behind decisions, thus building a hybrid method that links the gap between raw data-driven learning and human-like reasoning. By integrating LLMs with RL agents, the decision-making process of AVs can be enhanced to include not only optimal performance but also socially and ethically aligned actions.

1.2 Literature Review

The use of RL in AVs has gained attention because of its potential to improve decision-making. RL's adaptive learning capabilities in complex and dynamic environments make it a suitable tool for this purpose. Traditional rule-based systems are inadequate for handling such uncertain and dynamic conditions. Advanced techniques such as Deep Reinforcement Learning (DRL) offer a promising solution by enabling vehicles to learn from experiences and enhance performance over time. This section provides an overview of RL techniques and their application in autonomous driving.

1.2.1 Value-Based Methods

Q-learning is one of the most well-known value-based methods. It is an off-policy value-based method that uses a Temporal Difference (TD) approach to train its action-value function. These methods find the optimal policy indirectly by training a value or action-value function that will tell us the value of each state or each state-action pair. Q-Learning is the algorithm that we use to train our Q-function, an action-value function that determines the value of being at a particular state and taking a specific action at that state. The Q-values are updated through the Bellman equation $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$ as the agent interacts with the environment. Where $Q(s,a)$ represents the value of taking action a in state s, r_t is the reward received after taking action a_t at state s_t, $\gamma \in [0,1]$ is the discount factor, and $\alpha \in [0,1]$ is the learning rate. Q-function is encoded by a Q-table, a table where each cell corresponds to a state-action pair value. The problem is that Q-Learning is a tabular method. This becomes a problem if the states and actions spaces are not small enough to be represented efficiently by arrays and tables, which makes it not scalable. To address this issue, Deep Q-Network (DQN) extends Q-learning by using a neural network to approximate the Q-value function, allowing it to handle large and complex state spaces. Instead of storing a Q-table, DQN employs a deep neural network to predict the Q-values for a given state. Additionally, DQN applies the experience replay technique to break the correlation between successive experience samples and also for a better sample efficiency (Fig 1.4). To further enhance stability, DQN uses two neural networks: the parameters of the target network remain fixed for several iterations, while the parameters of the online network are updated.

DQN's ability to manage large and complex state spaces has made it particularly suitable for applications in autonomous driving, where decision-making tasks often involve high-dimensional sensor data, dynamic environments, and real-time constraints. Several studies have demonstrated the effectiveness of DQN in addressing key challenges in autonomous driving scenarios. In [8], two different DQNs (Sequential Actions and Time-to-Go) are trained on a variety of intersection scenarios. The performance is compared against the heuristic Time-to-Collision (TTC) algorithm [9]. The authors also investigate DQNs' ability to learn exploratory behaviors to navigate occluded

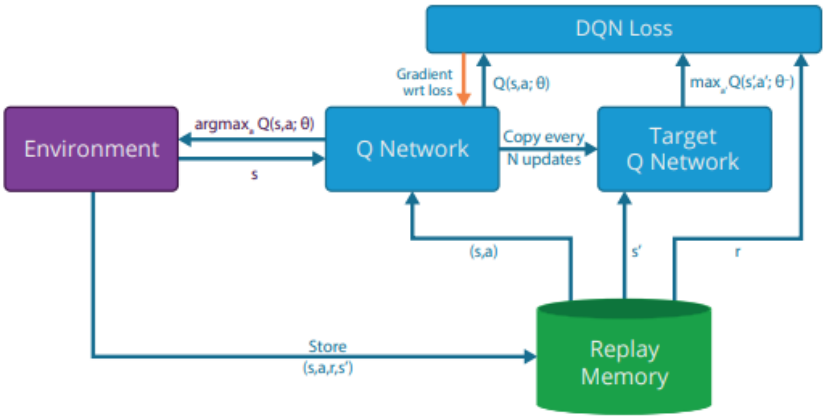


Figure 1.4: The DQN algorithm is composed of three main components: the Q-network $Q(s, a; \theta)$ that defines the behavior policy, the target Q-network $Q(s, a; \theta^{-})$ that is used to generate target Q values for the DQN loss term, and the replay memory that the agent uses to sample random transitions for training the Q-network [12].

intersections. Also, there has been a focus on addressing the challenges of safe navigation in complex environments such as intersections. For instance, in [10], a Deep Q-Network-based agent was trained in a simulation environment for intersection navigation among pedestrians. The authors proposed a grid-based state representation for the agent, which, combined with a carefully designed reward function, allowed the agent to learn policies that ensure safe driving while adhering to traffic rules. Ronecker *et al.* [11] proposed a Deep Q-Network (DQN)-based decision-making framework for autonomous driving, where the DQN serves as the central unit for trajectory planning. The approach was validated through simulation in two distinct highway traffic scenarios, demonstrating its capability to perform lane change maneuvers by combining generated trajectories with a longitudinal movement controller.

1.2.2 Policy-Based Methods

The primary objective of RL is to discover the optimal policy, π^{*}, that maximizes the expected cumulative reward. In policy-based methods, instead of learning a value function, the focus is on directly approximating π^{*}. This is done by parameterizing the policy, for example, with a neural

network π_θ, which produces a probability distribution over actions. The goal is to enhance the performance of this parameterized policy using gradient ascent. Specifically, we adjust the parameter θ to influence the action distribution for each state. Policy-based methods allow us to directly optimize the policy π_θ so that it outputs the action probability distribution $\pi_\theta(a \mid s)$, which results in the highest cumulative return. To achieve this, we define an objective function $J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$, which represents the expected cumulative reward, and aim to find the parameter θ that maximizes this function. Although there are some effective gradient-free approaches in the literature for optimizing policies in non-RL methods [13], gradient-based methods are known to be more useful for policy optimization in all types of RL algorithms. One of the simplest and foundational policy-gradient methods is the REINFORCE algorithm. It uses Monte Carlo sampling to estimate the gradient of the objective function and updates the policy according to $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t$ where G_t is the return following time step t, and α is the learning rate. Although REINFORCE provides an unbiased estimate of the policy gradient, it can have high variance, which leads to slow convergence.

1.2.3 Actor-Critic Methods

Pure policy-gradient methods like REINFORCE suffer from high variance, which can slow down learning. Actor-Critic methods are a hybrid approach that combines the strengths of both policy-based and value-based methods (see Fig 1.5). An Actor that controls how our agent behaves (Policy-Based method), and a Critic that measures how good the taken action is (Value-Based method). Actor-Critic methods are well-suited for problems with continuous action spaces, such as controlling the steering angle or throttle in autonomous driving. The actor can output continuous actions, while the critic helps in improving the policy through value-based feedback. The actor adjusts the policy parameters θ to maximize the expected cumulative reward. In Advantage Actor-Critic (A2C), the learning process is guided by the advantage function defined as $A(s, a) = Q(s, a) - V(s)$. Basically, the advantage function measures how much better an action is compared to the expected value of being in a given state. In other words, this function calculates the extra reward the agent gets if it takes this action at that state compared to the mean reward that it gets at that state. The update rule for the actor can be written as $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a \mid s) A(s, a)$,

where $Q(s, a)$ is the action-value function and $V(s)$ is the state-value function. The critic updates its value function $V_\phi(s)$ to evaluate how good it is to be in a particular state under the current policy. The update for the critic typically uses a TD error, which compares the predicted value of the current state to the observed return. The update rule for the critic is $\phi \leftarrow \phi + \beta \nabla_\phi \left(r + \gamma V_\phi(s') - V_\phi(s)\right)$, where ϕ is the parameter of the value function, and β is the learning rate for the critic network.

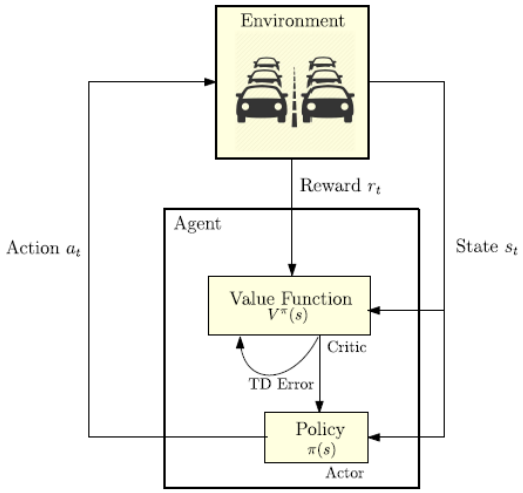


Figure 1.5: Actor Critic control loop [6].

Asynchronous Advantage Actor Critic (A3C) [14] is an asynchronous version of A2C that improves training speed and stability by running multiple agents asynchronously in parallel environments. In this method, each agent interacts with its environment independently and sends updates to a global network. This parallelism reduces the correlation between experiences and makes the learning process more efficient. A3C also uses the advantage function to stabilize policy updates and is effective in complex environments with continuous action spaces. Trust Region Policy Optimization (TRPO) [15] is a policy-gradient method designed to improve stability by ensuring that policy updates are small and within a safe trust region. It uses constrained optimization to limit how much the new policy can diverge from the old one, which prevents destabilizing updates and ensures more reliable learning in reinforcement learning tasks. This is done by enforcing a constraint on the KL divergence between the old and new policies $D_{KL}(\pi_{\text{old}} \mid \pi_\theta) \leq \delta$. TRPO optimizes the expected reward, but it constrains the optimization process to stay within a trust region around the current policy. Proximal Policy Optimization (PPO) [16] has some of the

benefits of TRPO, but it is much simpler to implement, more general, and has a better sample complexity. PPO avoids complex constraints like TRPO by using a clipped objective function $L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right]$ to limit how far the policy can deviate in each update, where A_t is the advantage function, and $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ is the probability ratio between the new and old policies. Also, ϵ is a small hyperparameter that controls how far the ratio can deviate from 1. These methods are also naturally suited to stochastic environments, where exploration and uncertainty are critical, and they excel at learning complex driving behaviors directly from raw sensor inputs which leads to more adaptive and human-like driving strategies. Folkers *et al.* [17] presented a control method for autonomous vehicles using DRL with PPO. They trained an agent to map estimated states to acceleration and steering commands in a simulated environment to achieve real-world applications on a full-size vehicle. Their approach demonstrates successful obstacle avoidance and turning maneuvers in a parking lot scenario. Ref [18] proposed a hierarchical reinforcement learning framework for autonomous driving, using a semi-Markov decision process (SMDP) to handle distinct behaviors like stopping and passing at traffic lights. Their hierarchical policy gradient method separates these behaviors into different modules, significantly improving performance compared to flat reinforcement learning approaches in complex driving scenarios. Tang *et al.* [19] introduced a decision-making and motion planning controller for highway driving scenarios. They utilized Soft Actor-Critic (SAC) to handle continuous action spaces. The proposed controller integrates behavioral decision-making and motion planning layers, allowing the autonomous vehicle to directly control acceleration and steering in dynamic environments. By adjusting the reward function, the authors develop two driving strategies, conservative and aggressive, which balance safety, efficiency, and comfort differently. The SAC-based controller outperforms rule-based and other reinforcement learning methods in terms of safety and efficiency.

1.2.4 Model-Based (vs. Model-Free) & On/Off Policy Methods

Model-based reinforcement Learning (MBRL) is an approach in which the agent learns a model of the environment and uses it to simulate possible outcomes of its actions. Instead of relying solely on direct interactions with the environment (as in model-free methods), model-based RL involves predicting the next state and reward based on the current state and action which allows the agent to

plan actions and evaluate strategies before executing them. The agent builds or learns a model that predicts transitions (next states) and rewards. This model can be represented as a set of transition dynamics $P(s' \mid s, a)$ and a reward function. The model can be given (if the dynamics are known) or learned from data (if the dynamics are unknown). Sample efficiency is one of the advantages of MBRL. The agent can learn from simulated data, and it makes it faster to improve its performance without constantly interacting with the environment. Although this approach has been successful across various domains and is strongly supported by the related field of optimal control [20], a well-recognized limitation of current MBRL algorithms is the limited accuracy of predictions over extended time horizons [21]. In fact, one-step models critically suffer from compounding errors of the dynamics [22], and recent work has shown that the performance of one-step ahead prediction models are not necessarily correlated with downstream control performance [23]. By contrast, Model-Free RL is an approach where the agent learns an optimal policy directly through interactions with the environment without explicitly building or using a model of the environment's dynamics. Model-Free RL often requires fewer assumptions, which makes it more straightforward to apply in environments where modeling is difficult. However, since the agent learns only from direct experience, it may require a large number of interactions with the environment to reach optimal behavior, which makes it sample inefficient.

Learning algorithms are categorized as on-policy or off-policy based on how updates are made. On-policy methods, like SARSA, use the same policy for both estimating value and control. Off-policy methods, like Q-learning, involve two policies: the behavior policy, which generates actions, and the target policy, which is improved upon. This separation allows the target policy to be deterministic (greedy) while the behavior policy can explore all possible actions [4].

1.2.5 Deep Reinforcement Learning (DRL)

Deep Reinforcement Learning (DRL) is a field that combines reinforcement learning with deep learning to enable agents to make decisions in environments with complex, high-dimensional data, such as raw pixel images or complex sensor inputs. Unlike traditional RL, which can struggle with large state spaces, DRL can operate effectively in high-dimensional environments by using neural networks to approximate value functions or policies. Different neural network structures

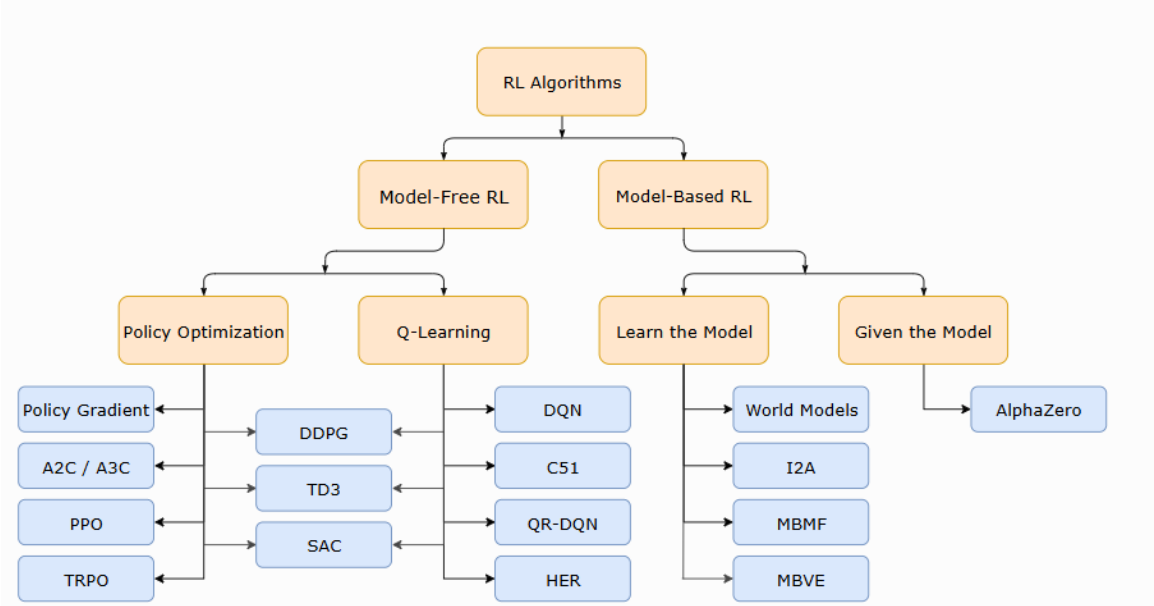


Figure 1.6: RL Taxonomy created by OpenAI .

such as convolutional neural network (CNN) and recurrent neural network (RNN) are used for training RL algorithms in large state spaces [24]. Mnih *et al.* [25] demonstrated how a convolutional neural network could learn successful control policies from just raw video data for different Atari environments. The model processes an 84×84×4 tensor composed of four stacked frames, allowing it to capture temporal information. The input passes through several convolutional layers: the first has 32 filters with 8×8 kernels and a stride of 4, followed by a ReLU activation. The second layer uses 64 filters with 4×4 kernels and a stride of 2, and the third has 64 filters with 3×3 kernels and a stride of 1, both also followed by ReLU activations. Next, a fully connected layer with 512 ReLU units integrates these features into a high-level representation. The final output layer provides a Q-value estimate for each possible action in the given state. Key innovations in DQN include experience replay, where past experiences are stored and randomly sampled to stabilize training, and the target network, a separate network updated periodically to prevent instability during Q-value updates.

Prioritized experience replay is an enhancement of the experience replay mechanism in DQN that addresses a limitation in the original replay buffer by selectively sampling experiences based on their significance to the learning process. In standard experience replay, experiences are sampled

uniformly, meaning each experience has an equal probability of being chosen for training. However, not all experiences contribute equally to the agent's learning. Certain experiences, particularly those with higher TD error, may be more informative for improving the policy [26]. In [27], a multi-agent deep reinforcement learning approach is presented for a 2-by-2 intersection grid model, leveraging max-plus and transfer learning techniques to achieve globally optimal learning through coordination among agents. This approach employs a recurrent neural network (RNN) structure, specifically using LSTM layers, to approximate the Q-function instead of traditional fully connected layers. The study demonstrates that the deep RL method with an RNN structure achieves lower average delays compared to Q-learning and fixed-time control strategies under both low and high-traffic demand conditions. Wu *et al.* [28] presented a human-in-the-loop framework that integrates real-time human guidance into DRL to enhance decision-making for autonomous driving. Their approach, termed Hug-DRL, allows human intervention to correct the agent's actions during training, thereby improving both the convergence rate and overall performance of the DRL policy. This method demonstrates that incorporating human insights into the training loop can improve the robustness and adaptability of autonomous driving systems, particularly in dynamic and unpredictable environments.

1.2.6 Reward Shaping

Since the agent aims to maximize the returns it provides in RL, the design of the reward function is crucial. Thus, the optimal policy for a domain is defined with respect to the reward function. In many real-world applications, learning can be challenging due to rewards that are sparse or delayed [4]. RL agents typically learn how to act in an environment based solely on the guidance of this reward signal. To enhance learning speed and final performance, additional knowledge can be introduced through a shaping reward, added to the reward naturally provided by the environment. This process, known as reward shaping, originates from experimental psychology, where it involves reinforcing all actions that contribute toward the desired behavior [4]. A typical implementation of reward shaping modifies the reward function $R'(s,a) = R(s,a) + F(s,a)$, where R is the original environment reward, F is the shaping reward, a function designed to provide additional feedback about desirable intermediate actions or states, and R' is the signal given to the agent.

While reward shaping can help an agent learn faster by adding extra guidance, it can sometimes cause problems. Specifically, when you add a shaping reward to the original reward, the agent ends up optimizing a new, augmented reward function R'. The issue here is that a policy that is optimal for this augmented reward function, may not actually be optimal for the original reward function R. The agent might end up with behaviors that maximize the shaping reward in ways that are different from, or even in conflict with, the original task goal defined by R. A classic example of reward shaping gone wrong for this exact reason was reported by [29], where the experimented bicycle agent would turn in a circle to stay upright rather than reach its goal. Several techniques have been proposed to address the potential issues with reward shaping. These methods aim to ensure that the agent's behavior aligns with the original task goal rather than being overly influenced by the shaping rewards. Potential-based reward shaping [30] ensures that the optimal policy for the augmented reward function is also optimal for the original reward function. It defines the shaping reward in a way that guarantees consistency by using a potential function, where the shaping reward is defined as $F(s, s') = \gamma\Phi(s') - \Phi(s)$. This approach guarantees that adding the shaping reward does not change the optimal policy, as it merely re-scales the original rewards without affecting the agent's learning in unintended ways.

Many tasks involve goals that are complex, poorly defined, or hard to specify. Addressing this challenge would significantly broaden the potential applications of DRL and could also extend the overall impact of machine learning. For instance, consider training a robot to clean a table or scramble an egg using RL. Designing an appropriate reward function tied to the robot's sensor data is challenging; a simple reward function might capture aspects of the task but could lead the robot to optimize behaviors that miss the true objective. This challenge contributes to concerns about the alignment of RL systems with human values and intentions. If we have demonstrations of the desired task, we can apply inverse reinforcement learning [31] or imitation learning to replicate the behavior. However, these methods are not well-suited for tasks that are difficult for humans to demonstrate, such as controlling a robot with a non-human-like physical structure or high degrees of freedom. An alternative approach is to allow a human to provide feedback on our system's current behavior and to use this feedback to define the task [32]. Reinforcement Learning from Human Feedback (RLHF) is a technique where human feedback is used to guide the training process of

a RL agent, especially when designing a reward function is difficult. In RLHF, humans provide explicit feedback on the agent's actions, and this feedback is used to shape the reward function.

Christiano *et al.* [32] introduced a framework where DRL is guided by human feedback to optimize agent behavior for complex tasks with difficult-to-specify reward functions. Their method uses human preferences over pairs of trajectory segments to learn a reward model, which is then used by the agent to maximize the intended behavior. By integrating human feedback in this way, the approach is shown to handle tasks in Atari games and simulated robotics, achieving successful performance while requiring significantly less human oversight. Ref [33] examined how well LLMs align with human intentions. It was reported that merely increasing model size does not enhance responsiveness to user requests. In this study, the InstructGPT framework was introduced, which fine-tunes LLMs using RLHF. By training on human-labeled examples and refining through reward modeling, InstructGPT significantly outperforms the baseline GPT-3 in user preference and instruction adherence. Additionally, this study indicated improvements in truthfulness and reduced toxic outputs and emphasized the potential of human feedback for better-aligning LLMs with human intentions and minimizing risks.

However, collecting human feedback is an expensive and complex task. Therefore, using pre-trained foundation models themselves to obtain reward signals for RL finetuning has recently emerged as a key paradigm in work on large language models [34]. Some approaches only require a small amount of natural language feedback instead of a whole dataset of human preferences [35]. However, similar techniques have yet to be adopted by the broader RL community [36]. While some work uses language models to compute a reward function from a structured environment representation [37], many RL tasks are visual and require using Vision-Language Models (VLMs) instead. Ref [38] proposed a novel method for training embodied agents to understand and follow natural language instructions by using pre-trained VLMs. The authors argue that VLMs, trained on vast amounts of internet data, possess a general understanding of language grounding, which can be adapted to teach specific tasks to embodied agents. Their approach leverages the VLM's ability to generate descriptive text based on images, using it to retroactively label the agent's actions, creating training data that bridges the gap between the agent's perception and its language understanding. Basically, they use VLMs to relabel the goal of agent trajectories for hindsight experience replay

but not for specifying rewards.

1.3 Problem Statement

To ensure the safe large-scale use of AVs, one should consider different technical, social, and ethical issues and integrate them into the decision-making processes of these vehicles [39]. Current decision-making systems mainly utilize rule-based methods, which are designed with a specific set of instructions or rules to handle various driving scenarios [40]. Additionally, creating a strong reward function that effectively reflects the complexities of ethical and context-sensitive driving behavior is not a straightforward task, and basic reward functions can result in unexpected behaviors. Manually adjusted/tuned rewards need significant resources and knowledge. In addition, the latter approach also struggles to generalize across various driving scenarios. One promising direction to address these challenges is to integrate VLMs to provide context-aware instructions or guidance. VLMs, trained on vast datasets, can interpret complex scenarios and generate human-aligned instructions that could help steer an AV's actions toward safer and more socially acceptable behaviors. However, the present research attempts to study how VLMs can potentially support RL in autonomous driving by providing scenario-specific feedback to enhance training. In this work, we utilize VLMs to generate high-level, context-aware instructions for AVs in an unsignalized intersection environment. These VLM-generated instructions function as additional guidance for the RL agent, which helps it to align its actions with human-like decision-making preferences.

1.4 Research Scope and Objectives

This work aims to develop and evaluate a novel framework that combines RL with VLMs to improve AV decision-making in unsignalized intersections, with a focus on mimicking human-like behaviors. The challenge of managing AV behavior at intersections without traffic signals requires an understanding of contextual, human-like decision-making processes. By leveraging CLIP as a VLM to build a reward model, this study aims to enable the AV to make safe and efficient decisions in response to real-time visual inputs. Additionally, we collect a data set (visual scenario, description) pair to fine-tune the CLIP using our dataset. To reach that, we use the transfer learning

technique by calculating the gradient only for the upper layer of CLIP to increase training efficiency. Once fine-tuned, CLIP can associate diverse visual scenarios with three distinct actions in the simulation environment to recommend the most appropriate action for the agent at each time step based on visual observation. This fine-tuned model is then integrated into the AV's decision-making system. In this thesis, we utilize DQN and PPO as RL algorithms. The agent is trained using a combination of the environment's basic reward function and additional rewards from the CLIP model. This dual reward structure encourages the agent to align its actions with the guidance from CLIP's reward model, which helps it to make decisions that reflect desired human-like behavior. The effectiveness of the proposed framework is demonstrated using the *Highway-env* simulation environment, specifically its intersection scenario, which is built on OpenAI Gym.

The contributions of this work can be summarized as follows:

- **Use of CLIP as a Reward Model in RL**: A key contribution is the adaptation of CLIP as a reward model for guiding AV behavior. By fine-tuning CLIP on a dataset of visual scenarios and their corresponding descriptions, this research utilizes CLIP's contextual understanding to provide real-time guidance to the AV to align its actions with safe and human-inspired behaviors.

- **Fine-Tuning Through Transfer Learning**: To optimize the training process, this study utilizes transfer learning by fine-tuning only the upper layer of CLIP while keeping the remaining layers frozen. This approach reduces computational demands, allows for efficient training with a limited dataset, and enhances the practicality of the framework.

- **Integration of CLIP into a Dual Reward Structure**: The research incorporates CLIP's recommendations as a secondary reward signal, combined with the environment's basic reward function, within the RL framework. This dual reward structure ensures that the AV's actions not only maximize traditional rewards but also align with CLIP's human-like guidance to improve decision-making quality.

- **Evaluation Using DQN and PPO Algorithms in a Realistic Simulation**: The framework's effectiveness is demonstrated through experiments using DQN, and PPO algorithms in the *Highway-env* simulation environment.

Chapter 2

Simulation Environment Setup

A controlled simulation environment is essential to develop and evaluate the decision-making capabilities of autonomous vehicles in complex traffic scenarios. This setting enables safe and repeatable testing of strategies in various, often challenging conditions that are hard to reproduce in practice. In this section, the tools used for the simulation of the environment and data collection are presented.

2.1 Environment

The simulations are performed in the intersection environment, from *highway-env* [41]. It is based on Open AI's Gym environment toolkit for developing and comparing reinforcement learning algorithms [42]. The environment can simulate different traffic situations such as a highway, roundabout, and intersection. In these situations, the cars could either be agents or considered to be driven by humans. It renders a 2D environment, top-down view, and accurately simulates automobile behavior with front wheel steering through the Kinematic Bicycle Model [43].

2.1.1 Kinematics

The Kinematic Bicycle Model represents the vehicle's kinematics in the "Vehicle" class. This more realistic representation captures the effect of steering angle and slip angles on the vehicle's movement. It approximates the car as a bicycle, meaning the vehicle's four wheels are simplified

into two—one for the front axle and one for the rear axle.

$$\dot{x} = v\cos(\psi + \beta) \tag{1}$$

$$\dot{y} = v\sin(\psi + \beta) \tag{2}$$

$$\dot{v} = a \tag{3}$$

$$\dot{\psi} = \frac{v}{l}\sin\beta \tag{4}$$

$$\beta = \tan^{-1}\left(\frac{1}{2}\tan\delta\right) \tag{5}$$

Where (x, y) is the vehicle position, v is the forward speed, ψ is the heading angle, a is the acceleration command, β is the slip angle at the center of gravity, and δ is the front wheel angle used as a steering command.

2.1.2 Route Planning

The route planning layer determines the sequence of road segments the vehicle should follow to reach its destination. For the ego vehicle, this destination is fixed, while for other simulated vehicles, it is chosen randomly at the start of the simulation. To plan the route, the layer employs the Breadth-First Search (BFS) algorithm on a graph representation of the road network. BFS is simple to implement and computationally efficient for relatively small and structured graphs like a road network in a simulation environment; it identifies the shortest path by systematically exploring each road segment step-by-step, completing all nodes at the current level before moving to the next. Also, it uses a queue to store the nodes that will be visited next and keep track of visited nodes to avoid cycles in the graph. So, BFS traverses every node and edge exactly once in a layer-wise order as shown in Fig. 2.1.

2.1.3 Motion Planning and Control

The ego vehicle and other vehicles $i \in [0, N_v]$ in the scene are equipped with a capability to execute the meta actions $\mathcal{A}$. This requires the ability to follow a lane L_i described by the road

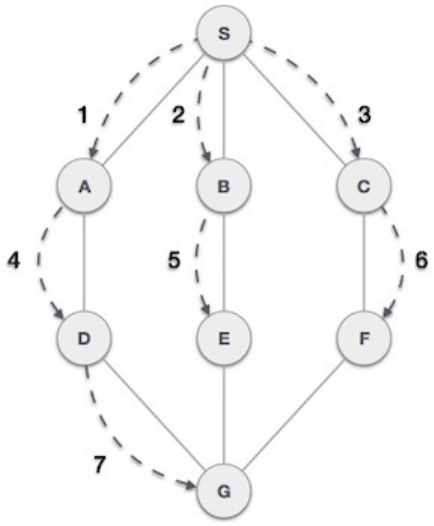


Figure 2.1: BFS starts at the root and explores nodes level by level [44].

information through its lateral position p_{Li}^{y} and heading ψ_{Li}. To that end, vehicles follow a cascade controller of lateral position and heading in the form:

$$\dot{\psi}_i = K_i^{\psi} \left(\psi_{Li} + \sin^{-1} \left(\frac{\tilde{v}_{i,y}}{v_i} \right) - \psi_i \right) \tag{6}$$

$$\tilde{v}_{i,y} = K_i^{y} (p_{Li}^{y} - p_i^{y}) \tag{7}$$

where $K_i^{y} \in \mathbb{R}$ and $K_i^{\psi} \in \mathbb{R}$ are control gains. Once the heading rate $\dot{\psi}$ is calculated, the required steering command for the vehicle would be determined as:

$$\beta_i = \sin^{-1} \left(\frac{l}{v_i} \dot{\psi}_i \right) \tag{8}$$

Furthermore, the ego-vehicle needs to be able to control its speed as per the meta actions $\mathcal{A}$. To do that, a longitudinal controller considers the vehicle's current speed, the desired speed (target), and the difference between the two to compute the necessary throttle/brake action. So, this equation describes how the vehicle achieves the speed determined by meta actions $\mathcal{A}$:

$$\dot{v}_0 = K_0^{v} (v_r - v_0) \tag{9}$$

where $v_r \in \mathbb{R}$ is the reference speed by the drive $faster$ and drive $slower$ meta-actions, and $K_0 \in \mathbb{R}$ is a control gain.

2.1.4 Behavioral Models

Other simulated vehicles follow simple and realistic behaviors that dictate how they accelerate and steer on the road.

Longitudinal behaviour: The acceleration command a_i of a vehicle $i \in [1, N_v]$ is controlled directly by the *Intelligent Driver Model (IDM)* from [45]:

$$\dot{v}_i = a_i \left[1 - \left(\frac{v}{v_0^i} \right)^{\delta} - \left(\frac{d_i^*}{d_i} \right)^2 \right] \tag{10}$$

where

$$d_i^* = d_0^i + T_i v_i + \frac{v_i \Delta v_i}{2\sqrt{a_i^i b_i}}$$

v_i is the vehicle velocity, d_i is the distance to its front vehicle. The dynamics of vehicle $i \in [1, N_v]$ are thus parametrized by the desired velocity v_0^i, the time gap T_i, the jam distance d_0^i, the maximum acceleration a_i and deceleration b_i, and the velocity exponent δ.

2.1.5 Rewards

The design of the reward function is crucial in RL, as it directly influences the agent's behavior and learning efficiency. For the intersection environment in *highway-env*, we define a parametrized reward function that includes three primary components. We show $R(s, a)$ as the reward received when the agent takes action a in state s and define it as:

$$R(s,a) = \begin{cases} r_{\text{speed}} & \text{if the agent maintains an efficient speed,} \\ r_{\text{collision}} & \text{if the ego-vehicle collides with another vehicle,} \\ r_{\text{destination}} & \text{if the agent successfully reaches its destination,} \\ 0 & \text{otherwise.} \end{cases}$$

where $r_{\text{speed}} \in \mathbb{R}_+$ represents the reward for maintaining an efficient speed while driving,

$r_{\text{collision}} \in \mathbb{R}_-$ denotes the penalty for a collision with another vehicle, and $r_{\text{destination}} \in \mathbb{R}_+$ is the reward for successfully reaching the destination. In this study, the values were chosen as $r_{\text{speed}} = 1$, $r_{\text{collision}} = -5$, and $r_{\text{destination}} = 2$. These values were selected to balance the agent's objectives effectively. Maintaining an efficient speed ($r_{\text{speed}} = 1$) encourages the agent to drive at speeds conducive to smooth traffic flow, promoting timely progression through the environment. The collision penalty ($r_{\text{collision}} = -5$) imposes a significant negative reward to strongly discourage collisions, thereby emphasizing safety. Additionally, the destination reward ($r_{\text{destination}} = 2$) provides sufficient incentive for the agent to reach its goal, ensuring that task completion remains a priority. These components make up the basic reward structure provided by the simulation environment in *highway-env*. While this foundational reward function effectively encourages fundamental behaviors like safety, speed, and goal completion, it may not fully capture nuanced aspects of driving comfort or complex safety requirements in dynamic environments. In the next chapters, we extend this basic reward function by incorporating additional terms informed by VLMs. These models allow us to account for human-like preferences, such as comfort and safety, by generating context-aware instructions. Through this approach, the reward function evolves to provide a more comprehensive assessment of driving performance, which aligns the agent's actions more closely with the intuitive and feasible expectations and standards.

2.1.6 Observation and Action Space

An observation space, often in the form of a matrix, is what the agents can see before they take an action. The matrix should contain the information needed for the agent to take an action that leads to a high reward. *Highway-env* offers different types of observation spaces based on the complexity and information needs of the environment setup:

- **Kinematics Observation:** This observation type provides the ego-vehicle and neighboring vehicles' position, velocity, and heading angle.
- **Grayscale Observation:** Instead of a structured kinematic data observation, this observation type represents the environment as a grayscale image (similar to a top-down view), where the positions of vehicles and road boundaries are visualized as pixel intensities.

- **Occupancy Grid Observation:** These observation types categorize the space around the ego-vehicle in a uniform grid of cells. Each cell has some feature attributions that describe in a cell if a vehicle exists as well as some kinematics such as position offsets, velocities, and heading angles. The size and resolution of the grid are configurable. By structurally representing the space around them, this helps the agent evaluate the motion of the surrounding traffic flow effectively.

- **Time to Collision Observation:** This observation type predicts the time of collision with the observed vehicles that are on the same road in view of the ego-vehicle. The observation will be an array projecting the estimated time for which a potential collision will happen with any of the surrounding vehicles, considering their velocities and trajectories. By including these estimations, the agent can better anticipate what imminent hazards might be and give it a greater capability to make proper decisions in improved safety conditions.

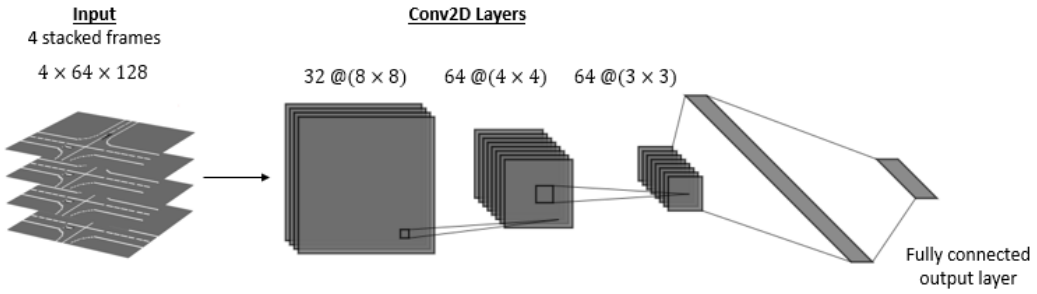


Figure 2.2: DQN: Convolutional Neural Network with the last 4 frames.

The observation space used in this simulation study is grayscale observation, a $W \times H$ grayscale image of the scene, where W and H are set with the observation shape parameter. Several images can be stacked with the stack size parameter. The agent lacks information about recent changes or movements when the environment only provides a single frame as an observation. For example, in a driving scenario, it's challenging to determine the speed or direction of other vehicles with just a single snapshot. As a result, we stacked the four most recent frames together to capture temporal information, providing the model with a sense of motion and changes in the environment. This stacked representation was fed into a Convolutional Neural Network (CNN) architecture within the

DQN framework (Fig 2.2). By processing these consecutive frames, the CNN could learn spatial and temporal features that enhance the model's decision-making capabilities in dynamic scenarios. The agent must drive a vehicle by controlling its acceleration chosen from the set of abstract meta-actions in a predefined path from the starting point to the target point. To that end, we specify the following space of meta-actions:

$$\mathcal{A}_i = [\texttt{Drive Slower, Maintain Speed (IDLE), Drive Faster}] \quad (11)$$

Meta-actions are rather slow to affect the vehicle's state and are thus executed at a low frequency of 1 Hz. This design choice simplifies the action space, which in turn causes the agent to concentrate its efforts on strategic speed adjustments rather than deal with fine tuning of control inputs. Executing actions at a lower frequency also make the smoother vehicle behavior possible and improves the stability of the learning process.

Chapter 3

Reinforcement Learning Algorithms

In RL, an agent interacts with an environment to learn a policy that maximizes cumulative rewards. The policy π is the function that specifies for the agent what action to take given the state. So it defines the agent's behavior at a given time. There are two approaches to train an agent to find this optimal policy π^*: 1) **Value-based** methods that intend to learn the optimal value function, which estimates the expected cumulative reward from each state (or state-action pair) following a specific policy. The agent indirectly derives the policy by selecting actions that maximize this value function. 2) In **Policy-based** methods, the agent directly learns a policy π which maps states to a distribution over actions. Also, the actor-critic method is a hybrid approach that combines value-based and policy-based methods to help stabilize the training by reducing the variance. In this thesis, we trained our agent using DQN, a value-based method, and PPO, an actor-critic method. We then compared the performance and results of both approaches.

3.1 Deep Q-Network (DQN)

In Chapter 1, we discussed that Q-Learning is a value-based algorithm we use to train the Q-Function, an action-value function that determines the value of being at a particular state and taking a specific action at that state. Q-function is encoded by a Q-table, in which each cell is linked to a state-action pair value. As the agent interacts with the environment, the Q-values are updated

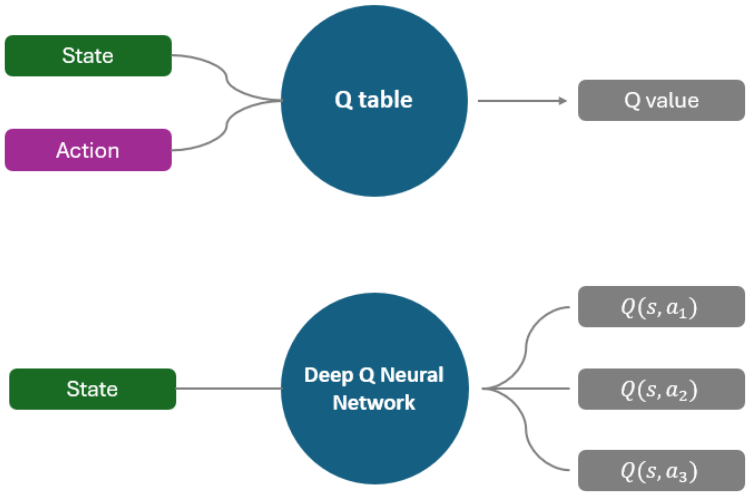


Figure 3.1: Difference between DQN and tabular Q-learning.

through the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{12}$$

where $Q(s,a)$ denotes the value of taking action a in state s, $\gamma \in [0,1]$ is the discount factor, and $\alpha \in [0,1]$ is the learning rate. The primary problems with tabular Q-learning are related to scalability and the inability to handle complex state spaces. As the state and action spaces grow, the number of entries in the Q-table increases exponentially. This becomes infeasible when we have high-dimensional or continuous state spaces where there could be an infinite number of states.

To address these issues, Deep Q-Network (DQN) was introduced [25], where a neural network is used to approximate the Q-function $Q(s,a;\theta)$ with θ representing the parameters of the neural network. This network can approximate the Q-values for each action given a high-dimensional input state (see Fig. 3.1).

3.1.1 Epsilon-Greedy Exploration

In RL, exploration and exploitation are two fundamental concepts that address how an agent chooses actions to maximize its reward over time. Exploration is the process by which an agent tries

out new actions, even if they may not seem immediately rewarding. Exploration aims to discover more about the environment and find potentially better actions that could lead to higher long-term rewards. Exploitation is exploiting known information to maximize the reward. In exploitation, the agent leverages its past experiences to maximize its reward in the current moment without trying to gather new information. The key challenge in RL is to balance exploration and exploitation. To address the limitations of a purely greedy approach, a common method is the ϵ-greedy strategy. In this approach, the agent chooses a random action with probability ϵ and exploits the best-known action with probability $1 - \epsilon$. Initially, ϵ is set to a high value to encourage exploration. Over time, as the agent gathers more experience and learns about the environment, ϵ is gradually reduced following a decay rate, so the agent shifts more toward exploitation. This strategy allows the agent to explore sufficiently in the early stages and refine its policy by focusing on actions that lead to higher rewards in later stages.

3.1.2 Experience Replay

Usually, in online RL, the agent interacts with the environment, gets experiences, learns from them, and discards them. This is not an efficient approach. DQN addresses this by storing experiences (s, a, r, s') in a replay buffer and sampling mini-batches of experiences randomly for training. In this experience replay method, the correlation between consecutive samples is dismantled, which improves the stability and efficiency of training.

3.1.3 Fixed Q-Target To Stabilize The Training

In Deep Q-Learning, we use two neural networks in the learning process. These networks have the same architecture but different weights. In every N step, the weights from the main network are copied to the target network. Using both networks leads to more stability in the learning process and helps the algorithm learn more effectively. The main and target neural networks map input states to an (action, q-value) pair. In this case, each output node representing an action contains the action's q-value. When calculating the TD error (loss), we calculate the difference between the TD target and the current Q-value. However, as we don't know the real TD target, we need to estimate it using

the Bellman equation:

$$y = R_{t+1} + \gamma \max_{a'} Q_{\text{target}}(s_{t+1}, a'; \theta') \tag{13}$$

The goal of DQN is to minimize the difference between the Q-network's prediction and the target Q-value calculated using the target network. This is done using a mean-squared error (MSE) loss function:

$$L(\theta) = \mathbb{E}\left[(y - Q(s, a; \theta))^2\right] \tag{14}$$

where the parameters θ are updated using gradient descent, with optimizers like stochastic gradient descent (SGD) or Adam.

Algorithm 1 Deep Q-Learning Algorithm

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    if every N steps **then**
      Reset $\hat{Q} = Q$
    end if
  end for
end for

3.2 Policy Based Methods

3.2.1 REINFORCE vs. Advantage Actor-Critic

The Vanilla Policy Gradient (VPG) method is an on-policy algorithm that can be used for environments with either discrete or continuous action spaces. The fundamental concept behind policy gradients is to increase the likelihood of actions that result in greater returns and decrease the likelihood of actions that lead to lower returns until the optimal policy is achieved. So for each state-action pair, we want to increase the $P(a \mid s)$, the probability of taking that action at that state which leads to a higher reward, or decrease it if it leads to a lower reward. The simplified version of a policy-gradient method is shown in Algorithm 2. To evaluate the effectiveness of a policy, we

Algorithm 2 Simplified Training Loop for Policy Gradient Method

Collect an **episode** with the policy π.
Calculate the **return** (sum of discounted rewards).
Update the weights of π:
if return is positive **then**
    increase the probability of each (s, a) pair taken during the episode.
else
    decrease the probability of each (s, a) pair taken during the episode.
end if

define an objective function called $J(\theta)$. The objective function gives us the performance of the agent given a trajectory. Each trajectory represents a sequence of state-action pairs:

$$\tau = (s_0, a_0, s_1, a_1, ...) \tag{15}$$

Finally, $J(\theta)$ outputs the expected cumulative reward.

$$J(\theta) = \mathbb{E}_{\tau \sim \pi} \left[R(\tau) \right], \tag{16}$$

$$R(\tau) = \left[\sum_{t=0}^{T} \gamma^t r_t \right] = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots \tag{17}$$

where $\gamma \in [0, 1]$ is the discount factor, and τ indicates the given trajectory. To give more details on this formula, we calculate the expected return $J(\theta)$ by summing for all trajectories, the probability

of taking that trajectory given θ, and the trajectory's return.

$$J(\theta) = \sum_{\tau} P(\tau;\theta) R(\tau), \tag{18}$$

$$P(\tau;\theta) = \rho_0(s_0) \prod_{t=0}^{T} \left[P(s_{t+1} \mid s_t, a_t)\, \pi_\theta(a_t \mid s_t) \right] \tag{19}$$

where $\rho_0(s_0)$ gives the probability of starting in the initial state. Then, our goal is to maximize the expected cumulative reward by finding the θ that will output the best action probability distributions.

$$\max_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[R(\tau) \right] \tag{20}$$

Policy gradient methods optimize the policy using gradient ascent on the expected reward. It's the inverse of gradient-descent since it gives the direction of the steepest increase of $J(\theta)$. The update rule for each gradient step is:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{21}$$

where α is the learning rate, which determines the step size in the gradient direction, and $\nabla_\theta J(\theta)$ is the gradient of the objective function with respect to θ. This step updates θ to increase the likelihood of actions that yield higher returns, thus improving the policy over time. The issue is that many possible trajectories in complex environments with many states and actions are extremely large. So, instead of calculating the true gradient using all possible trajectories, we use a sample-based estimate of the gradient by collecting a set of sample trajectories by running the current policy in the environment, and these sampled trajectories give us an estimate of the expected cumulative reward. We approximate the true gradient without evaluating every possible trajectory by taking an average over these sampled trajectories. In RL, we model the environment as a Markov Decision Process (MDP). The MDP includes:

- **State transition dynamics:** The probability of transitioning to the next state s_{t+1}, given the current state s_t and the action a_t taken by the agent.
- **Reward:** The immediate reward for each state-action pair.

The second issue is that the objective function relies on the environment's transition dynamics, which we cannot differentiate since these dynamics are typically unknown or complex components of the environment.

$$\nabla_\theta J(\theta) = \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) \tag{22}$$

To address this, the policy gradient theorem is applied, which allows us to reformulate the (Eq. 19) using log-derivative trick into a differentiable form that does not require differentiating the state distribution.

$$\log P(\tau;\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T} \left(\log P(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t)\right) \tag{23}$$

Since the environment has no dependence on θ, the gradient of $\rho_0(s_0)$ and $P(s_{t+1} \mid s_t, a_t)$ would be zero.

$$\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \tag{24}$$

Putting it all together, we derive the following:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right] \tag{25}$$

where $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is the log-probability gradient of taking action a_t in state s_t. Since the exact gradient $\nabla_\theta J(\theta)$ is generally difficult to compute, we use Monte Carlo sampling to estimate it by sampling trajectories from the environment. For each trajectory τ, we can estimate the gradient as:

$$\nabla_\theta J(\theta) \approx \hat{g} = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \tag{26}$$

We can also collect multiple trajectories to estimate the gradient:

$$\nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \tag{27}$$

where $|\mathcal{D}|$ is the number of trajectories in $\mathcal{D}$. One common issue with vanilla policy gradients (e.g., REINFORCE) is high variance in the gradient estimates. Given the environment's stochasticity and

the policy's stochasticity, trajectories can lead to different returns, which can lead to high variance. Consequently, the same starting state can lead to very different returns. Advantage functions are used to reduce this variance, and thus, more stable learning can be achieved. The advantage function $A(s, a)$ measures how much better an action is compared to the expected value of being in a given state. In other words, this function calculates the extra reward the agent gets if it takes this action at that state compared to the mean reward it gets at that state.

$$A(s, a) = Q(s, a) - V(s) = r + \gamma V(s') - V(s) \tag{28}$$

Using the advantage function, the policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A(s_t, a_t) \right] \tag{29}$$

Algorithm 3 Policy Gradient Algorithm with Advantage Function

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ based on the current value function V_{ϕ_k}.

Estimate policy gradient as

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t$$

Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

via gradient descent algorithm.

end for

Algorithm 3 shows the policy gradient algorithm with an Advantage Function. Advantage Actor-Critic (A2C) is a hybrid architecture that combines value-based and policy-based methods

to stabilize the training by reducing the variance. Getting good results via policy gradient methods is challenging because they are sensitive to the choice of step size. If the step size is too small, the policy updates will be minor, and the agent learns slowly. On the other hand, if the step size is too large, each update can overshoot the target, which leads to noisy and unstable learning. They also often have very poor sample efficiency. Sample inefficiency refers to the need for many interactions with the environment (timesteps) for the agent to learn effectively. In RL, the concepts of old and new policy generally refer to the policy before and after an update step. The old policy might perform well in many RL algorithms, especially those that involve stochastic environments. Making large jumps to a new policy without caution can destabilize performance. So, by comparing old and new policies, we can better control these changes.

3.2.2 Proximal Policy Optimization (PPO)

PPO is a model-free on-policy policy gradient method proposed by [16]. The idea with PPO is that we want to improve training stability by avoiding large policy updates. It is based on the Trust Region Optimization (TRPO) method [15], that aims to maximize an objective function (the "surrogate" objective) subject to a constraint on the size of the policy update:

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta \left(a_t \mid s_t \right)}{\pi_{\theta_{\text{old}}} \left(a_t \mid s_t \right)} \hat{A}_t \right] \tag{30}$$

$$\text{subject to } \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}} \left(\cdot \mid s_t \right), \pi_\theta \left(\cdot \mid s_t \right) \right] \right] \leq \delta \tag{31}$$

where θ_{old} denotes the old policy parameters, and θ denotes the current ones. Nevertheless, TRPO is computationally expensive since it uses conjugate gradient ascent and a line search to enforce the hard KL divergence constraint, which makes it hard to use in the real-time learning setting. PPO simplifies objective (Eq. 30) to reduce the computation cost. It has two variants. The first variant treats the KL divergence as a penalty, modifying the objective into the following unconstrained objective:

$$\underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta \left(a_t \mid s_t \right)}{\pi_{\theta_{\text{old}}} \left(a_t \mid s_t \right)} \hat{A}_t - \beta \text{KL} \left[\pi_{\theta_{\text{old}}} \left(\cdot \mid s_t \right), \pi_\theta \left(\cdot \mid s_t \right) \right] \right] \tag{32}$$

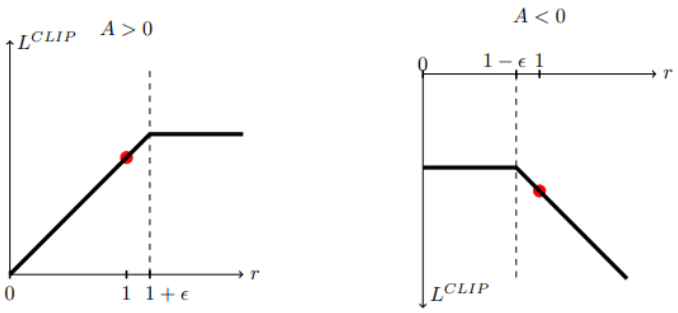


Figure 3.2: Plots showing one term (i.e., a single timestep) of the surrogate function L^{CLIP} as a function of the probability ratio r_θ [16].

where $\mathrm{KL}\left[\pi_{\theta_{\text{old}}}\left(\cdot \mid s_t\right), \pi_\theta\left(\cdot \mid s_t\right)\right]$ measures the distance between the new and old policies for a given state. Also, β is a scaling factor for the KL divergence penalty term in the objective function. A smaller β allows for more exploration and faster learning at the expense of potentially larger policy updates. This approach is called the Adaptive KL Penalty Coefficient. The second approach does not explicitly include the KL divergence penalty in the objective but instead implicitly respects the KL divergence by removing samples whose action probabilities are significantly different from the current policy by using specialized clipping. Specifically, this approach maximizes a Clipped Surrogate Objective defined as:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)A_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)A_t\right)\right] \tag{33}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ is the probability ratio between the new and old policies. If $r_\theta > 1$, the action a_t at s_t is more likely in the current policy than the old policy, and if $0 < r_\theta < 1$, the action is less probable for current policy than for the old one. So this probability ratio is an easy way to estimate the divergence between old and current policy, which in [16], the ratio can only vary from 0.8 to 1.2. As Fig 3.2 summarizes the effect of L^{CLIP} function, when $r_t(\theta) \in [1-\epsilon, 1+\epsilon]$, the clipping does not apply. On the left half of the diagram, where $(A > 0)$, the action is better than the average of all the actions in that state. Therefore, we should encourage our current policy to increase the probability of taking that action in that state. Conversely, on the right half of the diagram, where $(A < 0)$, the action is worse than the average of all actions at that state. Therefore, we should discourage our current policy from taking that action in that state. So basically, we only

update the policy with the unclipped objective part.

$r_t(\theta) > 0$	A_t	Objective is Clipped	Sign of Objective	Gradient
$r_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$+$	no	$+$	✓
$r_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$-$	no	$-$	✓
$r_t(\theta) < 1-\epsilon$	$+$	no	$+$	✓
$r_t(\theta) < 1-\epsilon$	$-$	yes	$-$	0
$r_t(\theta) > 1+\epsilon$	$+$	yes	$+$	0
$r_t(\theta) > 1+\epsilon$	$-$	no	$-$	✓

Table 3.1: Summarizing the behavior of PPO's objective function. (from "Towards Delivering a Coherent Self-Contained Explanation of Proximal Policy Optimization" by Daniel Bick).

Hence we restrict the range that the current policy can vary from the old one by using the clipped surrogate objective. Because we remove the incentive for the probability ratio to move outside of the interval since the clip forces the gradient to be zero.

3.3 RL Algorithms Library

To implement the RL algorithms in this work, we used the Stable-Baselines3 [46], which utilizes the PyTorch Machine Learning (ML) framework [47] based on the Torch library. Stable-Baselines3 provides easy-to-use implementations of popular RL algorithms, including DQN and PPO, making it a suitable choice for reproducibility and rapid experimentation. This framework allowed us to focus on algorithm tuning and performance analysis while benefiting from well-optimized and standardized RL algorithm implementations.

Chapter 4

Vision Language Models (VLMs)

In recent years, we have seen impressive developments in language modeling. Many Large Language Models (LLMs) such as Llama [48] or GPT-3 [49] can now solve such a large variety of tasks that their usage is becoming increasingly popular. While these models were initially designed for text inputs, they are now being extended to include visual inputs. Bridging vision and language opens up many pivotal applications to the ongoing AI-driven technological revolution. These sophisticated models are crucial in addressing complex tasks like generating captions for images and responding to visual questions.

4.1 Language Models (LMs)

NLP stands for Natural Language Processing. It is the branch of AI that allows machines to understand and process human languages. Recent advancements in NLP primarily use neural network-based methods, also known as deep learning. It is based on neural network architecture, such as Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), and Transformers. Language Models (LMs) are specific models within NLP designed to understand, predict, and generate human language. By learning patterns in large text datasets, LMs can predict the likelihood of sequences of words and generate coherent text based on learned linguistic patterns.

4.1.1 Large Language Models (LLMs)

LLMs are scaled-up versions of traditional LMs, with billions or even trillions of parameters. This parameter increase allows LLMs to capture more nuanced patterns in language, enabling them to perform more complex tasks. In a nutshell, LLMs are designed to understand and generate text like humans and other forms of content based on the vast amount of data used to train them. To have an understanding of how LLMs work, this part introduces the basic background for LLMs.

4.1.2 Transformers

Currently, LLMs are mainly built upon the Transformer architecture [50]. Unlike previous architectures that relied on recurrence or convolutions, the Transformer uses a self-attention mechanism that allows the model to dynamically weigh the importance of different elements in the input sequence. Transformer consists of these three key components: **1) Embedding, 2) Transformer Block, 3) Output Probabilities.** The text input must first be converted into a numerical format that

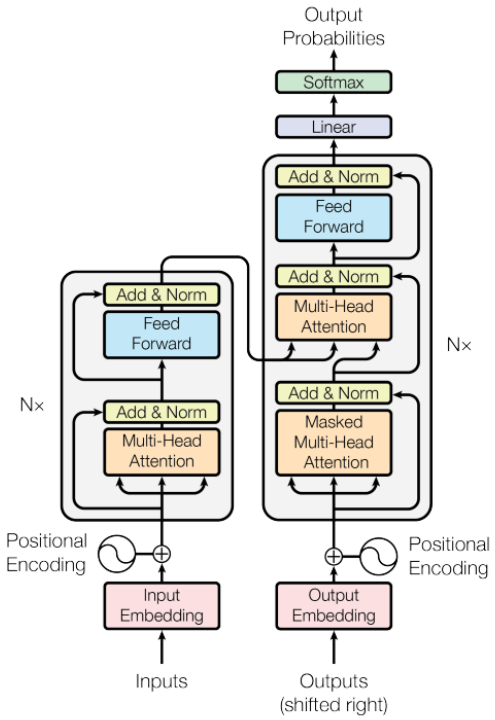


Figure 4.1: The Transformer - model architecture of encoder-decoder [50].

the model can process, where embedding comes in. Embedding transforms the text into a numerical

representation by following these steps: (1) tokenizing the input text, (2) obtaining embeddings for each token, (3) adding positional information to each token, and (4) combining token embeddings with positional encodings to produce the final embedding.

Tokenization breaks down input text into smaller, more manageable pieces called tokens. These tokens can be a word or a subword. After tokenization, each token is mapped to a unique integer ID from a predefined vocabulary. Then, the model uses an embedding matrix to map each token ID to a corresponding vector. This embedding matrix, typically denoted as W_e, is a learned matrix with dimensions vocabulary size $\times$ d_{model}. Each row in the embedding matrix represents the vector embedding for a token in the vocabulary. Since the Transformer has no inherent sequence information, positional encoding is added to the input embeddings to preserve the order of tokens. These encodings are often sinusoidal, defined as:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \tag{34}$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right) \tag{35}$$

where pos is the position, i is the dimension index, and d_{model} is the model's dimensionality. Finally, we sum the token and positional encoding to get the final embedding representation. This combined representation captures both the semantic meaning of the tokens and their position in the input sequence.

4.1.3 Encoder - Decoder Architecure

The Transformer model is structured around an encoder-decoder architecture, which is particularly effective for tasks requiring sequence-to-sequence mapping, such as translation. The encoding component is a stack of encoders, and the decoding component is a stack of decoders of the same number. In the encoder, each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise, fully connected feed-forward network (FFN). The encoder's inputs first flow through a self-attention layer. The self-attention mechanism allows each

token in the input sequence to attend to every other token, which helps the model capture dependencies and relationships within the input. In this step, each token's embedding vector is linearly mapped to three vectors: a query vector $Q = xW_q \in \mathbb{R}^{d_q}$, a key vector $K = xW_k \in \mathbb{R}^{d_k}$, and a value vector $V = xW_v \in \mathbb{R}^{d_v}$ with dimensions $d_q = d_k = d_v$. The attention mechanism then calculates the attention scores, normalizes them by $\sqrt{d_k}$, and computes the weighted sum of the value vectors:

$$\text{Attention}(Q, K, V) = \textit{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{36}$$

where dividing by $\sqrt{d_k}$ helps to keep attention scores within a reasonable range to stabilize training. Multi-head attention is an extension of the standard self-attention mechanism that uses multiple attention heads in parallel. It allows the model to focus on different aspects of token relationships

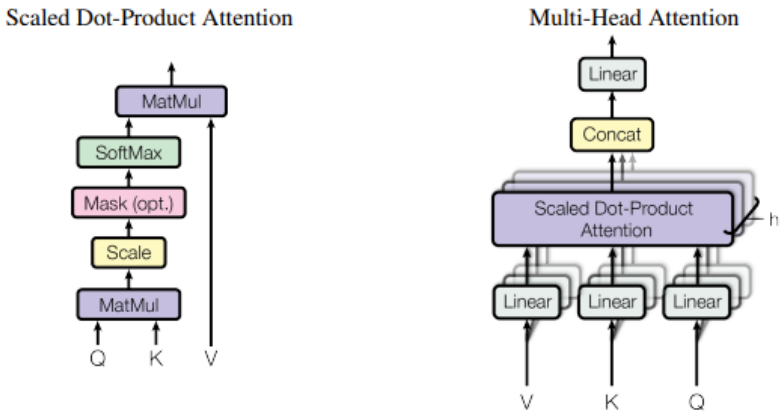


Figure 4.2: Block diagram of the multi-head attention mechanism [50].

simultaneously. Each head can learn different types of relationships between tokens, which gives the model more expressive power. Here, d_k is the dimension of the key and query vectors for each head ($d_k = d_{model}/h$). Then, the outputs from all heads are concatenated along the embedding dimension to form a single tensor:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{37}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{38}$$

$W_O \in \mathbb{R}^{d_{model} \times d_{model}}$ applies a learned transformation to the concatenated heads, which allows the model to combine information from the different heads.

For the second sub-layer (FFN), we apply two linear transformations with a ReLU activation in between. We can mathematically represent this operation as:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{39}$$

where $max(0, \text{X})$ represents the ReLU activation function, W_1 and W_2 are the weights, and b_1 and b_2 are the biases of the two linear transformations. This step introduces non-linearity so the network can learn more complex patterns. Unlike the self-attention mechanism, the FFN processes tokens independently and maps them from one representation to another.

There are also residual connections around the two sub-layers, plus layer normalization for the output of each sub-layer as shown in Fig. 4.1. In layer normalization, we compute the mean and the standard deviation of the input data. We then normalize the input data by subtracting the mean and dividing by the standard deviation plus a small number called epsilon to avoid any divisions by zero:

$$\text{LayerNorm}(x) = \gamma \frac{x - \mu}{\sigma} + \beta \tag{40}$$

where μ and σ are the mean and standard deviation of input (x), and γ, β are learnable parameters that allow the model to scale and shift the normalized values to adapt to the data. Residual connections address a common issue in deep networks. As the number of layers increases, the gradients may either vanish or explode during backpropagation. Residual connections help mitigate this by allowing gradients to flow more easily through the network. As shown in Fig. 4.1, each sub-layer, including the self-attention or feed-forward layer, has a residual connection that adds the original input of the layer to the output of that layer. Mathematically, for an input x:

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x)) \tag{41}$$

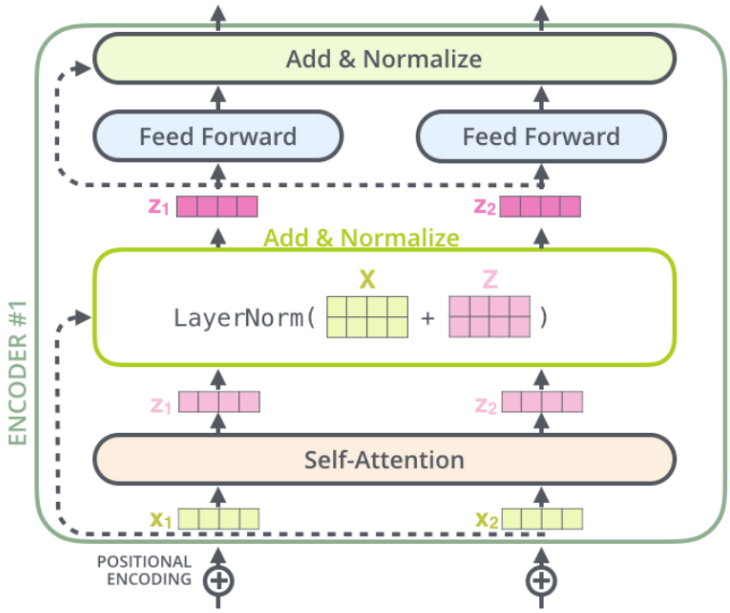


Figure 4.3: Visuliazation of layer normalization and residual connections [51].

The decoder's first sub-layer is a masked multi-head self-attention mechanism. It is similar to the encoder's multi-head self-attention but with a critical difference. It applies a mask to ensure that the model can only attend to the previously generated tokens, not future ones when predicting the next token. This masking is essential in tasks like text generation, where the model needs to predict the next token without seeing future tokens in the sequence.

$$\text{Attention}(Q, K, V) = \textit{softmax}(\frac{QK^T}{\sqrt{d_k}} + mask)V \tag{42}$$

The mask is an upper triangular matrix where each position (i, j) is set to a large negative value (e.g. $-\infty$) if $j > i$. It indicates that token i should not attend to the token j if j is in a future position. Then, any position with a large negative value due to the mask will result in a very low probability after softmax, effectively ignoring future tokens in the sequence. The second sub-layer allows the decoder to attend to the encoder's output, which provides context from the input sequence. Here, queries Q come from the previous masked self-attention layer of the decoder, but keys K and values V come from the encoder's output. Also, like the encoder, the decoder also has a feed-forward network that processes each token independently.

4.2 Computer Vision Models

Computer vision has become a vital field within AI to enable machines to process and interpret visual information. Like language models that interpret text, vision models are designed to extract meaningful features from images or videos.

4.2.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are an important step forward in terms of the scale and performance of computer vision. The CNN is a kind of feed-forward neural network that is able to extract features from data with convolution structures. They are a class of DNNs specifically designed for processing structured grid-like data, such as images. The main components of CNNs include convolutional layers, activation functions, pooling layers, and fully connected layers. CNNs can be used for various computer vision tasks, such as image processing, classification, segmentation, and object detection. The convolutional layer serves as the fundamental component of a CNN. It applies a set of learned filters (or kernels) to the input image to detect local patterns such as edges, textures, or shapes. The weights of the filters are determined during the neural network training process. Each filter is a matrix that slides over the image and performs element-wise multiplication with the input pixels to produce a feature map. The size of these filters is a hyper-parameter specified by the designers of the network architecture. By using several different filters, CNN can get a good idea of all the different patterns that make up the image. After each convolution, an activation function is applied to introduce non-linearity into the network. The most common activation function in CNNs is ReLU. There are many types of pooling layers in different CNN architectures. The purpose of pooling layers is to gradually decrease the spatial dimensions of feature maps, which retain the most important information while lowering computational complexity and improving the network's tolerance to variations in the input. Max pooling is one of the most common pooling methods that takes the maximum value within a specified window. In the final stage of a CNN, fully connected (Dense) layers and their inputs correspond to the flattened one-dimensional matrix generated by the last pooling layer. Finally, a softmax prediction layer generates probability values

for each of the possible output labels, and the final label predicted is the one with the highest probability score. Various architectures have contributed to the development of CNNs, including LetNet, AlexNet, VGGNet, and ResNet.

4.2.2 Vision Transformers (ViT)

Given that the Transformers architecture performed effectively in NLP, it was similarly implemented for images by dividing the image into small patches and considering them as tokens. While CNNs are good at extracting spatial features from images, they can struggle with capturing long-range dependencies. This limitation has led to exploring Vision Transformers (ViT), which use self-attention mechanisms to capture global context better and have shown competitive performance across various vision tasks. An overview of the model is shown in Fig. 4.4. The standard

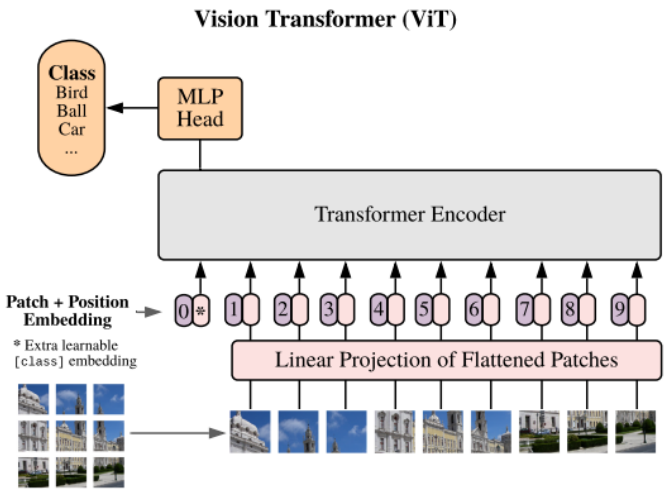


Figure 4.4: Vision Transformer Architecture [52].

Transformer receives a 1D sequence of token embeddings as input. However, input to a ViT is an image of dimension $(H \times W \times C)$, which is transformed into a sequence of flattened 2D patches $P \times P$, where (H, W) is the resolution of the original image, and C is the number of channels. So, if the image is of size $(H \times W)$, the number of patches N is:

$$N = \frac{H}{P} \times \frac{W}{P} \tag{43}$$

In the next step, each patch of size $P \times P \times C$ is then flattened into a vector of size $P^2 \times C$, which results in N patch embeddings. So, The patches can be represented as a matrix $X_{patches} \in R^{N \times (P^2 C)}$. To process these patches, ViT uses a linear projection (similar to word embeddings in NLP) to embed each patch into a higher-dimensional space. Let E be the learnable embedding matrix of shape $(P^2 \times C) \times D$ where D is the embedding dimension (e.g., 768 for ViT-B). For each patch X_i, the corresponding patch embedding Z_i is calculated as:

$$Z_i = X_i \cdot E + b \tag{44}$$

This results in a patch embedding matrix $Z \in R^{N \times D}$, where each row is a D-dimensional embedding for one patch. Then, positional encodings are added to the patch embeddings to retain spatial information about where each patch appears in the image. Then, similar to the original Transformer used in NLP, ViT uses an encoder stack that consists of multiple layers of self-attention and feed-forward neural networks. The encoder processes the sequence of image patches, and this allows the model to learn contextual relationships between different parts of the image. After passing through the encoder layers, the output from one of the patches, often referred to as the class token, is used for the final image classification. This output vector is processed by a classifier which typically is a fully connected layer, to produce the final class probabilities. If we denote the final output for the class token as Z_{class}, the classification score is given by:

$$\hat{y} = \textit{softmax}(\mathbf{z}_{\text{class}} W_{\text{cls}} + b_{\text{cls}}) \tag{45}$$

where $W_{cls} \in R^{D \times C}$, and $b_{cls} \in R^C$ are the learned parameters for the classification head, and C is the number of classes in the image classification task. Here, the term CLS stands for classification token. CLS token is added at the beginning of the input sequence. While the Transformer captures relationships and dependencies between patches through self-attention, the CLS token aggregates this information to form a final representation that is used for classification.

4.3 Bridging Vision and Language

While Transformers have revolutionized NLP tasks, and ViT has advanced image classification, there has been increasing interest in bridging the gap between these modalities by creating Vision-Language Models (VLMs). These models aim to simultaneously process visual and textual information to perform tasks that require understanding both data types. Large VLMs have good zero-shot capabilities, generalize well, and can work with many types of images, including documents, web pages, and more. The use cases include chatting about images, image recognition via instructions, visual question answering, document understanding, image captioning, etc.

4.3.1 Contrastive-based VLMs

Contrastive learning is a successful method used in VLMs. In this approach, models learn to connect images and text by grouping related pairs together and separating unrelated pairs in their representations. CLIP (Contrastive Language-Image Pretraining) [53], developed by OpenAI, is a contrastive-based model that learns image and text representations in a shared space. CLIP is trained on a large dataset of image-text pairs using a contrastive loss function. The model learns to project both the image and its corresponding caption into a shared embedding space. During training, CLIP maximizes the similarity between the embeddings of an image and its corresponding text while minimizing the similarity between embeddings of mismatched pairs. The CLIP's training process is summarized in Fig. 4.5. The fundamental mathematical concept in CLIP is the contrastive loss function, which ensures that matching image-text pairs are pulled together in the shared embedding space while non-matching pairs are pushed apart, and this loss function can be described using the cosine similarity between the image and text embeddings. The first step is to extract the feature representations of the image and text using their respective encoders. Given an image I and a corresponding text T, CLIP computes the image and text embeddings using separate encoders for each modality (using Vision Transformers or ResNet for images and Transformers for text). Then, these embeddings are normalized to unit length

$$\hat{f}_I(I) = \frac{f_I(I)}{\|f_I(I)\|}, \quad \hat{f}_T(T) = \frac{f_T(T)}{\|f_T(T)\|} \tag{46}$$

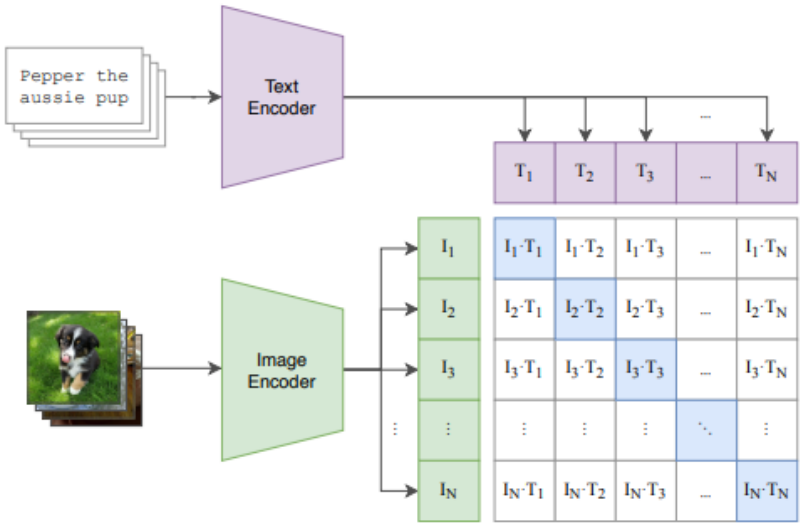


Figure 4.5: The process of training CLIP [53].

The cosine similarity between the image and embeddings is computed as:

$$\hat{f}_I(I) \cdot \hat{f}_T(T) \tag{47}$$

where $\hat{f}_I(I)$ and $\hat{f}_T(T)$ are the normalized image and text embeddings. The contrastive loss is designed to bring the positive (correct image-text) pair closer and push other (negative) pairs apart. The loss is typically calculated using the cross-entropy between the similarities of the image-text pairs:

$$\mathcal{L}_{\text{image-to-text}} = -\log \frac{\exp(\hat{f}_I(I) \cdot \hat{f}_T(T))}{\sum_{j=1}^{N} \exp(\hat{f}_I(I) \cdot \hat{f}_T(T_j))} \tag{48}$$

$$\mathcal{L}_{\text{text-to-image}} = -\log \frac{\exp(\hat{f}_T(T) \cdot \hat{f}_I(I))}{\sum_{i=1}^{N} \exp(\hat{f}_T(T) \cdot \hat{f}_I(I_i))} \tag{49}$$

The denominator in Eq. 48 sums the similarities between the image and all other text embeddings, and in Eq. 49, sums the similarities between the text and all other image embeddings in the batch. Finally, The total loss in CLIP is the average of the image-to-text and text-to-image losses:

$$\mathcal{L}_{\text{total}} = \frac{1}{2} \left(\mathcal{L}_{\text{image-to-text}} + \mathcal{L}_{\text{text-to-image}} \right) \tag{50}$$

By averaging the two losses, the total loss balances the contributions from both directions, and ensures that the image and text embeddings are aligned in the shared space.

4.4 Training & Dataset

A VLM, specifically CLIP, was employed to enhance the autonomous vehicle's decision-making process in various intersection scenarios. As mentioned, CLIP enables the model to associate visual inputs with natural language instructions, which can be used to guide the agent's behavior based on contextual understanding. The approach leverages transfer learning, which is particularly beneficial given the relatively small dataset available for fine-tuning. Transfer learning allows us to utilize the features learned by CLIP, which have been pre-trained on large, diverse datasets. Transferring these learned features to our dataset allows the model to improve performance, even with limited training data. This process avoids updating the entire model; most of the model's weights are frozen, and only specific layers are fine-tuned, which causes faster training times and reduces GPU usage. To do that, a dataset including 500 images from different frames of the intersection scenarios was collected. These images represented the diverse situations the AV could encounter during training. Alongside the images, 500 corresponding instructions were defined as single sentence text prompts, serving as labels for each image. These instructions are based on the vehicle's current visual input. Fig. 4.7 illustrates the dataset's structure which is used to fine-tune the CLIP model.

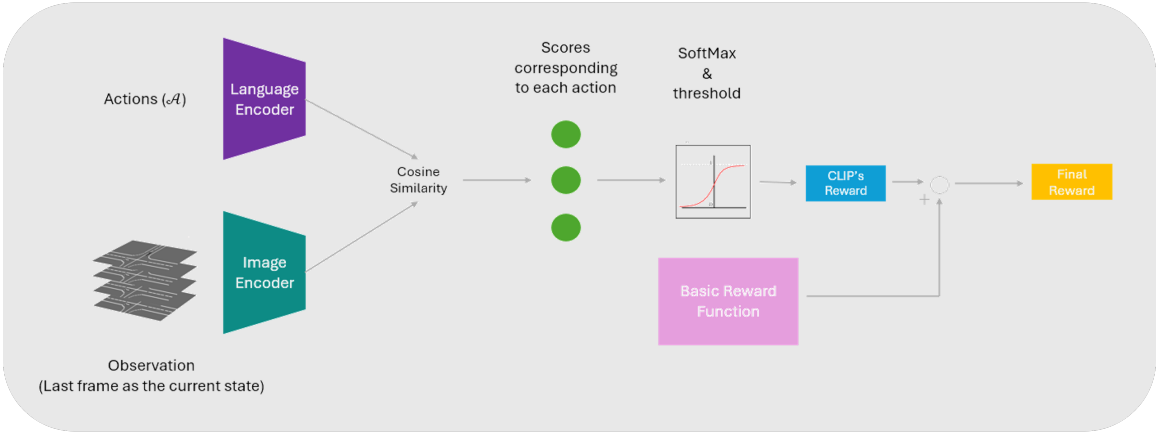


Figure 4.6: Architecture for CLIP as a reward model.

The CLIP model was fine-tuned on this dataset, allowing it to learn the relationship between the visual scenes and the appropriate instructions. Once fine-tuned, the trained model was incorporated into the AV's decision-making system. So, basically, we use CLIP as a reward model to train AV in an unsignalized intersection scenario. In each training step, the current environment state of the AV, which is represented by its visual input, is passed through the vision encoder of the CLIP model. Simultaneously, the three possible actions in the environment $\mathcal{A}_i$ = [`Drive Slower, Maintain Speed (IDLE), Drive Faster`] are passed through the text encoder. These actions are encoded as text embeddings corresponding to each potential behavior the AV might adopt. By leveraging the fine-tuned CLIP model, the agent receives rewards from CLIP's reward model whenever its action aligns with the instruction output by CLIP for the current scenario. This allows the agent to align its behavior with the predefined instructions and helps it make decisions that are consistent with the desired driving behaviors at the unsignalized intersection. To ensure that the basic reward structure from the simulation environment does not dominate the training process, we multiply CLIP's reward by a weight determined during fine-tuning. This scaling factor allows us to balance the influence of CLIP's reward with the basic reward from the simulation. Finally, the weighted reward from CLIP is then combined with the reward from the basic reward structure provided by the simulation environment:

$$R_{final} = r_b + w_c \times r_c \tag{51}$$

Where r_b is the basic reward from the environment, w_c is the corresponding weight, and r_c is the reward by CLIP reward model. Fig. 4.6 shows how the reward mechanism using CLIP as a reward model works.

In this work, we employed DQN and PPO as DRL methods while incorporating CLIP's reward model to guide the agent in optimizing its behavior in a human-like decision-making manner, at an unsignalized four-way intersection. We compare these two method's performance with and without using CLIP and discuss the results regarding the safety and efficiency of the proposed framework. Algorithm 4 provides more detailed steps via pseudo-code for the DQN implementation. Our experiments utilized the *Highway-env* simulation environment, specifically designed to mimic real-world

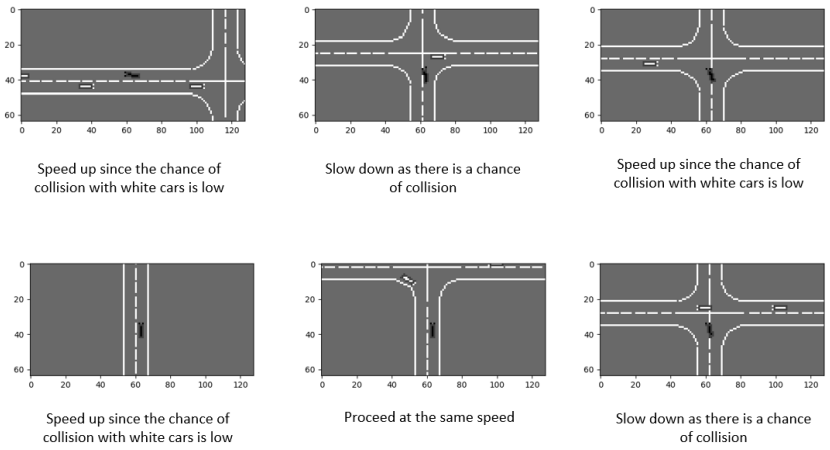


Figure 4.7: A summary of the (scenario, instruction) pairs in various driving situations at an unsignalized intersection, used for fine-tuning the CLIP model.

traffic scenarios. The simulation was conducted with a frequency of 15 Hz to ensure a realistic representation of dynamic traffic patterns. The computational experiments used an NVIDIA GeForce RTX 3050 Ti GPU and an AMD Ryzen 7-5800H CPU @3.20 GHz. The effectiveness of the PPO and DQN algorithms was influenced by the tuning of hyperparameters, which is demonstrated in Table 4.1. Different image encoder architectures are available for CLIP. In this work, we experiment

Hyperparameter	**DQN**	**PPO**
N_{steps}	8000	8000
Learning rate (α)	0.0005	0.0005
Discount factor (γ)	0.95	0.99
Replay buffer size	15000	-
Batch size	32	64
Epsilon decay	Linear	-
Epochs	-	10
Clip range	-	0.2
λ	-	0.95

Table 4.1: Hyperparameters used in DQN and PPO algorithms.

with the ViT. Specifically, we fine-tune the ViT-B/32 model on our dataset for 15 epochs using the Adam optimizer [54]. For the purpose of transfer learning, we updated the gradients only for the last layer of the model and left the earlier layers frozen. The hyperparameters used in this setup are listed in Table 4.2.

Algorithm 4 Pseudo-code of the proposed CLIP-Based reward function with DQN

Initialize the Q-network $Q(s, a; \theta)$ with random weights θ
Initialize the target Q-network $\hat{Q}(s, a; \theta^-)$ with $\theta^- \leftarrow \theta$
Initialize the replay buffer B with capacity N
Set hyperparameters: learning rate α, discount factor γ, epsilon ϵ, batch size N_b
for each episode **do**
    Initialize state s_0
    for each step in the episode **do**
        With probability ϵ, select a random action a_t, otherwise select:

$$a_t = \arg\max_a Q(s_t, a; \theta)$$

        Execute action a_t and observe reward r_b and next state s_{t+1}
        Pass the visual observation to the CLIP model to get the instruction
        Comparing the taken action with suggested action based on CLIP
        if a_t = CLIP instruction **then**

$$r_f = r_b + w \times r_c$$

        else
            Use only the basic reward $r_f = r_b$
        end if
        Store the transition (s_t, a_t, r_f, s_{t+1}) in the replay buffer B
        Sample a random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from the buffer B
        Set the target for each transition:

$$y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$$

        Perform a gradient descent step on:

$$L(\theta) = \frac{1}{N_b} \sum_{j=1}^{N_b} \left(Q(s_j, a_j; \theta) - y_j\right)^2$$

        Update the target network: $\hat{Q}(s, a; \theta^-) \leftarrow Q(s, a; \theta)$ every C steps
    end for
end for

	Learning	Embedding	Input	Vision Transformer			Text Transformer		
Model	rate	dimension	resolution	layers	width	heads	layers	width	heads
ViT-B/32	5×10^{-4}	512	224	12	768	12	12	512	8

Table 4.2: CLIP ViT-B/32 hyperparameters.

Chapter 5

Simulation Experiment Results

In this chapter, the performance of the proposed framework is evaluated using DQN and PPO as RL algorithms and CLIP as a VLM to build a reward model. An overview of the experimental setup is also provided, including the environment configurations, hyperparameters, and training protocols used for DQN and PPO. Subsequently, the results are presented based on several evaluation metrics to compare the performance of the two RL algorithms with and without the CLIP-based reward function. In addition, the influence of CLIP on agent behavior at unsignalized intersections will be evaluated based on key factors such as collision rate, success rate, and alignment with human-like decision-making behaviors through a series of simulation experiments.

5.1 Training Evaluation

Training sessions on environmental configuration were conducted, as presented in the Table. 5.1. The simulations were conducted for 8,000 steps to train the proposed models using DQN and PPO algorithms separately. The aim was to observe and analyze the behavior of the AVs in an unsignalized four-way intersection. Several key factors were measured, including the success rate, collision rate, and timeout rate (instances when an autonomous vehicle fails to reach its destination). The primary objective of these simulations was to assess the overall generalization ability of the trained policies with CLIP used as a reward model.

During the experiments, each episode's average reward and duration were recorded and assessed

Table 5.1: Environment configuration for intersection-v1.

Configuration	**Value**
Observation Type	Grayscale
Observation Shape	$(128, 64)$
Stack Size	4
Action Type	Discrete
Duration (seconds)	30
Initial Vehicle Count	5
Spawn Probability	0.2
Simulation Frequency	15

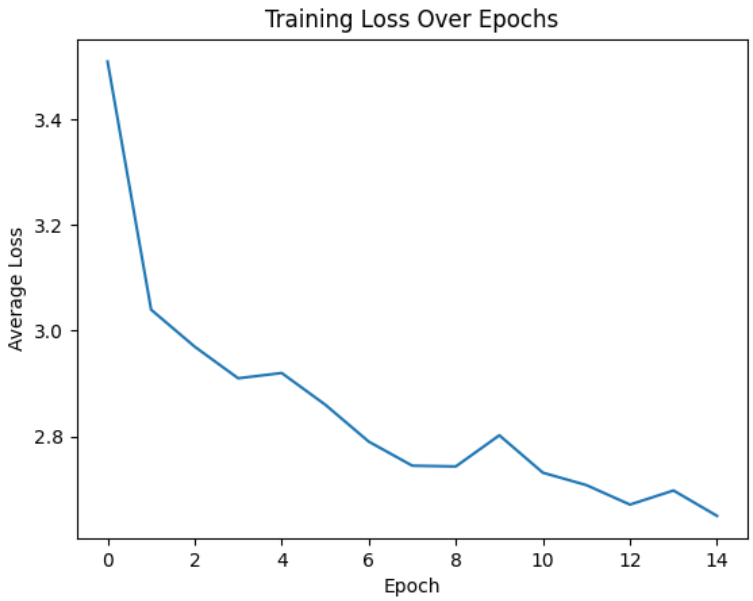
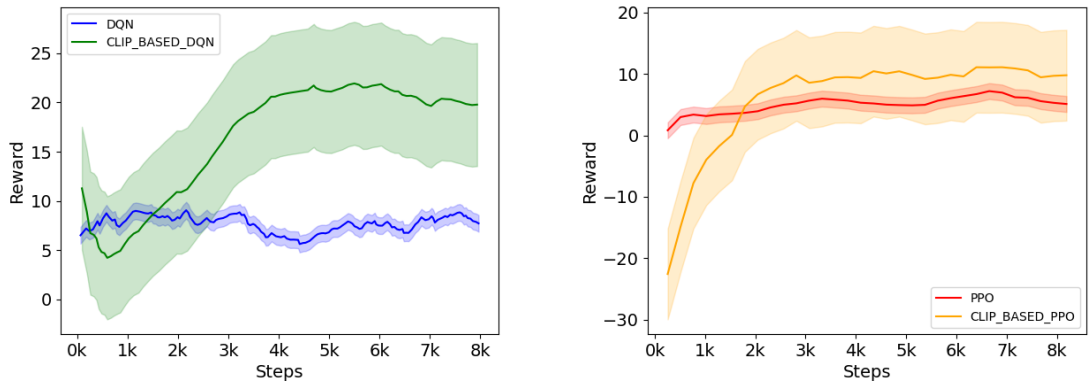


Figure 5.1: Training loss in the process of fine-tuning CLIP.

for the agent in order to evaluate the performance of the training session under each method. Both the episodic reward and the episode's length were evaluated as seen in Fig. 5.2, and 5.3, respectively. This indicates that integrating the CLIP-based reward model enhances the performance of both DQN and PPO algorithms considerably. The CLIP-based DQN shows better performance than CLIP-based PPO, which also indicates that rewards increase steadily over time. In contrast, both DQN and PPO without CLIP show limited learning progress, as their reward variations do not show a meaningful increase. This indicates that the agent struggles to optimize their behavior effectively in such a complex driving scenario. While CLIP-based PPO also performs better than standard PPO,

the improvement is more pronounced in CLIP-based DQN, which highlights the greater impact of the CLIP reward model when applied to DQN. Based on Fig. 5.3, a lower episode length suggests that the agent may have encountered a collision with other vehicles, which caused the episode to terminate prematurely. On the other hand, since the episode duration is set to 30 seconds, a high episode length indicates that the agent failed to complete the left-turn task within the allocated time. In contrast, for the CLIP-based DQN, the episode length gradually converges to an optimal value over the training process. This suggests that the agent learns to brake upon arrival at the intersection to avoid collision and successfully completes its task of turning left. Furthermore, Fig. 5.4 shows the epsilon decay variations to indicate the agent's transition from exploration to exploitation during training with DQN algorithm. During the training process, epsilon decreases, which means the agent shifts towards exploiting its learned policy rather than exploring new actions. Also, 5.1 shows the training loss variations and the progress of fine-tuning the CLIP model on the custom dataset over 15 epochs.

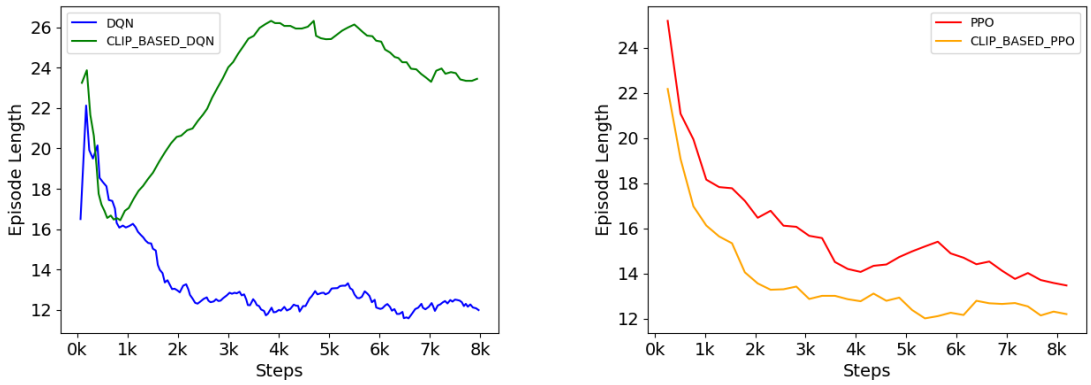


(a) DQN (b) PPO

Figure 5.2: The change of average reward per episode through the training process for CLIP-based and non-CLIP-based DQN, and PPO.

5.2 Post-training Evaluation

The post-training evaluation involves deploying the trained agent utilizing CLIP-based DQN, CLIP-based PPO, vanilla DQN, and PPO in an intersection setting to assess its navigation abilities



(a) DQN (b) PPO

Figure 5.3: The change of episode's length through the training process for CLIP-based and non-CLIP-based DQN, and PPO.

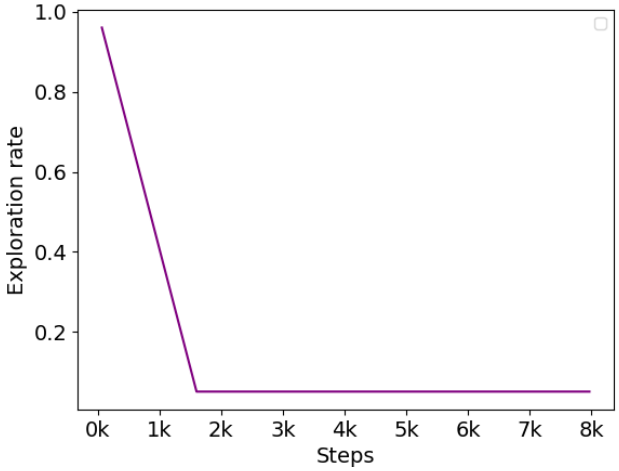


Figure 5.4: Exploration rate decay over training steps.

across 100 testing episodes. The trained agent employs its learned policy to make real-time decisions at every time step. The agent starts in an initial state and chooses the best action based on its policy. Following this, it receives visual observation and reward and transitions to the next state. This iterative process goes on until the agent arrives at a terminal state during the episode. Algorithm 5 shows how the required steps to do the evaluation process. The process begins by resetting the environment to obtain the initial state, and the trained agent selects an action a_t according to its policy $\pi_{\mathcal{A}}$. The agent executes this action, receiving a reward r_t and observing the next state. The evaluation continues until the episode terminates, either due to a collision, successful arrival

at the destination, or truncation. If a collision occurs during the episode, the collision counter c is incremented by one, and the episode is terminated. In contrast, if the agent successfully reaches its destination, the success counter s is incremented by one. During the evaluation, the agent's speed at each time step is appended to a list v. This allows for calculating the average speed $\bar{v}$ at the end of the evaluation process. At the end of all episodes, the algorithm outputs the normalized metrics. This approach ensures a comprehensive assessment of the trained agent's behavior across multiple scenarios.

Algorithm 5 Post-Training Evaluation

Input: Trained agent ($\mathcal{A}$), number of episodes (E)
Output: $\bar{c} = c/E$, $\bar{s} = s/E$, $\bar{v} = v/E$

Initialize c, s, v
for $e = 0 : E$ **do**
    done $\leftarrow$ **False**
    truncated $\leftarrow$ **False**
    $s_0 \leftarrow$ reset the environment
    $a_0 \sim \pi_{\mathcal{A}}(s_0)$
    while not *(done or truncated)* **do**
        $t \leftarrow t + 1$
        $s_{t+1}, r_t,$ *done* $\leftarrow$ agent $\mathcal{A}$ takes an action a_t
        $v \leftarrow$ append the agent's speed
        if *collision* **then**
            $c \leftarrow c + 1$
        end if
        if *arrived* **then**
            $s \leftarrow s + 1$
        end if
    end while
end for

Table 5.2 summarizes the success, collision, and time-out rates during the testing process with different trained policies. We extend our analysis to scenarios involving 1, 3, and 6 HVs to examine the impact of vehicle count on the performance of CLIP-DQN, CLIP-PPO, DQN, and PPO. We observe that the success rate, collision rate, and time-out rate for CLIP-DQN in the case with 1 on-road vehicle are 96%, 4%, and 0%, respectively. These values are 38%, 8%, and 54% for CLIP-PPO, 70%, 30%, and 0% for DQN, and 78%, 22%, and 0% for PPO. Similarly, in scenarios with 3 and 6 HVs, CLIP-DQN achieves the best success and collision rates, outperforming the other methods. However, as the number of on-road vehicles increases, the success rate decreases, and the collision rate rises due to the increasing complexity of the environment. For CLIP-DQN, the

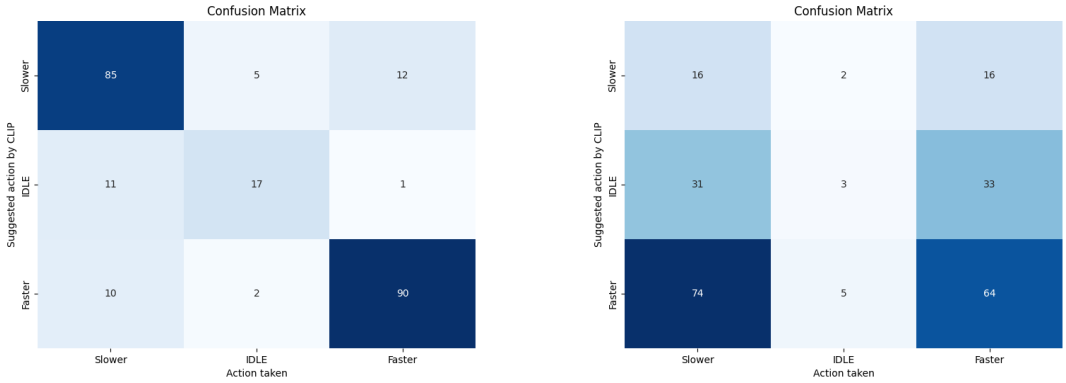
success rate decreases from 96% to 84% and then to 72%, while the collision rate increases from 4% to 16% and then to 28%, respectively. One notable drawback of CLIP-PPO is its non-smooth movement, often preventing the agent from completing the turning task within the specified time. This limitation is reflected in its significantly higher time-out rate compared to other methods.

Method	**Vehicles**	**Success (%)**	**Collision (%)**	**Time-out (%)**
CLIP-DQN	1	**96**	**4**	0
	3	**84**	**16**	0
	6	**72**	**28**	0
CLIP-PPO	1	38	8	54
	3	38	20	42
	6	26	26	48
DQN	1	70	30	0
	3	56	44	0
	6	50	50	0
PPO	1	78	22	0
	3	60	40	0
	6	56	44	0

Table 5.2: Comparison of performance for different methods across scenarios with varying initial numbers of on-road vehicles.

Confusion matrices in Fig. 5.5 illustrate the alignment between the actions taken by the trained agents using CLIP-based DQN and PPO and the actions suggested by the CLIP model. In these matrices, the rows represent the actions suggested by CLIP, and the columns represent the actions actually taken by the agent. For the agent trained with CLIP-based DQN, the actions taken are generally well-aligned with the CLIP recommendations. Most of the time, the agent follows the suggested actions, which shows the satisfactory coordination between the model and the agent's decisions. In contrast, for the agent trained using CLIP-based PPO, there is a significant misalignment between the suggested and actual actions, particularly when it comes to slowing down. This misalignment is most evident in the agent's failure to decelerate when approaching the intersection, which results in higher collision rates.

The histogram of rewards in Fig. 5.6 illustrates the distribution of rewards obtained by the AV during the evaluation process. The plot clearly shows that the overall reward achieved by both CLIP-based DQN and PPO is higher than that of the vanilla methods. When comparing CLIP-based



(a) CLIP-based DQN (b) CLIP-based PPO

Figure 5.5: Confusion matrices comparing the actions suggested by the CLIP model and those taken by the agent.

DQN and CLIP-based PPO, the reward gained by the DQN agent is noticeably higher than that of the PPO agent, further confirming the superior performance of the CLIP-based DQN algorithm in this scenario. The speed profile plot in Fig. 5.7 reflects the behavior of the AV over time during the

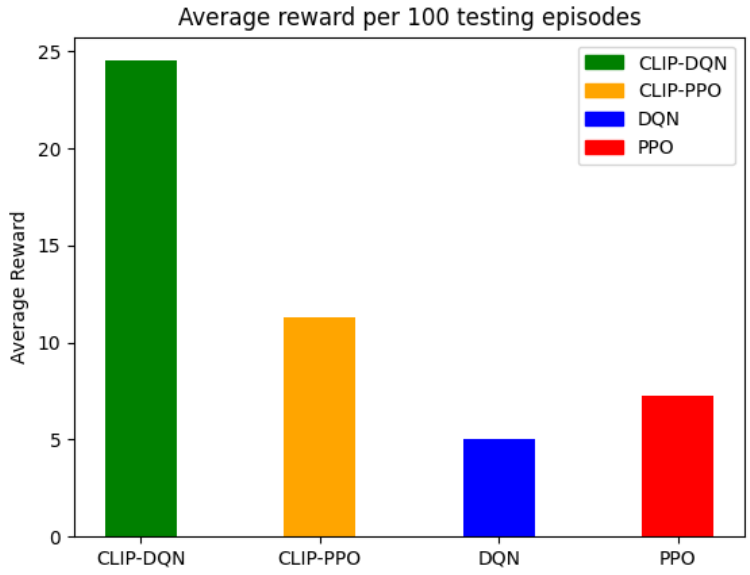


Figure 5.6: The histogram of rewards shows the distribution of rewards that the AV obtained during the evaluation process with different methods.

test. With CLIP-based DQN, the speed profile follows a logical pattern. As the AV approaches the

intersection, its speed to brake reduces. Once it's clear to make the left turn, the AV accelerates to complete the maneuver. This gradual shift between slowing down and speeding up reflects effective decision-making in response to the environment. In contrast, the speed profile for CLIP-based PPO shows significant fluctuations. The AV does not exhibit smooth driving and appears to struggle with maintaining appropriate speed during critical phases, particularly when approaching the intersection. These fluctuations suggest that the CLIP-based PPO agent is less consistent in following the desired behavior, which leads to less optimal and potentially unsafe driving actions.

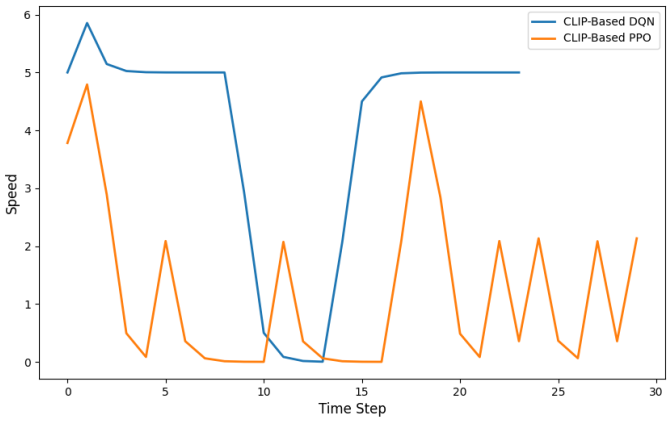


Figure 5.7: Ego vehicle speed's profile over time.

Figure 5.8 shows the frequency distribution of actions taken by the AV over multiple test episodes. The "Faster" action was the most frequently selected, which also means that the vehicle often prioritized increasing its speed. Conversely, the "Slow Down" action was selected less frequently, and "IDLE" had the lowest frequency. In this scenario, one of our key objectives is to complete the task as quickly as possible to improve traffic efficiency by reducing waiting times for other vehicles. At the same time, the policy is designed to help improve safety by avoiding collisions. The higher frequency of the "Faster" action means that the trained policy has effectively learned to balance these objectives while minimizing unnecessary deceleration. Given this objective, the "IDLE" action is less critical compared to other actions. The plot in Fig. 5.9 reflects the Q-values corresponding to each action. At the beginning of the episode, the Q-values for the "Faster" action are higher than those for "Slow Down," indicating that the AV prioritizes increasing

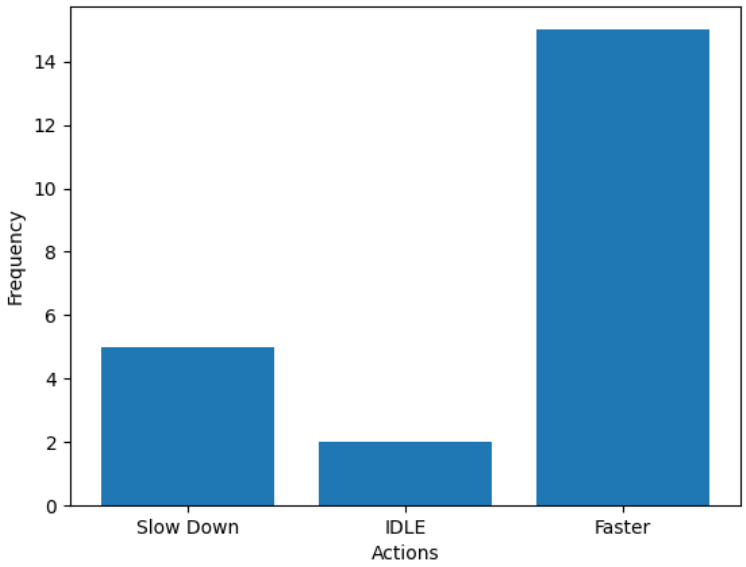


Figure 5.8: Frequency of each action with CLIP-based DQN

its speed during the initial stages. As the AV approaches the intersection (highlighted by the vertical dashed lines), the Q-values for "Slow Down" increase, which shows that the policy identifies the need to decelerate to navigate the intersection safely.

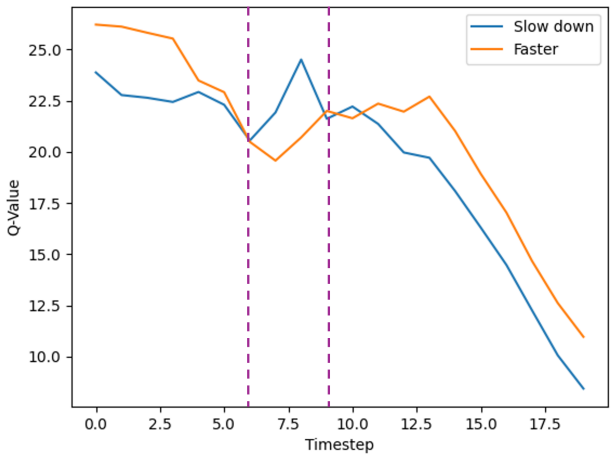


Figure 5.9: Q-Values from CLIP-based DQN, for each action over time.

This also indicates that the policy appropriately values caution in potentially risky scenarios, which also abides by the safety objectives of the task. After passing the intersection, the Q-values for

”Faster” rise again, which indicates a shift back to prioritizing speed for efficient task completion.

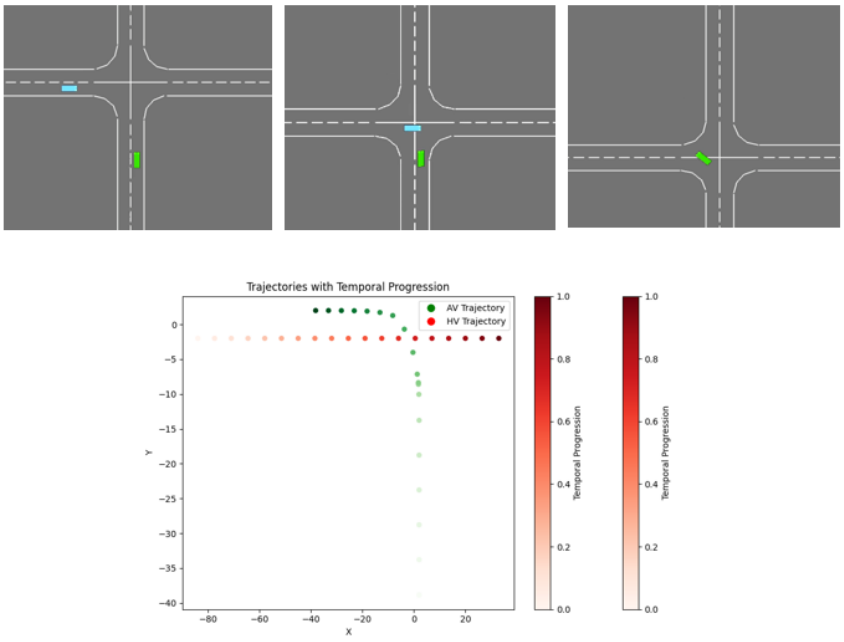


Figure 5.10: Scenario 1, the AV yields to the HV approaching from the west.

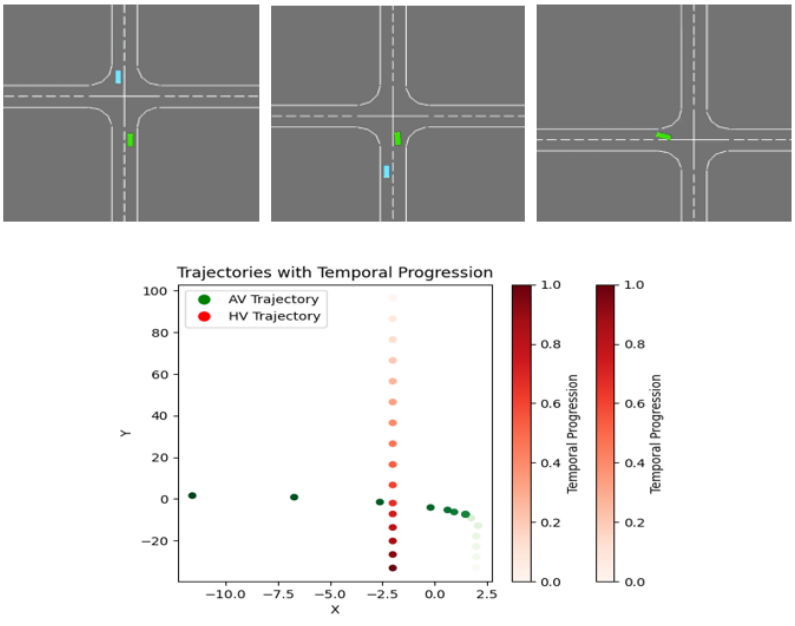


Figure 5.11: Scenario 2, the AV yields to the HV approaching from the north.

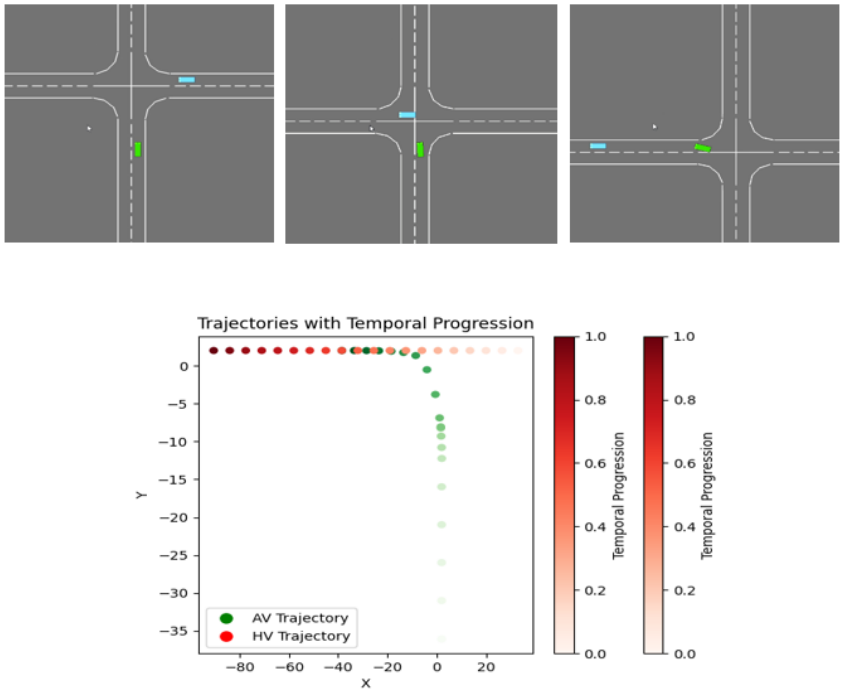
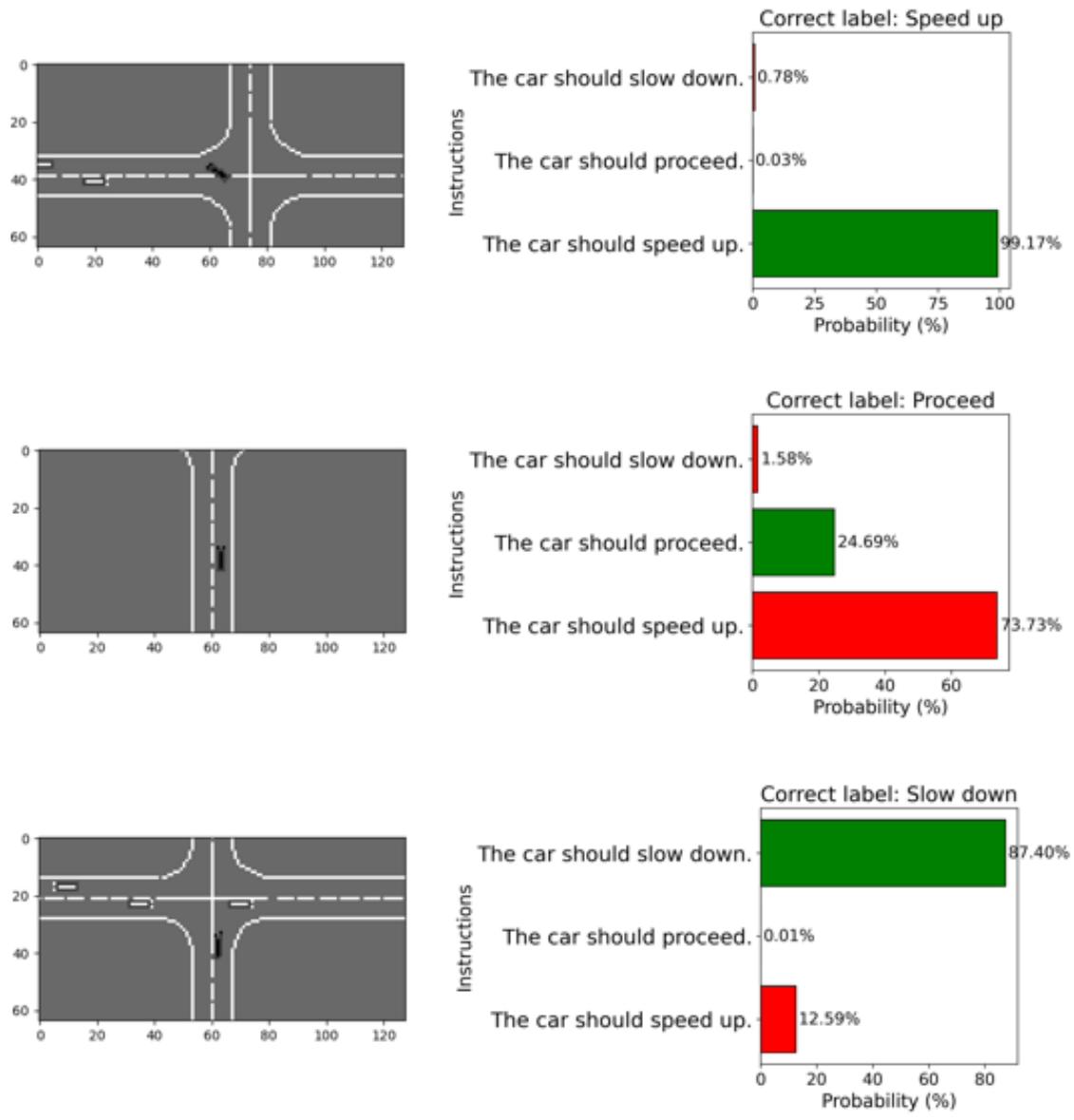


Figure 5.12: Scenario 3, the AV yields to the HV approaching from the east.

Figures 5.10, 5.11, and 5.12 display the trajectories of the AV and HVs in three intersection scenarios using an agent trained with the CLIP-based DQN. In each scenario, the AV starts from the same initial position (south), while HVs approach the intersection from different directions. These figures show the AV's decision-making process and emphasize its altruistic behavior. Also, the bottom plot in each figure presents the temporal progression of the trajectories. The intensity of the color gradient shows the temporal progress of the vehicles. In Fig. 5.10, the AV begins its trajectory from the south side of the intersection, and an HV approaches from the east. The AV slows down to allow the HV to pass safely through the intersection. Once the interestection is clear, the AV accelerates and executes a left turn. In Fig. 5.11, and 5.12, the HV starts from north, and west, respectively. Similar to Scenario 1, the AV yields to the HV to maintain a cautious approach, and then it speeds up and completes the left turn.

Figure 5.13 illustrates the performance of the CLIP-based model in predicting driving instructions for some random intersection scenarios. For each scenario, the green bar represents the probability of the correct label, while the red bars indicate the probabilities of the incorrect labels. The

results demonstrate the model's ability to align the intersection scene with the appropriate driving action.



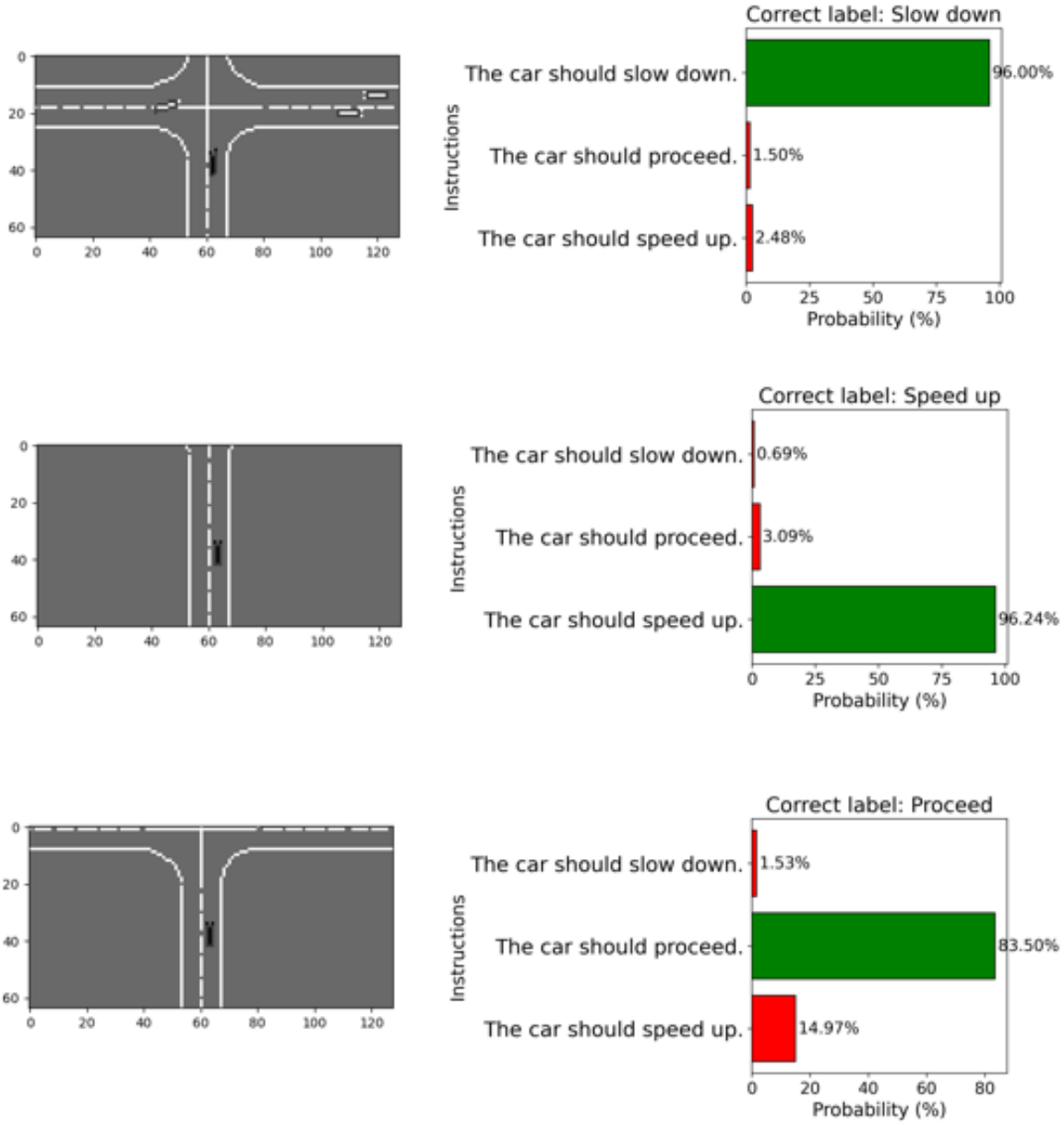


Figure 5.13: CLIP-based predictions for driving instructions at different intersection scenarios. Each row corresponds to a distinct intersection situation, with the left panel displaying the intersection scene and the right panel showing the probabilities of three possible actions. The ground truth label is colored green while an incorrect prediction is colored red. The probabilities are computed based on the similarity between the intersection scene and the textual instructions using CLIP.

Chapter 6

Conclusions and Future Research

We proposed a novel framework that integrates VLMs, specifically CLIP, with RL algorithms such as PPO and DQN to improve the decision-making capabilities of autonomous vehicles navigating through unsignalized four-way intersections, which is a complex environment to find the optimal policy with traditional approaches. Our research has successfully developed and demonstrated that by introducing a reward mechanism based on VLMs, we enable the AV to align its actions more closely with human-like decision-making patterns to address the limitations of rule-based systems. Our experiments demonstrated that the agent achieved higher rewards when guided by this enhanced reward function, compared to using only the conventional reward function. This work highlights the potential of combining VLMs with RL techniques to develop more adaptive and context-aware AV systems.

Despite the promising outcomes, our study is not without its limitations. Fundamentally, our approach relies on the reward model, generalizing from a text description corresponding to each potential driving scenario in the simulation environment, to capture what a human does in a similar case. However, the proposed approach was evaluated in a simplified simulation environment with a limited dataset. It is proposed to extend this work by using high-fidelity simulation environments such as CARLA or CarSim, which provide richer environmental features and more realistic traffic behaviors. Additionally, we anticipate that future advancements in VLMs will enhance their ability to generalize across more complex and nuanced decision-making tasks. We expect their integration into AV systems to enable more sophisticated and adaptable behavior, even in scenarios that are

challenging to describe in a simple and short language prompt.

To overcome these limitations, it is proposed to expand the dataset to include a wider variety of intersection scenarios and edge cases, as these are essential for improving the generalization of the proposed approach. Furthermore, a human-in-the-loop framework could be developed to enhance the realism of the evaluation process. By incorporating virtual reality environments and real-time human interaction, subjective human factor analysis could be performed to understand better how humans perceive and respond to AV behavior. This approach would allow for the fine-tuning of algorithms based on direct feedback for other practical applications.

Bibliography

[1] W. H. Organization et al. *Global Status Report on Road Safety 2018: Summary*. Tech. Rep. Accessed: 2024-10-17. World Health Organization, 2018. URL: https://www.who.int/publications/i/item/9789241565684.

[2] SAE. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles J3016 202104*. Tech. rep. SAE International, 2021.

[3] Jingyuan Zhao et al. "Autonomous driving system: A comprehensive survey". In: *Expert Systems with Applications* (2023), p. 122836.

[4] B Ravi Kiran et al. "Deep reinforcement learning for autonomous driving: A survey". In: *IEEE Transactions on Intelligent Transportation Systems* 23.6 (2021), pp. 4909–4926.

[5] Badr Ben Elallid et al. "A comprehensive survey on the application of deep and reinforcement learning approaches in autonomous driving". In: *Journal of King Saud University-Computer and Information Sciences* 34.9 (2022), pp. 7366–7390.

[6] Ammar Haydari and Yasin Yılmaz. "Deep reinforcement learning for intelligent transportation systems: A survey". In: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2020), pp. 11–32.

[7] Prasoon Goyal, Scott Niekum, and Raymond J Mooney. "Using natural language for reward shaping in reinforcement learning". In: *arXiv preprint arXiv:1903.02020* (2019).

[8] David Isele et al. "Navigating occluded intersections with autonomous vehicles using deep reinforcement learning". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 2034–2039.

[9] Michiel M Minderhoud and Piet HL Bovy. “Extended time-to-collision measures for road traffic safety assessment”. In: *Accident Analysis & Prevention* 33.1 (2001), pp. 89–97.

[10] Niranjan Deshpande and Anne Spalanzani. “Deep reinforcement learning based vehicle navigation amongst pedestrians using a grid-based state representation”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019, pp. 2081–2086.

[11] Max Peter Ronecker and Yuan Zhu. “Deep Q-network based decision making for autonomous driving”. In: *2019 3rd international conference on robotics and automation sciences (ICRAS)*. IEEE. 2019, pp. 154–160.

[12] Arun Nair et al. “Massively parallel methods for deep reinforcement learning”. In: *arXiv preprint arXiv:1507.04296* (2015).

[13] Luis Miguel Rios and Nikolaos V Sahinidis. “Derivative-free optimization: a review of algorithms and comparison of software implementations”. In: *Journal of Global Optimization* 56.3 (2013), pp. 1247–1293.

[14] Volodymyr Mnih. “Asynchronous Methods for Deep Reinforcement Learning”. In: *arXiv preprint arXiv:1602.01783* (2016).

[15] John Schulman. “Trust Region Policy Optimization”. In: *arXiv preprint arXiv:1502.05477* (2015).

[16] John Schulman et al. “PPO algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[17] Andreas Folkers, Matthias Rick, and Christof Büskens. “Controlling an autonomous vehicle with deep reinforcement learning”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 2025–2031.

[18] Jianyu Chen, Zining Wang, and Masayoshi Tomizuka. “Deep hierarchical reinforcement learning for autonomous driving with distinct behaviors”. In: *2018 IEEE intelligent vehicles symposium (IV)*. IEEE. 2018, pp. 1239–1244.

[19] Xiaolin Tang et al. “Highway decision-making and motion planning for autonomous driving via soft actor-critic”. In: *IEEE Transactions on Vehicular Technology* 71.5 (2022), pp. 4706–4717.

[20] Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2004.

[21] Nathan Lambert et al. "Learning accurate long-term dynamics for model-based reinforcement learning". In: *2021 60th IEEE Conference on decision and control (CDC)*. IEEE. 2021, pp. 2880–2887.

[22] Kavosh Asadi et al. "Combating the compounding-error problem with a multi-step model". In: *arXiv preprint arXiv:1905.13320* (2019).

[23] Nathan Lambert et al. "Objective mismatch in model-based reinforcement learning". In: *arXiv preprint arXiv:2002.04523* (2020).

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[25] Volodymyr Mnih. "Playing atari with DRL". In: *arXiv preprint arXiv:1312.5602* (2013).

[26] Tom Schaul. "Prioritized Experience Replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[27] Saijiang Shi and Feng Chen. "Deep recurrent Q-learning method for area traffic coordination control". In: *Journal of Advances in Mathematics and Computer Science* 27.3 (2018), pp. 1–11.

[28] Jingda Wu et al. "Toward human-in-the-loop AI: Enhancing deep reinforcement learning via real-time human guidance for autonomous driving". In: *Engineering* 21 (2023), pp. 75–91.

[29] Jette Randløv and Preben Alstrøm. "Learning to Drive a Bicycle Using Reinforcement Learning and Shaping." In: *ICML*. Vol. 98. 1998, pp. 463–471.

[30] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *Icml*. Vol. 99. 1999, pp. 278–287.

[31] Pieter Abbeel and Andrew Y Ng. *Inverse Reinforcement Learning*. 2010.

[32] Paul F Christiano et al. "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.

[33] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *Advances in neural information processing systems* 35 (2022), pp. 27730–27744.

[34] Yuntao Bai et al. “Constitutional ai: Harmlessness from ai feedback”. In: *arXiv preprint arXiv:2212.08073* (2022).

[35] Jérémy Scheurer et al. “Training language models with language feedback at scale”. In: *arXiv preprint arXiv:2303.16755* (2023).

[36] Juan Rocamonde et al. “Vision-language models are zero-shot reward models for reinforcement learning”. In: *arXiv preprint arXiv:2310.12921* (2023).

[37] Tianbao Xie et al. “Text2reward: Automated dense reward function generation for reinforcement learning”. In: *arXiv preprint arXiv:2309.11489* (2023).

[38] Theodore Sumers et al. “Distilling internet-scale vision-language models into embodied agents”. In: *arXiv preprint arXiv:2301.12507* (2023).

[39] Veljko Dubljevic et al. “Toward a rational and ethical sociotechnical system of autonomous vehicles: A novel application of multi-criteria decision analysis”. In: *Plos one* 16.8 (2021), e0256224.

[40] Yaodong Cui et al. “Drivellm: Charting the path toward full autonomous driving with large language models”. In: *IEEE Transactions on Intelligent Vehicles* (2023).

[41] Edouard Leurent. *An Environment for Autonomous Driving Decision-Making*. https://github.com/eleurent/highway-env. 2018.

[42] G Brockman. “OpenAI Gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[43] Philip Polack et al. “The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?” In: *2017 IEEE intelligent vehicles symposium (IV)*. IEEE. 2017, pp. 812–818.

[44] TutorialsPoint. *Breadth-First Traversal - Data Structures and Algorithms*. Accessed: 2024-11-28. Accessed 2024. URL: https://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm.

[45] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. “Congested traffic states in empirical observations and microscopic simulations”. In: *Physical review E* 62.2 (2000), p. 1805.

[46] Antonin Raffin et al. “Stable-baselines3: Reliable reinforcement learning implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8.

[47] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[48] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).

[49] Tom B Brown. “Language models are few-shot learners”. In: *arXiv preprint arXiv:2005.14165* (2020).

[50] A Vaswani. “Attention is all you need”. In: *Advances in Neural Information Processing Systems* (2017).

[51] Jay Alammar. *The Illustrated Transformer*. Accessed: 2024-11-28. 2018. URL: https://jalammar.github.io/illustrated-transformer/.

[52] Alexey Dosovitskiy. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).

[53] Alec Radford et al. “Learning transferable visual models from natural language supervision”. In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[54] Diederik P Kingma. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).