Bachelor's Thesis Part1

Report

# SAFE MULTI-AGENT CONTROL WITH HUMAN-LIKE DEXTERITY FOR AV'S

**Supervisor:**
Prof. Mayank Baranwal

**By:**
Prachit Gupta
210100111
UG 4th Year
Mechanical Engineering

## Abstract

Reinforcement Learning (RL) offers promising solutions for automated driving by enabling flexible, precise, and human-like behavior. However, challenges in reward design and interpretability hinder its full potential in real-world intelligent driving scenarios like highways. This report explores the integration of Large Language Models (LLMs) to enhance RL agents' decision-making in intelligent driving environments. By leveraging the capabilities of LLMs to generate human-like responses from minimal data, we perform reward shaping for standard DQN agents in the gymnasium's *highway env*. LLMs are trained on huge datasets for a variety of uses. For this particular use case, however, we don't need to exploit the full capabilities of LLMs. Thus an attempt is made to fit a standard SL model on an LLM-generated dataset to bypass the increase in training time associated with prompting LLM repeatedly. Additionally, we explore a novel method of learning safety-critical control for multi-agent systems with decentralized neural barrier certificates as a potential future direction for enhancing safety and human-like dexterity in multi-agent robotic systems.

## Motivation

Uncertainty is inherent in any intelligent driving scenario. The major challenge currently faced by Autonomous vehicles is accounting for this uncertainty without unduly impacting planner performance all the while ensuring safety with other agents in multi-agent settings or with humans in collaborative environments.

### Reinforcement Learning for Safe Human-Like Behaviour

- In recent years RL has proven to be an effective method for safe and efficient navigation in intelligent driving scenarios as shown in 2. However, the challenge remains in devising precise reward functions for promoting safe, and anthropomorphic driving behavior.
- RL agents may resort to reward hacking and potentially engaging in risky behavior to maximize their reward functions necessitating the design of intuitive and balanced reward functions.
- Existing methods aiming at making RL agents mimic human behavior especially for reward design either rely on human feedback as shown in 3 or require a massive amount of data making them resource-intensive and less adaptable
- The traditional reward functions of standard RL environments lack interpretability in decision making further complicating the task of ensuring safety and anthropomorphism

### LLMs For guiding RL agents

- LLMs are trained on massive human datasets and their response can be considered to mimic human behavior for all practical purposes
- LLMs guide RL agents more intuitively to desired outcomes thus integrating them for reward shaping makes it more aligned with human-like decision-making patterns.
- LLMs can also significantly improve the training efficiency of RL agents by optimizing total reward acquisition by generating suitable reward signals.

Thus Large Language Models (LLMs), renowned for their ability to learn from limited context and generate human-like responses through training on human data, are emerging as a promising tool for tailoring reinforcement learning (RL) agents in autonomous driving and enhancing the interpretability of RL-based decisions making them ideal for enhancing safety and anthropomorphism associated with their actions which is the focus of this thesis report.

# Related Work and Literature Review

## 0.1. Reinforcement Learning in Autonomous Driving

3 introduced a Deep Q-Network (DQN)-based approach with safety verification to ensure collision-free lane changes. This method outperformed rule-based agents in safety and learning efficiency.

## 0.2. Large Language Models in Reinforcement Learning

### 0.2.1. Human Feedback in Reward Shaping

*Human Feedback as Action Assignment in Interactive Reinforcement Learning* 3 incorporated human-generated shaping rewards into the RL reward structure, enhancing agent performance. Our work substitutes human input with LLM-based shaping thus reducing notorious manual labour.

### 0.2.2. LLMs as Proxy Reward Functions

*Reward Design with Language Models* 7 demonstrated the use of LLMs as proxy rewards in RL, improving alignment with objectives in negotiation tasks. Unlike their static proxy approach, we explore dynamic, combined rewards in autonomous driving scenarios.

### 0.2.3. Leveraging LLM for Reinforcement Learning Agent in Intelligent Driving Scenario

4 explores various reinforcement learning-based policies for intelligent highway driving scenarios and leverages llm for reward shaping of RL agents. The results of this blog motivate us to train Deep Q learning agents with a combined reward-based strategy to leverage LLM in order to incentivize DQN agents to take safe and human-like actions. A comparative analysis of various reward signals and corresponding RL policy is provided along with optimum prompt construction philosophy.

### 0.2.4. Framework for Intrinsic Reward-Shaping for RL using LLM Feedback

5 proposes three simple methods of reward shaping using LLMs for iteratively generating reward functions during deep RL training. This study experimentally validates that under a fixed number of training iterations, reward shaping with LLMs can achieve higher sample efficiency by injecting its domain knowledge to reduce the agent's exploration of redundant states

# Thesis Objectives

1. Does LLM reward guide the training process of RL agents and achieve a generally high reward?
2. Can we reduce training time for training Deep-Q Networks while shaping rewards with LLMs by fitting appropriate machine learning models?
3. Does the prompt affect the performance or behavior pattern of the LLM-RL agent?
4. Understanding the control-barrier function coupled with neural networks as a potential strategy for promoting safe multi-agent control.

# Preliminaries

## Deep Q-Learning

Deep Q-Learning is an extension of Q-Learning that leverages deep neural networks to approximate the Q-value function, $Q(s,a)$. This approach is particularly advantageous when the state space is large or continuous, making traditional Q-tables impractical.
The Q-value function is updated iteratively using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left( R_t + \gamma \max_{a'} Q(s',a') - Q(s,a) \right), \tag{1}$$

where:

- $\alpha \in [0,1]$: learning rate,
- $R_t$: reward received at time $t$,
- $\gamma \in [0,1]$: discount factor,
- $s'$: next state,
- $a'$: action in the next state.

The following pseudo-code illustrates the Deep Q-Learning algorithm using standard epsilon greedy policy for exploration:

**Algorithm 1:** Deep Q-Learning with $\epsilon$-greedy policy

```
Input: Learning rate $\alpha$, discount factor $\gamma$, exploration rate $\epsilon$, number of episodes $N$, neural network $Q_\theta$
Output: Trained Q-value function $Q_\theta(s,a)$
1  Initialize neural network $Q_\theta$ with random weights;
2  for episode $= 1$ to $N$ do
3      Initialize state $s$;
4      while not terminal state do
5          Generate random number $rand \in [0,1]$;
6          if $rand < \epsilon$ then
7              Choose random action $a$;
8          else
9              Choose $a = \arg\max_{a'} Q_\theta(s,a')$;
10         end
11         Execute action $a$, observe reward $R_t$ and next state $s'$;
12         Compute target $y = R_t + \gamma \max_{a'} Q_\theta(s',a')$;
13         Update neural network weights $\theta$ to minimize loss:
                   $\mathcal{L} = (y - Q_\theta(s,a))^2$
           ;
14         Update state $s \leftarrow s'$;
15     end
16 end
```

This method combines exploration through the $\epsilon$-greedy policy with the Bellman equation for value iteration, using a deep neural network to approximate the Q-value function.

## Reward Shaping

Numerous environments offer a sparse structure of rewards as only one reward signals success or failure to the agent. In such scenarios, the *credit assignment problem* becomes difficult because the agent traverses an ever-increasing number of states to obtain any reward. *Reward shaping* offers a solution by introducing pseudo-rewards at intermediate states, leveraging prior knowledge of what constitutes "good"behavior within the environment. This approach accelerates exploration by encouraging the agent to avoid states known to be "*bad*" and guiding it toward states that are more likely to yield a reward.
The iterative update in Q-learning is expressed as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \tag{2}$$

where $\alpha \in [0,1]$ is the learning rate. This update improves the Q-function by combining the prior Q-value with the temporal difference (TD) error, scaled by the learning rate. The TD error is computed by subtracting the prior Q-value from the TD target.
In reward shaping, we extend this framework by introducing an additional shaped reward, $F(s,s')$, during specific transitions:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[ r + F(s,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]. \tag{3}$$

Here, $F(s,s')$ reflects domain knowledge and is manually encoded or learned. The total shaped reward for an entire episode can then be expressed as:

$$G = \sum_{i=0}^{\infty} \gamma^i \left( r_i + F(s_i, s_{i+1}) \right). \tag{4}$$

The properties of $F(s,s')$ are:

- $F(s,s') > 0$: Provides a positive reward for transitioning from $s$ to $s'$, encouraging such actions.
- $F(s,s') < 0$: Provides a small negative reward for transitioning from $s$ to $s'$, discouraging such actions.

Our approach is to leverage LLMs to dynamically shape rewards based on context and domain-specific requirements. In our case, however, we already have intermediate rewards based on the notions of velocity and collisions. Nonetheless complementing them with LLMs domain knowledge is expected to reduce redundancy in DQN agent's decision making and acceleratethe training process as a whole

## 0.3. Control Barrier Functions

The goal of Control Barrier Functions (CBFs) is to ensure that the system's state remains within a defined safe set. This is achieved by maintaining a scalar safety function $h(x)$ positive at all times:

$$h(x) \geq 0 \implies \text{The system is safe.} \tag{5}$$

The time derivative of $h(x)$ is given by the chain rule:

$$\dot{h}(x) = \frac{\partial h}{\partial x} \cdot \dot{x}. \tag{6}$$

For a system with control input $\mathbf{u}$, where the dynamics are defined as:

$$\dot{x} = f(x) + g(x)\mathbf{u}, \tag{7}$$

the time derivative of $h(x)$ becomes:

$$\dot{h}(x) = \frac{\partial h}{\partial x} \cdot f(x) + \frac{\partial h}{\partial x} \cdot g(x)\mathbf{u}. \tag{8}$$

To enforce safety, the following condition is imposed:

$$\dot{h}(x) \geq -\alpha h(x), \tag{9}$$

where $\alpha > 0$ is a tunable parameter controlling the responsiveness of the system near the safety boundary. Substituting $\dot{h}(x)$, the condition becomes:

$$\frac{\partial h}{\partial x} \cdot f(x) + \frac{\partial h}{\partial x} \cdot g(x)\mathbf{u} \geq -\alpha h(x). \tag{10}$$

This can be rewritten as a linear constraint on the control input $\mathbf{u}$:

$$\mathbf{a}^\top \mathbf{u} \geq b, \tag{11}$$

where:

$$\mathbf{a} = \frac{\partial h}{\partial x} \cdot g(x), \quad b = -\alpha h(x) - \frac{\partial h}{\partial x} \cdot f(x).$$

By solving the above linear inequality, we can ensure that the system remains within the safe set.

## Environment Setup

In this work, we build a custom environment on Highway-env [1] based on stable-baseline3

### Observations

- The experimental setup consists of 10 vehicles in each observation frame
- For each vehicle the observation space consists of lateral position, lane id, and velocity in lateral and longitudinal directions respectively.

### Configuration

Configuration settings are controlled by the following parameters: `vehicle_density` (controls congestion), `ego_spacing`, `lane_id`, and `initial_spacing`. These parameters are appropriately determined and varied according to the use case (generalized training or random data generation).

### Actions

Discrete Meta action space used with following possible actions:

- **0:** `LANE_LEFT`
- **1:** `IDLE`
- **2:** `LANE_RIGHT`
- **3:** `FASTER`
- **4:** `SLOWER`

### Reward used

The reward function is designed to prioritize two key objectives:

- The vehicle should progress quickly on the road.
- The vehicle should avoid collisions.

Based on these objectives, the reward function is typically composed of a velocity term and a collision term:

$$R(v, c) = \beta_v \frac{v - v_{\min}}{v_{\max} - v_{\min}} - \beta_c c, \tag{12}$$

This reward structure encourages the vehicle to maintain high speeds while penalizing collisions, effectively balancing the trade-off between safety and efficiency

## DQN for Highway-env

DQN agent trained on custom environment described above in 100000 steps with a learning rate of 5e-4 and an exploration factor of 0.7 to start with.

### Shortcomings

- Reward function is straightforward and doesn't completely capture notions of safety and efficacy of actions
- Agent in some cases performs aggressive and frequent lane changes and sudden acceleration which are not feasible in real-life scenarios.

## LLM-setup

Claude Sonnet is deployed for reward shaping as it explicitly targets conversational AI with enhanced reasoning and ethical dimensions. As mentioned in 10, Claude-3.5 demonstrates greater adaptability when handling complex or evolving prompts, as it is engineered to manage intricate dependencies and logical chains in text.

| **Criteria** | **Phi-3** | **Groq** | **Claude-3.5 (Sonnet)** |
|---|---|---|---|
| **Action Quality** | Repeatedly recommended sub-optimal actions such as aggressive maneuvers. | Similar to Phi-3, often suggested sub-optimal actions, not prioritizing safety as prompts became complex. | Produced more optimal and human-like responses, aligning with safety prioritization. |
| **Safety Prioritization** | Limited ability to prioritize safety in complex scenarios. | Faced challenges with maintaining safety focus as the prompt complexity increased. | Demonstrated strong safety prioritization, with responses closer to human judgment. |
| **Adaptability to Complex Prompts** | Struggled with adapting effectively to nuanced and complicated scenarios. | Similar challenges in handling complex, layered prompts effectively. | Showed better adaptability to complex prompts, providing nuanced and thoughtful responses. |

## Problem Formulation

An autonomous driving scenario can be formalized as a Markov Decision Process (MDP):

$$M = \langle S, A, \rho, R, \gamma \rangle,$$

where:

- $S$: State space representing driving scenarios (e.g., traffic density, relative vehicle speeds, lane positions).
- $A$: Action space for possible driving actions (e.g., accelerate, decelerate, change lanes).
- $\rho : S \times A \times S \to [0, 1]$: Transition probability function indicating the likelihood of state transitions given an action.
- $R : S \times A \to \mathbb{R}$: Reward function mapping states and actions to real-valued rewards.
- $\gamma$: Discount factor representing the agent's focus on future rewards.

To enhance the reward mechanism, a Large Language Model (LLM) is employed as a reward proxy. The LLM is modeled as:

$$\text{LLM} : A^* \to A^*,$$

where it takes a concatenated prompt $\rho$ as input and outputs a string. A parser:

$$g : A^* \to \{0, 1\},$$

maps the LLM's textual output to a binary reward signal. This framework replaces the traditional reward function $R$ with the proxy reward function LLM, and can be used with any Reinforcement Learning algorithm.

## Prompt Design

Inspired by 4 and 6, A prompt in one interaction is constructed as scene description, last action description, pre-define prompt, available actions, and suggested answer format.
The prompts consist of three main components to elicit a task-specific rule-based robot controller:

1. **Basic Info and Task Descriptions:** Provide the fundamental setup for the task, including details about the ego vehicle, basic environment description, available actions to perform, and traffic rules or attention points.
2. **Highway Scenario Description:** illustrate the context of the current road situation, basically we converted the RL agent's current observation and actions into a natural language context through several analysis functions, which could then, be processed by LLM
3. **Safety Verification:** Additional layer of safety added on top of existing prompt asking llm to re-evaluate its recommended actions based on safety verification parameters 11.

Below, we describe prompt design strategies for two specific environments.

### Roundabout Environment

The roundabout environment presents unique challenges such as merging, yielding, and maintaining safe interactions with multiple agents. The prompt design for this scenario includes:

## Basic Information and Task Description

```
# Part 1: Initial prompt introducing the scenario
prompt1 = 'You are Claude, a large language model. You are now acting as a mature
    driving assistant, who can give accurate and correct advice for human drivers in
    complex urban driving scenarios.'

env_description = "ego-vehicle is approaching a roundabout with flowing traffic. Ego
    must handle lane changes and longitudinal control to pass the roundabout as fast as
     possible while avoiding collisions."

rules = "There are several rules for highway driving:
1. Maintain a safe distance from the car ahead.
2. Avoid frequent lane changes; double-check before switching lanes.
3. On a roundabout, complete half a circle before continuing forward."

prompt1 += env_description + rules
```

## Scene Description

```
# Part 5: Describing the current highway scenario
prompt2 = 'Here is the current scenario:
There are 2 lanes on the roundabout: Lane-1, Lane-2.'

# Ego vehicle details
ego_position = self.env.unwrapped.vehicle.position  # (x, y)
ego_lane = self.env.unwrapped.road.network.get_closest_lane_index(ego_position,
    ego_heading)[2]
ego_x, ego_y = ego_position
ego_vx, ego_vy = vx[0], vy[0]

# Append ego vehicle information
prompt2 += f'Ego vehicle:
    Current lane: Lane-{ego_lane}
    Current speed: {ego_vy} m/s'

# Lane info with vehicles ahead
lane_info = 'Lane info:\n'
for i in range(2):
    inds = np.where(veh_lanes == i)[0]
    num_v = len(inds)
    if num_v > 0:
        closest = min(abs(veh_y[inds]))
        lane_info += f'\tLane-{i}: {num_v} vehicle(s) ahead, closest {closest} m away
            .'
    else:
        lane_info += f'\tLane-{i}: No vehicles ahead.'

prompt2 += lane_info
```

### Safety Verification

```
# Part 6: Adding attention points and final decision instruction
safety_verification = '''
Attention points:
1. Safety is the main priority; avoid collisions at all costs.
2. When merging into the roundabout, check for traffic and slow down if necessary.
3. Avoid excessive idling. Make efficient decisions unless safety is compromised.
4. Always verify the safety of your decision. If unsafe, redo the decision process.
5. Choose one from: IDLE, SLOWER, FASTER, LANE_LEFT, LANE_RIGHT.

Your last action was ' + self.prev_action + '. Recommend action only as:
Final decision: <final decision>.
'''
prompt2 += safety_verification
```

## Standard Highway Environment

The standard highway environment involves lane-keeping, lane-changing, and overtaking maneuvers. The prompt design includes:

### Basic Info and Task Description

Listing 1: Basic Info and Task Description

```
You are Claude, a large language model. You are now acting as a mature driving
   assistant, who can give accurate and correct advice for human drivers in complex
   urban driving scenarios. The information in the current scenario:
- You, the 'ego' car, are now driving on a highway. You have already driven for 0
   seconds.
- The decision made by the agent LAST time step was 'FASTER' (accelerate the vehicle).

There are several rules you need to follow when you drive on a highway:
1. Try to keep a safe distance from the car in front of you.
2.  D O N T  change lanes frequently. If you want to change lanes, double-check the
   safety of vehicles in the target lane.
```

### Scene Description

Listing 2: Scene Description

```
Here is the current scenario:
There are four lanes on the highway: Lane-1 (leftmost), Lane-2, Lane-3, Lane-4 (
   rightmost).

Ego vehicle:
- Current lane: Lane-X
- Left lane: [availability information]
- Right lane: [availability information]
- Current speed: [ego_speed] m/s

Lane info:
Lane-1: [description of vehicles ahead, distance, and speeds]
Lane-2: [description of vehicles ahead, distance, and speeds]
Lane-3: [description of vehicles ahead, distance, and speeds]
Lane-4: [description of vehicles ahead, distance, and speeds]
```

## Safety Verification

Listing 3: Safety Verification

```
Attention points:
1. Safety is the main priority. You can stay IDLE or even go slower, but under no
   circumstances should you collide with the lead vehicle.
2. You are not supposed to change lanes frequently, only when necessary to keep the
   vehicle safe. Before changing lanes, check safety factors like safe distance and
   speed of other vehicles.
3. Safety is a priority, but do not forget efficiency.
4. You should only make a decision once you have verified safety with other vehicles.
   Otherwise, make a new decision and verify its safety from scratch.
5. Your suggested action has to be one from the five listed actions:
   - IDLE
   - SLOWER
   - FASTER
   - LANE_LEFT
   - LANE_RIGHT

Your last action was '[last_action]'. Please recommend an action for the current
   scenario only in this format:
Final decision: <final decision>
```

The above structure proved to be most optimum for Claude 3.5 devised via *iterative refinement* to ensure that the guidance provided was practical and beneficial to the agent's learning process:

- The agent only changes lanes when it's both safe and practical i.e maintains a safe distance from the lead vehicle as well as from vehicles in adjacent lanes
- In the roundabout environment, before merging, the agent stops and performs safety checks regarding safe distances.
- However, The response time of llm is enhanced significantly due to complicated prompts and the safety verification layer
- Almost no aggressive jerky maneuvers are performed

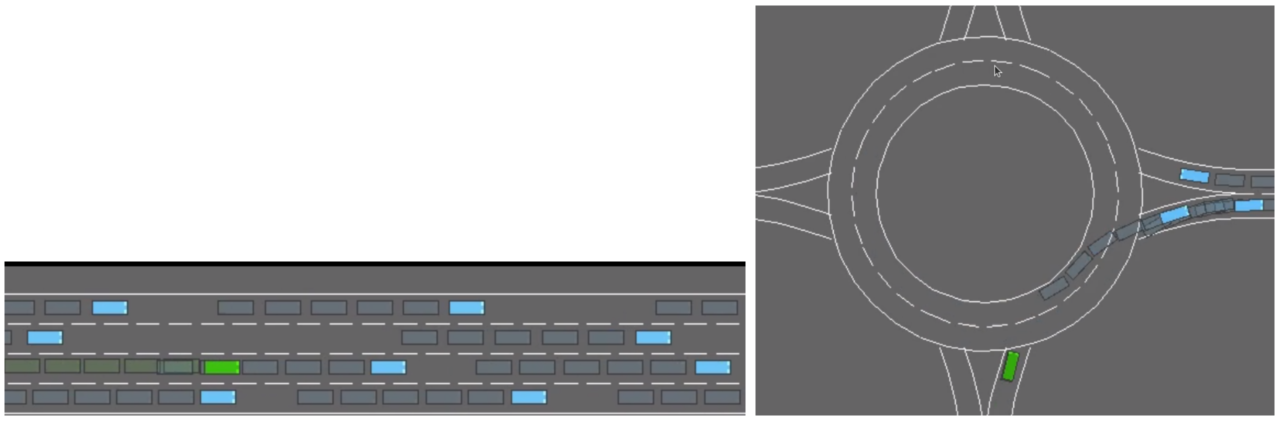


Figura 1: Results with claude

## Reward signal generation

- Based on comparative analysis performed in 4 we choose a combined reward strategy i.e. sum the original reward from the environment and the reward signal given from LLM.
- Given the original reward function in a highway driving environment, We propose an adjusted reward function that integrates a reward signal from LLM as
**0.7 dqn_reward + 0.3*llm_reward** where llm_reward is obtained by judging based on LLM's standard answer for reward, output 1 if it conforms to LLM's standard answer, otherwise output 0.

## Reducing training Time for DQN agents

Though injecting intermediate intuitive balanced rewards by integrating LLMs by itself accelerates the training process and reduces redundancy, the entire process of prompting the Claude through API and then mapping the response to appropriate action is time-consuming and takes a large chunk of training time which is undesirable.
To bypass this entire loop of repeatedly prompting Claude for a response, we proposed a novel technique of generating a labeled dataset from the highway environment as done in 12 and then fit any standard classification ML model which can be queried for LLM response instead of prompting it throughout training interval

### Dataset Generation

The dataset was generated by simulating the environment with various configurations and capturing observations alongside prompting Claude for corresponding actions.

- Given a 10 x 5 observation space, we have 50 features namely {presence (boolean), x (in m), y (lane_id), vx (in m/s), and vy (in m/s)}.
- **Set Configuration Parameters:** Random values were assigned to key configuration parameters from a uniform normal distribution:
  - `vehicles_density`: Controls the congestion level in the environment, sampled from a predefined range to introduce variability.
  - `initial_spacing`: Specifies the initial spacing between vehicles, ensuring realistic traffic dynamics.
  - `initial_lane_id`: Randomly selects the starting lane for the ego vehicle to explore different lane configurations.
  - `ego_spacing`: Determines the ego vehicle's initial spacing relative to other vehicles for diverse initialization scenarios.
- **Generate Prompts:** Using the observation, we created prompts for Claude according to the prompt design given above to analyze the state and determine appropriate actions.
- **Map LLM Actions to Labels:** The LLM's suggested action was mapped to a numerical label using a predefined mapping function.
- **Save the Dataset:** After all samples were generated, the dataset was saved to a file for subsequent use in training and evaluation.

### Preprocessing Techniques

Several preprocessing techniques were applied to prepare the data for classification:

1. **Removing Constant Features:** Columns with constant values which mainly comprised of the "presence"features across the dataset were removed to reduce redundancy and computational overhead.
2. **Standardization:** Features were standardized using `StandardScaler` to ensure all features have mean zero and unit variance, essential for models like t-SNE and SVM.
3. **Handling Class Imbalance:**
   - **Downsampling Majority Classes:** Majority classes were downsampled to match the size of minority classes.
   - **SMOTE (Synthetic Minority Oversampling Technique):** Minority classes (faster, and lane-changing actions) were oversampled to balance the class distribution and improve model performance.
4. **One-Hot Encoding:** Categorical variables (lane_id) with limited unique values were one-hot encoded to represent them in a binary format suitable for machine learning models.
5. **Dimensionality Reduction:** Principal Component Analysis (PCA) was applied to reduce dimensionality while retaining 98 % of the variance.

## Results

### Model Accuracy

| **Model** | **Accuracy (%)** |
|---|---|
| Random Forest (RF) | 53.78 |
| XGBoost (XGB) | 40.29 |
| Ensemble (Voting Classifier) | 42.33 |
| Multi-layer Perceptron (MLP) | 29.00 |
| Support Vector Classifier (SVC) | 28.00 |

Despite preprocessing efforts and applying various machine learning models, classification on the given dataset remains challenging. The primary reasons include:

- **Class Imbalance:** Significant imbalance in class distribution leads to poor generalization, especially for minority classes.
- **Overlapping Features:** Features do not sufficiently separate the classes, as observed in the t-SNE plot.
- **Noisy Data:** The dataset contains noise or redundant information that hinders accurate predictions.

## t-SNE Visualization

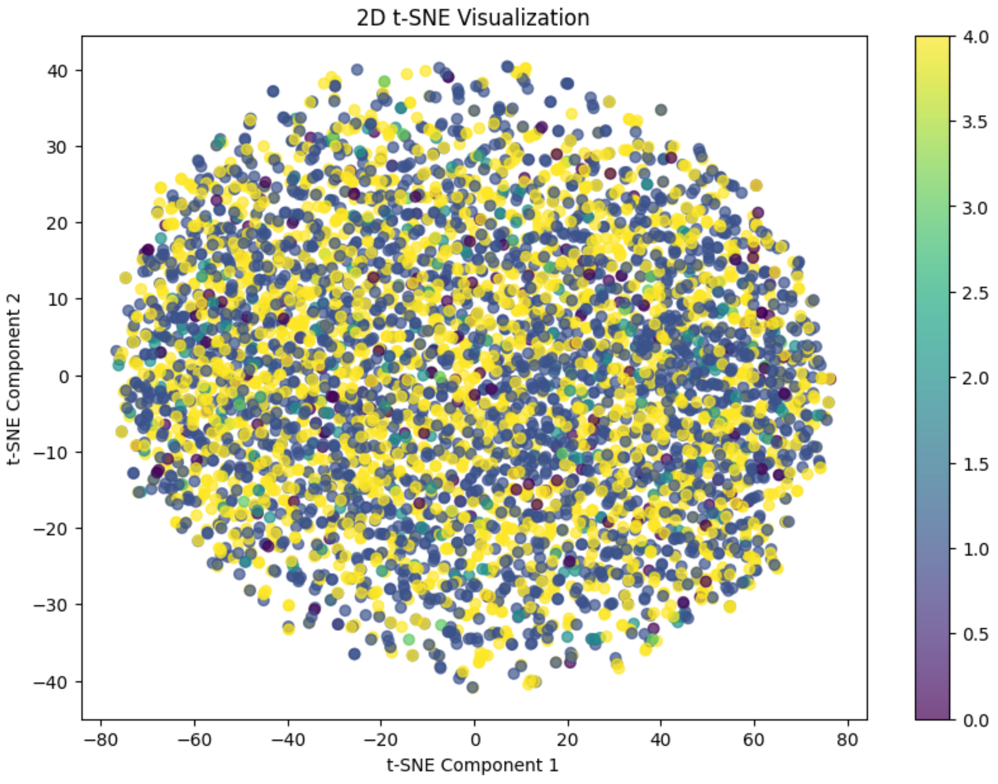
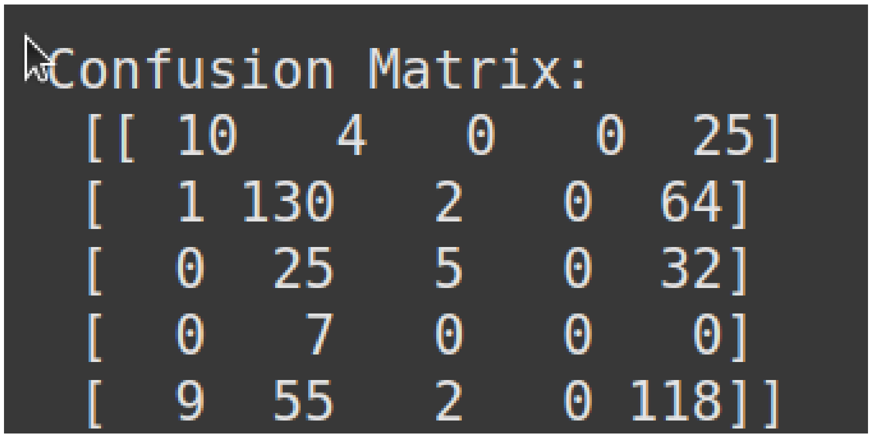


Figura 2: 2D t-SNE Visualization of the Dataset

## Confusion Matrix



```
Confusion Matrix:
 [[ 10   4   0   0  25]
 [  1 130   2   0  64]
 [  0  25   5   0  32]
 [  0   7   0   0   0]
 [  9  55   2   0 118]]
```

Figura 3: Confusion Matrix for Random Forest Classifier

This suggests that given available features from the predefined highway-env it is not possible to classify the data points into corresponding action labels without having new features and the model demonstrates an extremely high confusion with action 4 or moving slower

# Safe Multi-agent control with decentralized neural certificates

Another part of this project was to understand the formulation and proceed with the Pytorch implementation of decentralized NN-based control policies with Control Barrier Functions (CBFs)as safety certificates in multi-agent systems. The work is based on the paper "Learning Safe Multi-Agent Control with Decentralized Neural Barrier Certificates"(Qin et al., ICLR 2021).

## 1. Theoretical Formulation

**Control Lyapunov Functions (CLFs):** CLFs ensure system stability by decreasing a scalar energy function $V(x)$ over time. For a system $\dot{x}(t) = f(x, u)$, a CLF satisfies:

$$V(x) > 0 \quad \forall x \neq 0, \quad V(0) = 0, \quad \text{and} \quad \dot{V}(x, u) < 0.$$

**Control Barrier Functions (CBFs):** CBFs extend CLFs to enforce safety by maintaining the state $x$ within a predefined safe set $S = \{x \mid h(x) \geq 0\}$. A CBF $h(x)$ satisfies:

$$\dot{h}(x, u) = \nabla h(x) \cdot f(x, u) + \alpha(h(x)) \geq 0, \quad \forall x \in S,$$

where $\alpha(\cdot)$ is a class-K function.

**Integration of CLFs and CBFs:** By combining CLFs and CBFs, controllers are designed to achieve both stability and safety. This involves solving a Quadratic Programming (QP) problem:

$$\min_{u} \|u\|^2 \quad \text{subject to} \quad \dot{V}(x, u) < 0, \; \dot{h}(x, u) \geq 0.$$

**Learning Framework:** The decentralized framework trains two neural networks per agent:

- A **Control Policy** $\pi_i(s_i, o_i)$, mapping state $s_i$ and observation $o_i$ to actions $u_i$.
- A **CBF** $h_i(s_i, o_i)$, ensuring safety by enforcing $h_i(s_i, o_i) \geq 0$.

The optimization objective combines safety conditions and task-specific goals:

$$\max_{h_i, \pi_i} y_i(\tau_i, h_i, \pi_i) \quad \text{subject to} \quad y_i(\tau_i, h_i, \pi_i) \geq \gamma,$$

where $\gamma > 0$ is a safety margin, and $y_i$ encapsulates safety and task performance.

## 2. PyTorch Implementation

**Modular Structure:** The implementation leverages PyTorch for flexibility and scalability, with the following components:

- **config.py:** Defines hyperparameters and configurations.
- **core.py:** Implements neural networks for CBFs and policies, and helper functions for collision detection.
- **train.py:** Manages data collection, loss computation, and parameter updates.
- **evaluate.py:** Evaluates learned models in unseen scenarios.

**Training Process:**

1. **Data Collection:** Simulate trajectories using the current policy $\pi_i$.

2. **Loss Computation:** Combine safety loss ($L_c$) and task completion loss ($L_g$):

$$L = L_c + \eta L_g,$$

   where $L_c$ enforces $h_i \geq 0$ and $\dot{h}_i \geq -\alpha(h_i)$, and $L_g$ minimizes the deviation from a reference controller.

3. **Parameter Updates:** Use backpropagation and gradient descent to update the networks.

4. **Iteration:** Repeat until convergence.

**Evaluation:** The trained models are tested in structured (e.g., polygonal) and random environments. Key metrics include:

- **Safety Ratio:** Percentage of agents avoiding collisions.
- **Distance Error:** Average deviation from goal positions.
- **Trajectory Efficiency:** Smoothness and directness of paths.

## Future Directions

- Train DQN agent in highway and roundabout-env using proposed prompt design directly by leveraging claude
- Maybe try intrinsic reward shaping like 5 for reducing training time
- Integrate LLM-based reward shaping in multi-agent settings like MARL (Sagar's thesis)

## Software Contributions

- Developed a Reward shaping-LLM package based on gymnasium with documentation on:
  - Generating dataset for LLM
  - Training DQN agent with Claude, groq and phi3
  - Evaluating performances of only RL, only LLM and Rl integrated with LLM approaches with visual feedback

## References

1. https://github.com/Farama-Foundation/HighwayEnv/blob/master/docs
2. High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning: Branka Mirchevska, Christian Pek, Moritz Werling, Matthias Althoff, and Joschka Boedecker
3. Human Feedback as Action Assignment in Interactive Reinforcement Learning: SYED ALI RAZA and MARY-ANNE WILLIAMS
4. Leveraging LLM for Reinforcement Learning Agent in Intelligent Driving Scenario: Jingyuan Zhang
5. A Simple Framework for Intrinsic Reward-Shaping for RL using LLM Feedback: Alex Zhang, Ananya Parashar, Dwaipayan Saha

6. Accelerating Reinforcement Learning of Robotic Manipulations via Feedback from Large Language Models: Kun Chu, Xufeng Zhao, Cornelius Weber, Mengdi Li, Stefan Wermter

7. REWARD DESIGN WITH LANGUAGE MODELS: Minae Kwon, Sang Michael Xie, Kalesha Bullard† , Dorsa Sadigh

8. Control Barrier Functions - A Simple Case Study - Tech Notes

9. Steve Brunton Rl series - https://cassyni.com/events/LDfHwDgTPqnUuz3JoEEbCZ

10. Claude Sonnet - https://www.anthropic.com/news/claude-3-5-sonnet

11. Prompt Engineering: https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-

12. Enhancing Autonomous Vehicle Training with Language Model Integration and Critical Scenario Generation: Hanlin Tian, Kethan Reddy, Yuxiang Feng

13. Learning Safe Multi-Agent Control with Decentralized Neural Barrier Certificates"(Qin et al., ICLR 2021