R & D Project

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) enables flexible and human-like behavior for autonomous driving, but reward design and interpretability challenges limit its real-world applicability. This work investigates Large Language Models (LLMs) to improve RL decision-making in driving environments, specifically using reward shaping for DQN agents in the gymnasium's *highway-env*. This work explores dataset generation using Claude 3.5 Sonnet for two distinct driving scenarios—roundabout and highway. We investigate optimal strategies to train lightweight supervised learning models on this synthetic data, then integrate them with standard DQN networks to guide RL agent to take safe but efficient actions mimicing human drivers. Additionally, we explore decentralized graph control barrier certificates (GCBFs) implemented via pytorch geometric as a future direction for scalable, safety-critical control in multi-agent systems, enhancing robustness to dynamic topologies and initial configurations.

Problem Description

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. On one hand we need the agent to be **risk-averse** and **efficient** incentivizing choosing reward functions like :

$$R(v, c) = \beta_v \frac{v - v_{\min}}{v_{\max} - v_{\min}} - \beta_c c, \tag{1}$$

penalizing collisions and promotimg high speeds but this may lead to **overemphasize large but rare events** (e.g., collisions) and **ignore smaller but frequent rewards** and causing instability in training due to vanishing gradients . One way to mitigate the same is to **normalise the reward signal** using sigmoid function but this approach can cause poor reward shaping leading to **halting** (velocity =0) as it **smooths out critical reward differences** making it hard for the network to differentiate between safe slow driving and effective fast driving.

LLMs for reward shaping

- 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 LLMs, renowned for their ability to learn from limited context and generate human-like responses can be used as a 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.

Report Objectives

1. Can we reduce training time and computational resources for training Deep-Q Networks while shaping rewards with LLMs by fitting appropriate machine learning models?
2. Understanding the graph control-barrier function certificates learned jointly with the control policy using GNNs as a potential strategy for promoting safe multi-agent navigation in dynamic topology and agent configurations.

Previous Work

In 1, it was demonstrated that Claude 3.5-Sonnet can serve as an effective tool for intuitively guiding the actions of reinforcement learning agents. Detailed formulations for prompt design were provided for both environments. It was shown that, with appropriately crafted prompts, the LLM alone can function as a capable driving agent, achieving performance on par with DQN models—and in fact, surpassing them in terms of speed and efficiency. However, the dataset generated using the LLM relied on standard observations from the highway-env, which introduced challenges for fitting a simple ML model due to correlated features and insufficient data for certain actions.

Gaps

- A significant imbalance in the class distribution leads to poor generalization, particularly for minority classes.
- The presence of highly correlated features limits the model's ability to effectively separate the classes.
- The absence of domain expertise and lack of data tailored to diverse scenarios. For example, ensuring an adequate number of instances where Claude recommends *"FASTER"* or *ÏDLE"* actions following a previous *"SLOWER"* action, which typically prevents a halting situation was largely absent in the dataset generation.

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. 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.

0.2.2. 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.

0.2.3. 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

Preliminaries

Feature Engineering

Feature engineering is the process of transforming raw input data into meaningful numerical representations (features) that improve the performance of machine learning models. In supervised learning, each data sample is a feature-label pair:

$$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}, \quad x^{(i)} \in \mathbb{R}^d$$

A new transformed feature space $\phi(x) \in \mathbb{R}^{d'}$ may be constructed where $d' \geq d$, often capturing nonlinear or higher-order interactions. For example, including polynomial terms like x^2 allows linear models to fit quadratic trends:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon$$

Effective features should ideally be:

- **Relevant** to the target output.
- **Non-redundant** and non-collinear.
- **Normalized** for numerical stability.
- **Interpretable**, if needed.

Feature Selection

Feature selection involves identifying a subset of features that contribute the most to prediction accuracy. Given a full feature set $\{x_1, x_2, \ldots, x_d\}$, the objective is to find a subset $S \subset \{1, \ldots, d\}$ of size $k \ll d$ such that:

$$\max_S \operatorname{Score}(f_S)$$

where f_S is a model trained using only features in S, and Score is a chosen performance metric.
Main techniques:

- **Filter Methods**: Rank features by statistical tests (e.g., correlation, mutual information).
- **Wrapper Methods**: Use model performance to evaluate subsets (e.g., recursive feature elimination).
- **Embedded Methods**: Perform selection during training (e.g., LASSO with ℓ_1-penalty).

Reducing dimensionality improves generalization and combats the curse of dimensionality, where data becomes sparse in high-dimensional spaces.

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 accelerate the training process as a whole

Graph Neural Networks

Graph Neural Networks (GNNs) are specialized neural architectures designed to process graph-structured data, making them highly effective for multi-agent systems (MAS) where agents and their interactions are represented as nodes and edges in a graph. In distributed control scenarios, GNNs model agents as nodes (e.g., controlled agents or LiDAR observation points) and their interactions as edges based on a sensing radius R. Node features v_i encode agent types (e.g., $v_i = 0$ for controlled agents, $v_i = 1$ for uncontrolled obstacles), while edge features e_{ij} capture relative information such as positions or velocities between agent i and neighbor j. GNNs aggregate information from neighbors using attention mechanisms to prioritize critical interactions, enabling scalable and distributed control policies. The aggregation process is formalized as:

$$q_{ij} = f_{\theta_1}(e_{ij}, v_j),$$

$$q_i = \sum_{j \in \mathcal{N}_i} \text{softmax}(f_{\theta_2}(q_{ij})) \cdot f_{\theta_3}(q_{ij}),$$

where $f_{\theta_1}, f_{\theta_2}, f_{\theta_3}$ are neural networks parameterizing the encoding and attention processes, and $\mathcal{N}_i$ is the set of neighbors of agent i. The output q_i is further processed to compute control actions or safety certificates, ensuring adaptability to varying graph sizes and topologies. This approach is particularly useful in applications like multi-agent path finding and resilient coordination under agent attrition or communication disturbances, as demonstrated in crowd-aware planning and multi-robot systems.

The goal of integrating GNNs with control frameworks is to learn distributed policies that generalize to large-scale MAS. By leveraging local information, each agent reacts based on its neighborhood without requiring global system knowledge, addressing scalability and generalizability challenges in collision-avoidance tasks. GNNs also enhance situational awareness through local communication, improving decision-making in congested or dynamic environments.

Graph Control Barrier Functions

Graph Control Barrier Functions (GCBFs) extend traditional Control Barrier Functions (CBFs) to MAS by incorporating graph structures for distributed safety guarantees, particularly collision avoidance. The objective of GCBFs is to ensure that each agent's state remains within a defined safe set, avoiding collisions with other agents and obstacles. This is achieved by maintaining a scalar safety function $h(\bar{e}_i, \bar{v}_i)$ positive at all times:

$$h(\bar{e}_i, \bar{v}_i) \geq 0 \implies \text{The system is safe for agent } i,$$

where $\bar{e}_i$ and $\bar{v}_i$ represent collected edge and node features for agent i, respectively.

The time derivative of h accounts for the dynamics of the agent and its neighbors, given by:

$$\dot{h}(\bar{e}_i, \bar{v}_i) = \frac{\partial h}{\partial \bar{e}_i} \cdot \frac{\partial \bar{e}_i}{\partial x_i} \cdot F(x_i, u_i) + \frac{\partial h}{\partial \bar{e}_i} \cdot \sum_{j \in \mathcal{N}_i} \frac{\partial \bar{e}_i}{\partial x_j} \cdot F(x_j, u_j),$$

where $F(x_i, u_i)$ represents the nonlinear dynamics of agent i, and u_i is the control input. For a general system with dynamics:

$$\dot{x}_i = f(x_i) + g(x_i)u_i,$$

the derivative becomes:

$$\dot{h} = \frac{\partial h}{\partial \bar{e}_i} \cdot \frac{\partial \bar{e}_i}{\partial x_i} \cdot f(x_i) + \frac{\partial h}{\partial \bar{e}_i} \cdot \frac{\partial \bar{e}_i}{\partial x_i} \cdot g(x_i)u_i + \text{neighbor terms.}$$

To enforce safety, the following condition is imposed:

$$\dot{h}(\bar{e}_i, \bar{v}_i) \geq -\alpha h(\bar{e}_i, \bar{v}_i),$$

where $\alpha > 0$ is a tunable parameter controlling responsiveness near the safety boundary. This condition translates into a constraint on the control input u_i:

$$\mathbf{a}^\top u_i \geq b,$$

where:

$$\mathbf{a} = \frac{\partial h}{\partial \bar{e}_i} \cdot \frac{\partial \bar{e}_i}{\partial x_i} \cdot g(x_i), \quad b = -\alpha h(\bar{e}_i, \bar{v}_i) - \frac{\partial h}{\partial \bar{e}_i} \cdot \frac{\partial \bar{e}_i}{\partial x_i} \cdot f(x_i) - \text{neighbor terms.}$$

By solving this inequality, GCBFs ensure that agent i remains in the safe set $\mathcal{S}_i$, defined as states where inter-agent distances exceed a safety threshold and no collisions occur with obstacles. GNNs are employed to learn GCBFs, enabling distributed and scalable safety enforcement across large-scale MAS, even with dynamic topologies and LiDAR-based observations.

Highway Environment Setup

In this work, we build a custom environment on Highway-env as described in the previous version of this work in 1 based on stable-baseline3

Scene: Straight multi-lane highway with dense traffic flow. Ego vehicle must navigate while avoiding collisions.

Observation Space:

- `Kinematics` features: presence, x, y coordinates, velocity components v_x, v_y
- Absolute coordinates with normalization

- Tracks $N =$ `vehicleCount` nearest vehicles
- 360° perception (`see_behind=True`)

Action Space:

- `DiscreteMetaAction` with 5 discrete choices (IDLE, SLOWER, LEFT, RIGHT, FASTER)
- Meta-actions include lane changes and speed adjustments

0.3. Dataset Generation

0.3.1. Collision-Free Episode Filtering

The dataset generation strictly records only collision-free interactions:

- **Validation Check:** After each episode, the environment's `info` dictionary is inspected for `çrashed": True` flags
- **Data Discarding:** Full episode trajectories are excluded if any collision occurs during the 40-timestep horizon
- **Progressive Saving:** Two datasets are maintained - `_all` (raw) and `_collision_free` (filtered) versions
- **Safety Margin:** 10,000m placeholder distance assigned to empty lanes to prevent false collision positives

0.3.2. Halting Mitigation Strategy

The `SLOWER` action bias is addressed through:

- **Action Sequencing:** Forced alternation between `SLOWER` and `FASTER/IDLE` in high-density scenarios (`vehicles_density >1.5`)
- **Temporal Context Injection:** Previous action history (1-step) included in observations to break action inertia
- **Traffic Forcing:** Configurations with `initial_spacing <15m` require subsequent acceleration after slowing
- **Reward Shaping:** Exponential decay reward $r = \frac{1}{1+e^{-R_{\text{env}}}}$ penalizes prolonged low-speed operation

Feature Selection

0.3.3. Feature Selection Criteria

The 10 critical features were selected through domain-driven analysis of driving dynamics:

- **Vehicle Density:** Counts in ego lane (current lane), left lane, and right lane – 3
- **Proximity Metrics:** Distance to closest vehicle in ego/left/right lanes – 3
- **Relative Motion:** Velocity differences with closest vehicles in each lane –3
- **Temporal Context:** Previous control action (1-step history) –1

0.3.4. Observation Space Compression

The conversion from 50-D observations to 10-D features follows:

1. **Ego State Extraction:** Isolate position (x, y), velocity (v_x, v_y), and lane ID
2. **Neighbor Processing:**
   - Reshape 90 neighbor features into 9 vehicles $\times$ 5 attributes
   - Calculate relative distances $\Delta x = |x_{\text{ego}} - x_{\text{neighbor}}|$
   - Compute relative velocities $\Delta v = v_{\text{neighbor}} - v_{\text{ego}}$
3. **Lane-wise Aggregation:**
   - For each lane direction (ego/left/right):
   - Find closest vehicle using masked distance minimization
   - Assign 10,000m distance placeholder for empty lanes
4. **Temporal Features:** Append previous action from timestep $t - 1$

0.3.5. Theoretical Justification

This feature set aligns with machine learning principles:

- **Dimensionality Reduction:** Mitigates curse of dimensionality $(50 \to 10)$
- **Relevance Filtering:** Retains only collision-critical spatial-temporal features
- **Physics Compliance:**
  - Relative velocities enable Time-to-Collision (TTC) estimation
  - Lane-specific distances capture collision risk vectors
- **Markovian Preservation:** Previous action inclusion maintains state transition memory

$$\text{Feature Relevance Score} \propto \frac{1}{\Delta x} + |\Delta v| + \mathbb{I}_{\text{lane conflict}}$$

0.4. Data Analysis

To address significant class imbalance in the action labels, the following strategies were adopted:

- **Upsampling:** Minority classes (`LEFT` and `RIGHT` actions) were upsampled by synthetically generating additional scenarios where these actions are recommended.
- **Downsampling:** Overrepresented majority classes (such as `IDLE`) were downsampled to prevent model bias.
- **Class Weights:** During model training, class weights were applied in the loss function to further compensate for any residual imbalance and improve generalization across all action types.

Multiple Random forests strategy

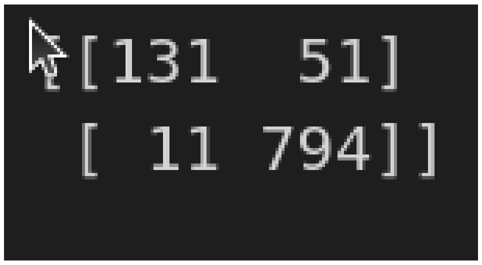
Three separate RandomForest models were trained for action classification. First, a binary classifier distinguishes majority classes (FAST, IDLE, SLOW) from minority classes (LEFT, RIGHT). Then, two additional RandomForests are used: one for classifying among the majority actions and another for distinguishing LEFT and RIGHT. This approach, combined with feature engineering, produced promising results and helped overcome the class imbalance challenges described in 1.

1. Results

1.1. RandomForest Model Performance

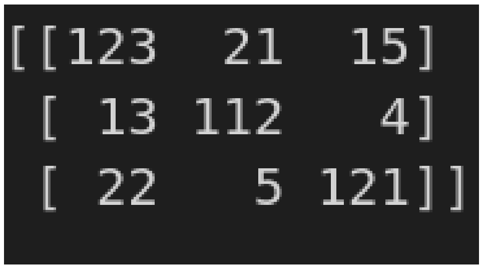
Cuadro 1: Performance Metrics for RandomForest Models

Model	Accuracy	Precision	Recall
Binary Classifier	93.72 %	93.96 %	98.63 %
Majority Class RF	81.65 %	78 %	77 %
Minority Class RF	98.21 %	96 %	100 %



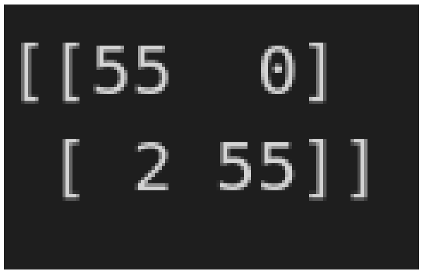
```
[[131  51]
 [ 11 794]]
```

(a) Binary Classifier



```
[[123  21  15]
 [ 13 112   4]
 [ 22   5 121]]
```

(b) Majority Classes



```
[[55  0]
 [ 2 55]]
```

(c) Minority Classes

Figura 1: Confusion matrices for the three RandomForest models

1.2. Performance Comparison

Metric	Claude Only	Claude with RF	DQN
Average Score per episode	25.83	26.12	26
Collisions/50eps	6	7	8

Roundabout Environment Setup

The roundabout environment extends the same foundation as the highway setup in 1, with structural adaptations for circular intersection dynamics.

Scene: Multi-lane circular intersection with merging traffic streams. Ego vehicle must negotiate right-of-way while maintaining safe gaps.

Action Space:

- `DiscreteMetaAction` with 5 choices (IDLE, SLOWER, FASTER, RIGHT, LEFT)
- Low level Controllers coordinate acceleration/braking profiles during merges, and curved topology.

1.3. Dataset Generation

Implementation mirrors Section 3.2 with two key adaptations:

- **Turn-Specific Validation:** Additional checks for successful merges and collision prevention added in the prompt itself [1]

1.4. Feature Selection

The same 10-feature engineering framework applies in this environment based on same intuitive and theoritical insights.

1.5. Data Analysis

Class balancing follows identical up/down-sampling strategies, with TAKE_RIGHT as the critical minority class.

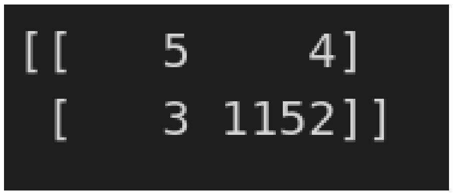
1.6. Binary RandomForest Strategy

- **First-Stage Classifier:** Binary RF separates TAKE_RIGHT (less than 1 %) from other actions
- **Second-Stage Classifier:** Multi-class RF handles remaining actions (IDLE/SLOWER/FASTER/-LEFT)

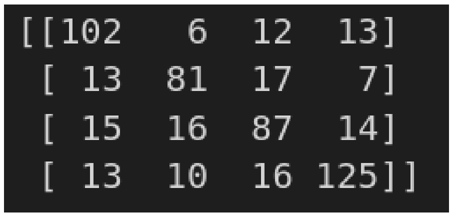
2. Results

2.1. RandomForest Model Performance

Model	Accuracy	Precision	Recall
Binary Classifier	99.40 %	62 %	56 %
Majority Class RF	72.21 %	71 %	77 %



```
[[   5    4]
 [   3 1152]]
```



```
[[102   6  12  13]
 [ 13  81  17   7]
 [ 15  16  87  14]
 [ 13  10  16 125]]
```

2.2. Performance Comparison

Metric	Claude Only	Claude with RF	DQN
Average Score per episode	30.22	18.5	22.53
Collisions/50eps	11	23	20

3. Training DQN with Claude-Based Reward Shaping

3.1. Methodology

The DQN agent is trained using a hybrid reward signal combining:

- Native environment reward r_{DQN}
- LLM alignment reward $r_{\text{LLM}} \in \{0, 1\}$ (1 if agent action matches LLM suggestion)

```
Result: Hybrid Reward Policy
1 while training do
2     Observe state $s_t$, take action $a_t$;
3     Query LLM for suggested action $a_{\text{LLM}}$;
4     Calculate hybrid reward: $r_t = 0{,}7 r_{\text{DQN}} + 0{,}3\mathbb{I}(a_t = a_{\text{LLM}})$;
5     Apply transformed reward: $R_t = \sigma(r_t) = \frac{1}{1+e^{-r_t}}$;
6     Update DQN parameters via TD-learning;
7 end
```

3.2. Key Observations

- **Halting Phenomenon:** Agent develops persistent zero-velocity states after coupleof steps for all episodes
- **Comparison Baseline:** Pure DQN recovers from halting within 5 steps
- **Claude** itself don't lead to halting situation with or without a RandomForest model fitted on the data.

3.3. Implementation Gaps

Component	Issue	Impact
Reward Scaling	Sigmoid transformation compresses r_{DQN} dynamics	Loss of gradient direction
LLM Integration	Binary alignment reward provides sparse supervision	Promotes risk-averse local optima
Action Space	No penalty for prolonged inactivity	Enables reward-hacking via halting

3.4. Possible Future Directions

- **Adaptive Reward Balancing:** Dynamic weighting of $r_{\text{DQN}}/r_{\text{LLM}}$ via:

$$\alpha_t = 1 - \frac{1}{1 + e^{-(v_t/v_{\max})}}$$

- **LLM Reward Enrichment:** Replace binary alignment with parametric safety score
- **Halting Mitigation:** Introduce velocity-dependent penalty term:

$$r_{\text{penalty}} = -\lambda \max(0, t_{\text{halt}} - \tau_{\text{safe}})$$

Software Contribution

- Developed a gymnasium based Reward shaping-LLM [9] package with documentation on:
    - Generating robust dataset for LLMs
    - Training DQN agent with Claude, groq and phi3
    - Evaluating performances of only RL, only LLM and RL integrated with LLM approaches with visual feedback.

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 "Neural Graph Control Barrier Functions Guided Distributed Collision-avoidance Multi-agent Control"(Kunal Garg, Chuchu Fan., CoRL 2023).

4. Introduction

This document outlines the implementation of the paper 'Neural Graph Control Barrier Functions Guided Distributed Collision-avoidance Multi-agent Control' in PyTorch Geometric. The method described in the paper focuses on leveraging Graph Neural Networks (GNNs) to jointly learn Graph Control Barrier Functions (GCBFs) and distributed control policies for safe and scalable multi-agent systems (MAS).

5. Problem Statement

The paper addresses the problem of distributed collision-avoidance in large-scale environments with potentially moving obstacles. It introduces **Graph Control Barrier Functions (GCBFs)** to handle collision-avoidance constraints in multi-agent systems, utilizing graph structures to ensure safety. The challenge is to develop a scalable, decentralized control system for a large number of agents operating with limited local information, while ensuring safety and goal-reaching. Mathematically the problem is formulated as follows:

Given a set of N agents with safety radius r, sensing radius R, and assigned goal locations $\{p_i^{\text{goal}} \in \mathbb{R}^n\}_{i=1}^N$ that don't overlap, design a distributed control policy $\pi_i = \pi_i(x_i, \bar{x}_i, \bar{y}_i, x_i^{\text{goal}})$ for each agent i, where:

- $\bar{x}_i$ represents the combined states of neighbors $j \in \mathcal{N}_i^o$
- $\bar{y}_i$ represents the combined sensor observations from $\mathcal{N}_i^s$

The designed policy must guarantee the following for all agents' trajectories:

- **Safety:** All agents maintain safe distances from obstacles and each other:
  - $\|y_i^j(t)\| > r, \forall j$ (agents don't hit obstacles)
  - $\|p_i(t) - p_j(t)\| > 2r$ for all $t \geq 0, j \neq i$ (agents don't collide with each other)
- **Goal Reaching:** Each agent eventually reaches its assigned destination:
  - $\text{ínf}\, \|p_i(t) - p_i^{\text{goal}}\| = 0$ (agents reach their goals)

6. Methodology

The core methodology is based on the following concepts:

6.1. Graph Structure

Each agent and obstacle is represented as a node in a graph (v i = 0 for controlled agents and v i = 1 for uncontrolled agents (i.e., the hitting points for LiDAR rays)). Agents are connected if they are within each other's sensing radius. Since the safety objective depends on the relative positions, one of the edge features is the relative position $p_{ij}(t)$.

6.2. Graph Control Barrier Functions (GCBF)

GCBF is used to ensure safety by encoding collision-avoidance constraints in the graph structure of MAS (refer Preliminaries section of this paper).

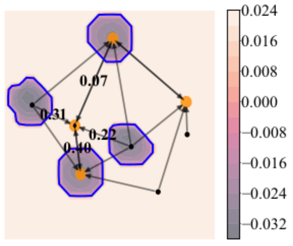


Figura 3: GCBF showing agents (orange dots) with safety regions (purple boundaries) and attention weights on edges. The color gradient represents GCBF values across the state space, with higher values (lighter colors) indicating safer regions and lower values (darker colors) indicating potentially unsafe regions.

6.3. Safe Control Policy

The proposed framework uses a **nominal controller** for the goal-reaching objective, which is a basic controller (e.g., LQR or PID).Using this nominal controller, a minimum-norm controller is designed that satisfies the GCBF based safety constraint using an optimization framework.

$$\min_{u_i \in \mathcal{U}_i} \quad \|u_i - u_i^{\text{nom}}\|^2, \tag{5}$$

$$\text{s.t.} \quad \frac{\partial h(\bar{e}_i, \bar{v}_i)}{\partial \bar{e}_i} \frac{\partial \bar{e}_i}{\partial x_i} F(x_i, u_i) + \sum_{j \in \mathcal{N}_i} \frac{\partial h(\bar{e}_i, \bar{v}_i)}{\partial \bar{e}_i} \frac{\partial \bar{e}_i}{\partial x_j} F(x_j, u_j) \geq -\alpha(h(\bar{e}_i, \bar{v}_i)), \tag{6}$$

6.4. Learning Approach

A **Graph Neural Network (GNN)** is employed to jointly learn the GCBF and the control policy. The GNN processes the graph and provides a certificate hi to ensure the safety of agents under the control policy. A **distributed controller** is trained to modify agent behaviors in a similar fashion based on the nominal controller to avoid collisions while reaching the goal.

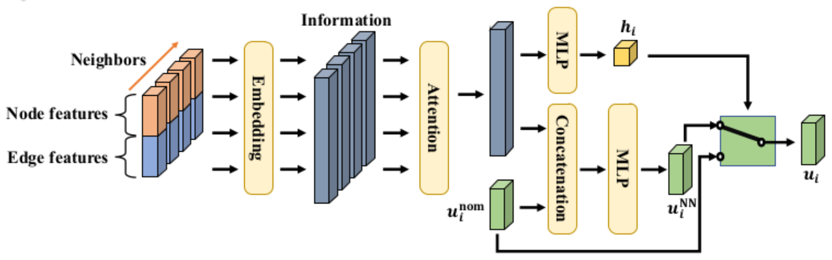


Figura 4: A neural GCBF is implemented using a graph neural network (GNN) backbone and multilayer perceptron head, encoding local agent and neighbor features. Attention mechanisms aggregate neighbor information, assigning higher importance to critical agents for safety. The distributed control policy shares this GNN structure and additionally incorporates a nominal goal-reaching policy, enabling agents to achieve goals while avoiding collisions with obstacles and other agents.

6.5. Training Process

The system is trained using a supervised approach where the loss function combines:

- **Safety loss** (ensures GCBF conditions are met):

$$\mathcal{L}_{\text{safety}} = \sum_{x_i \in \mathcal{S}_i} [\gamma - h_\theta(\bar{e}_i, \bar{v}_i) - \alpha(h_\theta(\bar{e}_i, \bar{v}_i))]^+ + \sum_{x_i \in \mathcal{S}_i} [\gamma - h_\theta(\bar{e}_i, \bar{v}_i)]^+$$

- **Unsafe loss** (penalizes unsafe states):

$$\mathcal{L}_{\text{unsafe}} = \sum_{x_i \in \mathcal{S}_{u,i}} [\gamma + h_\theta(\bar{e}_i, \bar{v}_i)]^+$$

- **Control loss** (limits deviation from nominal controller):

$$\mathcal{L}_{\text{control}} = \eta \left\| \pi_\phi(\bar{e}_i, \bar{v}_i, \pi_{\text{nom}}(x_i, x_i^{\text{goal}})) - \pi_{\text{nom}}(x_i, x_i^{\text{goal}}) \right\|$$

Control policy:

$$\mathbf{u}_i = \begin{cases} \mathbf{u}_i^{\text{nom}} & \text{if GCBF conditions are satisfied} \\ \mathbf{u}_i^{\text{learned}} & \text{otherwise (collision avoidance needed)} \end{cases}$$

6.6. Implementation Details

The neural GCBF and controller are implemented with:

- A GNN backbone using attention-weighted aggregation of edge/node features
- Separate MLP heads for safety certification (GCBF) and control generation
- Integration of nominal control signals through feature concatenation
- Composite loss function enforcing safety, obstacle avoidance, and control smoothness

Algorithm 1: Neural GCBF & Controller Training

```
1  Input: Graph data $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node features $v_i$, edge features $e_{ij}$
2  Parameter: GCBF network $h_\theta$, controller $\pi_\phi$, learning rate $\eta$
3  for $epoch = 1$ to $N_{epochs}$ do
4      for each agent $i \in \mathcal{V}$ do
5          # Encode edge features with attention
6          $q_{ij} \leftarrow f_{\theta_1}(e_{ij}, v_j) \quad \forall j \in \mathcal{N}_i$
7          $\alpha_{ij} \leftarrow \text{softmax}(f_{\theta_2}(q_{ij}))$                // Attention weights
8          $q_i \leftarrow \sum_{j \in \mathcal{N}_i} \alpha_{ij} \cdot f_{\theta_3}(q_{ij})$
9          # Compute GCBF value and control
10         $h_i \leftarrow f_{\theta_4}(q_i)$                // Safety certificate
11         $u_i^{\text{nom}} \leftarrow \pi^{\text{nom}}(x_i, x_i^{\text{goal}})$
12         $u_i^{\text{nn}} \leftarrow f_\phi([q_i, u_i^{\text{nom}}])$                // Controller output
13         # Switching logic
14         if GCBF conditions satisfied then
15             $u_i \leftarrow u_i^{\text{nom}}$
16         else
17             $u_i \leftarrow u_i^{\text{nn}}$
18         end
19     end
20     # Loss computation
21     $\mathcal{L}_{\text{safety}} \leftarrow \sum[\gamma - h_i - \alpha(h_i)]_+ + [\gamma - h_i]_+$
22     $\mathcal{L}_{\text{unsafe}} \leftarrow \sum[\gamma + h_i]_+$
23     $\mathcal{L}_{\text{control}} \leftarrow \eta \| u_i^{\text{nn}} - u_i^{\text{nom}} \|$
24     $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{safety}} + \mathcal{L}_{\text{unsafe}} + \mathcal{L}_{\text{control}}$
25     # Parameter update
26     $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{total}}$
27     $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}_{\text{total}}$
28 end
```

Algorithm 2: GNN Forward Pass (Detailed)

```
1  for each node $i$ do
2      $\mathcal{N}_i \leftarrow \text{GetNeighbors}(i, \mathcal{E})$
3      if $\mathcal{N}_i = \emptyset$ then
4          $q_i \leftarrow 0$
5      end
6      foreach $j \in \mathcal{N}_i$ do
7          $e_{ij} \leftarrow \text{GetEdgeFeatures}(i, j)$
8          $q_{ij} \leftarrow \text{MLP}_{\theta_1}([e_{ij}, v_j])$
9          $\alpha_{ij} \leftarrow \text{softmax}(\text{MLP}_{\theta_2}(q_{ij}))$
10         $w_{ij} \leftarrow \alpha_{ij} \cdot \text{MLP}_{\theta_3}(q_{ij})$
11     end
12     $q_i \leftarrow \sum_j w_{ij}$
13     $h_i \leftarrow \text{MLP}_{\theta_4}(q_i)$
14 end
```

7. Conclusion

This report summarizes the implementation steps for a distributed collision-avoidance multi-agent control system using **Graph Control Barrier Functions** (GCBFs) and **Graph Neural Networks** (GNNs). The method is scalable, decentralized, and ensures safety in large-scale multi-agent systems due to GNN'scapability to handle variable inputs and changing graph structures. We have outlined the necessary steps to implement this method using **PyTorch Geometric** and provided the associated pseudocode for reference.

References

1. https://github.com/prachitgupta/Reward-shaping-with-LLMS/blob/main/papers/btp.pdf

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. A Gentle Introduction to Graph Neural Networks

9. Reward Shaping LLM - https://github.com/prachitgupta/Reward-shaping-with-LLMS.git

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. Neural Graph Control Barrier Functions Guided Distributed Collision-avoidance Multi-agent Control (Songyuan Zhang, Chuchu Fan, Kunal Garg)