Application of LLM Guided Reinforcement Learning in Formation Control with Collision Avoidance

Chenhao Yao, Zike Yuan, Xiaoxu Liu*, Chi Zhu

Abstract— **Multi-Agent Systems (MAS) excel at accomplishing complex objectives through the collaborative efforts of individual agents. Among the methodologies employed in MAS, Multi-Agent Reinforcement Learning (MARL) stands out as one of the most efficacious algorithms. However, when confronted with the complex objective of Formation Control with Collision Avoidance (FCCA): designing an effective reward function that facilitates swift convergence of the policy network to an optimal solution. In this paper, we introduce a novel framework that aims to overcome this challenge. By giving large language models (LLMs) on the prioritization of tasks and the observable information available to each agent, our framework generates reward functions that can be dynamically adjusted online based on evaluation outcomes by employing more advanced evaluation metrics rather than the rewards themselves. This mechanism enables the MAS to simultaneously achieve formation control and obstacle avoidance in dynamic environments with enhanced efficiency, requiring fewer iterations to reach superior performance levels. Our empirical studies, conducted in both simulation and real-world settings, validate the practicality and effectiveness of our proposed approach.**

Project Website: https://macsclab.github.io/LLM_FCCA

I. INTRODUCTION

Multi-Agent Systems (MAS) have demonstrated superior performance across various fields, showcasing higher task efficiency and stronger fault tolerance compared to single-agent systems. However, current MAS applications predominantly operate within structured environments, with less satisfactory performance in more complex, unstructured scenarios. Traditional methods, such as Optimal Reciprocal Collision Avoidance (ORCA) [1] and Artificial Potential Fields (APF) [2] , explicitly model the system and precisely calculate the instructions for each agent at every moment. However, these approaches often rely on certain assumptions (e.g., ORCA assumes that all agents follow the same obstacle avoidance strategy), which can lead to policy failures when real-world deployments do not align with these preconditions.

In recent years, Multi-Agent Reinforcement Learning (MARL), a type of reinforcement learning (RL) has made notable advancements in addressing challenges such as formation control, obstacle avoidance, and maintaining system stability within MAS, showcasing its substantial potential. Through continuous interaction with their environment, agents iteratively refine their policies to maximize cumulative rewards, highlighting the significant impact that the design of reward function can have on policy quality. While crafting an effective reward function is relatively straightforward for single-task scenarios, the Formation Control with Collision Avoidance (FCCA) problem introduces the complexity of accounting for interdependencies between different tasks. This added layer of complexity often necessitates considerable time and effort in designing and tuning reward functions; even minor adjustments, such as tweaking the weights of individual reward components, can require retraining of the models based on the original models to ensure optimal performance across multiple objectives.

To address the above issues, we propose a framework where agent observations are provided to an LLM, which generates an initial reward function focused on core objectives rather than optimal performance across all tasks. Instead of relying on reward magnitude, effectiveness is evaluated by how well predefined performance criteria are met. After a fixed number of iterations, these criteria and task-specific rewards are fed back to the LLM, enabling online adjustments to improve the reward function. We validate our method in a complex scenario where a MAS maintains formation, avoids dynamic obstacles, and reaches its destination in minimal time with stable actions. An overview of the proposed method is shown in Fig. 1.

In summary, the contributions of this work can be highlighted as follows:

- We are the first to apply LLM-guided RL to the multi-agent FCCA task, enabling the creation of sophisticated reward structures that guide agents in achieving complex objectives.
- We implemented a framework that dynamically updates reward functions, allowing continuous improvement and higher efficiency with fewer iterations.
- We validated the effectiveness and practicality of our approach through the use of sim-to-sim and sim-to-real validation methods.

II. RELATED WORKS

A. Multi-Agent Reinforcement Learning

With the proposal and refinement of various effective RL algorithms, such as Proximal Policy Optimization (PPO) [3] , Importance Weighted Actor-Learner Architecture(IMPALA) [4], and QMIX [5], MARL has made significant progress. These approaches employ neural networks to approximate the policy of an agent rather than modeling the entire task as an optimization problem.

This work was supported by National Natural Science Foundation of China under Grant 62003218, Stable Support Projects for Shenzhen Higher Education Institutions under Grant 20220717223051001.

The authors are with the School of Sino-German College of Intelligent Manufacturing, Shenzhen Technology University, Shenzhen 518118, China. Emails: yaochenhao@sztu.edu.cn; yuanzike2022@email.szu.edu.cn; liuxiaoxu@sztu.edu.cn; zhuchi@sztu.edu.cn.

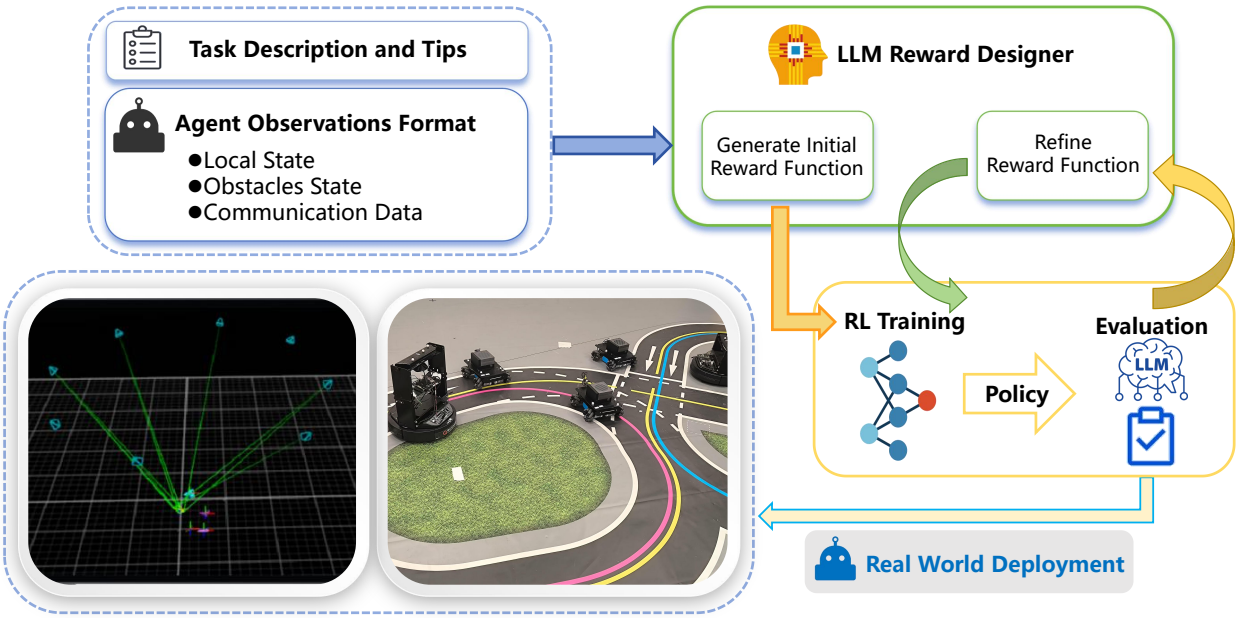


Fig. 1: Overview of our method

Initially, MARL was primarily applied in video games, like StarCraft II [6] and DOTA II [7]. Furthermore, in [8] , the researchers conducted a series of extensive experiments utilizing the Proximal Policy Optimization (PPO) algorithm across a multitude of distinct cooperative multi-agent benchmarks. The experimental results consistently demonstrated superior performance, with the level of coordination among the agents even exceeding that of human teams. However, these achievements were not substantiated by real-world deployments, leaving questions about the efficacy of these RL algorithms in real-world robotics.

Utilizing a hybrid approach that integrates imitation learning (IL) and RL, the researchers [9] trained MAS for tasks FCCA, resulting in markedly improved performance in maintaining formations and avoiding collisions compared to earlier methods, while also demonstrating that models trained with MARL methodologies possess robust generalization capabilities and achieved satisfactory performance in real-world experiments. However, the effectiveness of these systems remains constrained by the leader-follower architecture, as the system can become unstable if the leader makes incorrect decisions or fails to effectively transmit decision information to the followers, potentially leading to a loss of control over the entire system.

By adopting a distributed architecture in [10], the researchers facilitated independent decision-making for each agent, thus mitigating the risk of system instability due to a leader's failure. Curriculum learning was utilized to streamline the training process, incrementally raising the complexity of objectives faced by agents. However, the real-world experimental scenarios were relatively simplistic, featuring only sparse static obstacles, which may not sufficiently demonstrate the superiority of MARL approaches. Moreover, individual reward components converged more slowly at the beginning due to the necessity of learning multiple objectives concurrently, raising concerns that in more complex environments, the system might become trapped in local optima, thereby reducing success rates. In the context of tackling complex objectives, it becomes clear that the design of reward functions is one of the most crucial elements in MARL.

B. LLM for Robotics

The intersection of LLM with various disciplines represents an emerging and popular frontier in research; at present, LLM have already acquired substantial semantic knowledge and exhibit formidable text generation capabilities. Within the framework described [11], LLM have been employed to generate a series of high-level action sequences as instructions, and to pre-train a suite of low-level skills through RL. Through these high-level action sequences, LLM guide agents in selecting the appropriate low-level skills to perform corresponding natural language actions, thereby achieving real-world grounding. Based on this, the CodeAct method [12] allows for online adjustment of actions based on observed information within the framework of SayCan. Meanwhile, [13] breaks down a navigation command into multiple sub-commands, enabling the intelligent agent to learn navigation strategies that adapt to environmental changes. However, these methodologies necessitate additional learning for low-level actions and cannot be directly deployed on agents.

On the other hand, [14] and [15] directly utilizes the LLM to generate the reward functions required for RL, and employs these reward functions to learn low-level actions. The results from validation across several challenging tasks indicate that models trained using this method achieve per-

formance that is comparable to or surpasses that of human experts in executing complex objectives. However, generating reward functions in a one-time process for training and simply feeding back the magnitude of rewards obtained from LLM-generated reward functions after environmental evaluation to the LLM like the he aforementioned works lead to tendency to get trapped in local optima when dealing with complex objectives, i.e., the agents focus on completing only a single task while neglecting others.

In extracting the advantages of the aforementioned approaches, we have established high-level evaluation metrics for each task and permitted the LLM-generated reward functions to initially focus on critical or more challenging-to-converge tasks. Following this initial phase, the reward functions are progressively fine-tuned to address secondary objectives. This method ultimately enables superior performance across all tasks.

III. PRELIMINARIES

A. Formation Control with Collision Avoidance

FCCA is a classic multi-agent coordination problem with applications spanning ground-based robots[16], aerial drones[17], and surface vehicles[18]. In the absence of obstacles, multiple agents need to obtain positional information from each other to maintain a good formation and collectively move toward the goal. When obstacles appear, one or more agents must temporarily disrupt the formation to avoid the obstacles, creating a conflict between "maintaining formation" and "avoiding obstacles." After leaving the obstacle area, there is also a conflict between "reaching the goal" and "maintaining formation." For static obstacles, the issue can be explicitly formulated as an optimization problem through pre-planned paths; however, for dynamic obstacles, especially in unstructured environments unlike roadways, real-time path planning becomes significantly challenging. To address the FCCA problem with dynamic obstacles, we have opted for an RL-based approach.

B. Reinforcement Learning Modeling

RL is typically formalized theoretically as a Markov Decision Process (MDP) [19]: $(\mathcal{S}, \mathcal{A}, P, R, \mathcal{O})$, which means that each agent selects actions $\mathcal{A}$ based on its own observations $\mathcal{O}$ to maximize the reward R.The other variables ($\mathcal{S}$ represent the global state, which include the observations of all agents, and P is the state transition function. For each agent i, our objective is to learn an individual policy $\pi_{\theta_i}(\boldsymbol{a_t}|\boldsymbol{o_t})$, where θ_i are the network parameters of the corresponding agent's policy network such that the agent can take actions $\boldsymbol{a_t}$ based on its observations $\boldsymbol{o_t}$ at the time step t. Similarly, the value function can be parameterized as: $V^i_\omega(\mathcal{S})$.

IV. METHODOLOGY

A. Agents Observations as Context

To enable the LLM to generate executable reward functions, it is necessary to provide context to the LLM. In [14] and [15], the entire environmental information is input into the LLM. However, in distributed MAS, each agent typically gathers its own local observations, observations of obstacles within its vicinity, as well as data acquired through communication with neighboring agents, a diagram is shown in Fig. 2. Specifically, for each individual agent, the available own local observations are comprised of: the coordinates of the destination g_x, g_y as well as the velocity v and orientation θ acquired from the policy network; the observations of obstacles include: the position of the obstacles in the agent's local coordinate system p^i_{ox}, p^i_{oy}, the velocity in the global coordinate system v^i_{ox}, v^i_{oy}, which are obtained through differencing positions.

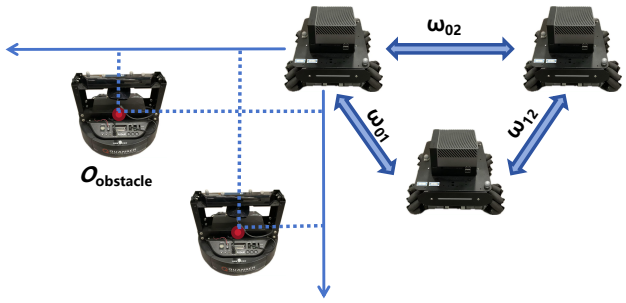


Fig. 2: Diagram of the agents observatios

The data obtained through communication with neighboring agents consist solely of the coordinates of those adjacent agents. However, designing a reward function directly from the raw coordinate information would involve mathematical derivations related to graph theory. In the absence of human feedback, LLM can understand and make efficient use of this information. Therefore, we preprocess the formation information acquired through communication according to the method outlined in [20].

Given that communication between agents in MAS is bidirectional, the entire system which includes N agents can be modeled as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where agents are represented as the set of nodes $\mathcal{V} := \{1, 2, \ldots, N\}$ and communication links between them as the set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Each edge $e_{ij} \in \mathcal{E}$ connecting nodes i and j signifies that agents i and j have the capability to measure their relative distances.Therefore, the weight of each edge can be defined as:

$$w_{ij} = \|\mathbf{p}_i - \mathbf{p}_j\|^2, \quad (i,j) \in \mathcal{E}, \tag{1}$$

where $\mathbf{p}_i$ is the position vector of the agent i, $\|.\|^2$ is Euclidean norm. Therefore, the Laplacian matrix can be derived from the weights of the edges:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}, \tag{2}$$

where $\mathbf{A}$ is the adjacency matrix and $\mathbf{D}$ is the degree matrix. Additionally, to adapt to the complex environments with dynamic obstacles, we permit the MAS to perform scaling and rotation while maintaining formation. Therefore, it is also necessary to symmetrically normalize the Laplacian

matrix:

$$\hat{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}, \tag{3}$$

Next, the formation characteristics can be obtained as:

$$f = \|\hat{\mathbf{L}} - \hat{\mathbf{L}}_{\text{des}}\|_F^2. \tag{4}$$

where $\hat{\mathbf{L}}_{\text{des}}$ is the symmetric normalized Laplacian of the desired formation and $\|.\|_F^2$ is the Frobenius norm.

Finally, the agent's own observations and the formation information are encapsulated into a class, denoted as: $\boldsymbol{o_{agent}} = [g_x, g_y, v, \theta]$; the observations of obstacles are encapsulated into another class, denoted as: $\boldsymbol{o^i_{obstacle}} = [p^i_{ox}, p^i_{oy}, v^i_{ox}, v^i_{oy}]$; these classes serve as the context input for the LLM.

B. LLM-Based Reward Function Generation

The generation and selection of the initial reward function are critical. If the LLM fails to accurately understand the task requirements when generating the initial reward function, it will be challenging to achieve results comparable to those crafted by human experts, even with multiple revisions based on feedback test results in the absence of human intervention. Firstly, it is essential to clearly define for the LLM that its role is that of a reward function designer, with the objective of generating a reward function for MARL. This reward function must address multiple objectives simultaneously:

- Avoiding dynamic obstacles within the environment.
- Maintaining the specified formation shape.
- Reaching the destination.

On the condition that the aforementioned tasks are successfully completed, it is also necessary to ensure, as much as possible, the following requirements:

- Sustaining a stable velocity to facilitate practical deployment.
- Completing the mission in the shortest possible time.

Previous methods often provide LLMs with expert-designed reward templates, which improves executability but risks inheriting design flaws. LLMs may also struggle to handle complex tasks with multiple interacting objectives, potentially omitting key elements. To address this, we start with a simple reward function focused on the most convergent objective, 'reaching the destination', and progressively incorporate other tasks after validating convergence in a basic environment. The overall process is outlined in Algorithm 1.

C. Online Tuning of Reward Functions

Upon establishing that the initial reward function is operational and can effectively achieve tasks in a simplified testing environment, the next step involves iteratively adjusting the reward function. However, judging the adequacy of the reward function by merely the magnitude of the rewards, as mentioned in [15], risks leading the LLM to amplify the reward coefficients or manipulate the reward structure, offering rewards for actions that do not authentically contribute to better completion of the task.

To this end, we have developed a high-level evaluation mechanism for the reward function, grounded in the task

Algorithm 1: LLM-Guided Reward Initialization

Input: Observation structure $\mathcal{O}$, task list $\mathcal{T}$, prompt $\mathcal{P}_0$, environment $\mathcal{E}$, threshold η
Output: Initial reward R_0, policy π_{θ_i}, value ω_i

```
Initialize P_0 with O and T;
for k = 0, 1, 2, ... do
    R_k ← LLM(P_k);
    Train π_{θ_i}^{(k)}, ω_i^{(k)} in E using R_k;
    Evaluate success rate s_k and feedback metrics M_k;
    if s_k ≥ η then
        break and return R_0 ← R_k, π_{θ_i} ← π_{θ_i}^{(k)}, ω_i ← ω_i^{(k)};
    end
    Update P_{k+1} using M_k;
end
```

objectives. First, it is crucial to explain the purpose of the function. At each time step t, the reward function computes the rewards r_t based on environmental observations. These rewards are then used alongside the predictions from the state value function $V(s_t)$ to calculate the Temporal Difference (TD) error:

$$\delta_t = r_t + \gamma V(s_{t-1}) - V(s_t). \tag{5}$$

Subsequently, by aggregating the TD errors across k time steps, we can derive :

$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta^V_{t+l}, \tag{6}$$

Combining the aforementioned equation, the Generalized Advantage Estimator(GAE) [21] can be defined as :

$$\begin{aligned} \hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} &= (1-\lambda)\sum_{l=0}^{\infty} \lambda^l \hat{A}_t^{(l+1)} \\ &= (1-\lambda)\left(\delta_t^V + \lambda \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta^V_{t+l}\right) \\ &= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta^V_{t+l}, \end{aligned} \tag{7}$$

where γ is the discount factor that reduces the weight of future rewards, and λ is another discount factor tunes the balance between immediate and long-term rewards. Here, we select $\gamma = 0.99$ and $\lambda = 0.95$.

Finally, the reward function can be defined according to the PPO algorithm [3] as:

$$\begin{aligned} \mathbf{loss}(\boldsymbol{\theta}^i) = \hat{\mathbb{E}}\Bigg[\min\Big(& r_t(\boldsymbol{\theta}^i)\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)}, \\ & \mathrm{clip}\left(r_t(\boldsymbol{\theta}^i), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} \Big) \Bigg], \end{aligned} \tag{8}$$

wherein, $r_t(\boldsymbol{\theta^i}) = \frac{\pi(a_t, o_t; \boldsymbol{\theta^i})}{\pi(a_t, o_t; \boldsymbol{\theta^i_{\text{old}}})}$ represents the probability ratio between the new policy and the old policy. The term ϵ serves as a clipping factor to restrict large deviations between the new and old policies, here $\epsilon = 0.2$. The loss function serves as a crucial indicator for policy convergence. Only when the loss function is trending towards convergence can we assert that the agents' actions are being guided by the current reward function. Subsequently, return the value of the following high-level evaluation metrics to the LLM:

- Success Rate: Reaching the destination without collisions.
- Hazard Incidents: Instances where an agent comes too close to an obstacle.
- Formation Error: Average deviation from the desired formation until reaching the destination.
- Total Time: Duration taken to complete the task.
- Average Acceleration: Mean absolute acceleration values until the destination is reached.

These metrics will assist in refining the reward function, ensuring it more accurately reflects the desired outcomes and improves the performance of the agents.

Algorithm 2: Online Tuning of Reward Function

Input: Initial reward R_0, policy π_{θ_i}, value ω_i, complex env. $\mathcal{E}$, iteration count N
Output: Updated policy π_{θ_i}, value ω_i

for $k = 1$ *to* N **do**
  Train $\pi_{\theta_i}^{(k)}, \omega_i^{(k)}$ using R_{k-1} in $\mathcal{E}$ until loss converges;
  Evaluate task metrics $\mathcal{M}_k$ (e.g., formation error, hazard, time);
  Update prompt $\mathcal{P}_k$ with $\mathcal{M}_k$ and policy summary;
  Generate new reward $R_k \leftarrow \texttt{LLM}(\mathcal{P}_k)$;
end
return final π_{θ_i}, ω_i

An objective-focused reward evaluation mechanism enables the LLM to more easily pinpoint adjustments needed for the reward function, resulting in enhanced task completion. The pseudocode is shown in Algorithm 2.

V. EXPERIMENTAL RESULTS

A. Training Result

We conducted the training in a custom simulation environment, with the specific training framework shown in Fig. 3. In the policy network, the feature values of obstacles are extracted using both a Long Short Term Memory (LSTM) network with a hidden state size of 128 and a Multilayer Perceptron (MLP) with a hidden state size of 128. The feature values of the agent are extracted through a separate MLP with a hidden state size of 128. These extracted feature values (for both obstacles and the agent) are then concatenated and passed through a linear layer activated by ReLU to produce the final output. For the value network, all states are processed directly through an MLP with a hidden state size of 256, followed by a linear layer activated by ReLU to generate the output. Each episode involves 15 iterations of the PPO algorithm to refine and update the network weights.

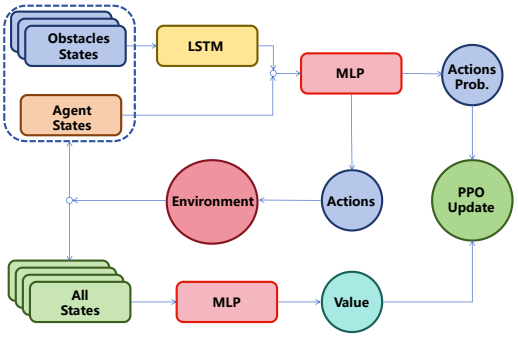


Fig. 3: Training framework by using Actor-Critic method [22] and PPO algorithm

We adopt a centralized training and decentralized execution (CTDE) framework. During training, data from all agents are aggregated, and a shared reward is used; however, each agent maintains its own individual policy network. During deployment, each agent makes decisions independently based on its own accessible observations and its respective policy network.

Qwen2.5-72B model [23] is selected as the LLM to generate the reward function. Each iteration encompasses four key steps: generating the reward function, training, evaluation, and feeding the evaluation results back into the LLM to proceed to the next iteration. All evaluation results are summarized in Table I, the formation error in Table I is computed by (4). Each iteration consists of 20 episodes, with the reported results being the average performance across these episodes.

TABLE I: Evaluation results of all iterations

Iteration	Success rate (%)	Average time (s)	Formation error
0	60	11.0	74.6
1	50	10.9	56.0
2	95	11.5	73.8
3	100	11.7	27.2

For the 0th iteration, both training and evaluation were conducted in a simplified environment featuring only three sparsely distributed dynamic obstacles. This setup aims to assess whether the LLM can comprehend the task and facilitate obtaining more substantial rewards in subsequent iterations. For all other iterations, the training and evaluation take place in a complex environment that includes seven densely distributed obstacles, combining static and dynamic elements. In this environment, both the obstacles and the agent have a maximum speed of 1.25 m/s.

Before proceeding to simulation and real-world deployment, it is important to clarify that the LLM is only involved in the reward function design during the training phase.

Once the model training and evaluation are completed, the deployment and inference of the learned policy no longer rely on the LLM. In other words, the LLM does not affect the inference efficiency of the final trained model in either simulation or real-world deployment.

Fig. 4 summarizes the conversation content with the LLM. The initial reward function generated in the 0th iteration could only accomplish the tasks of reaching the destination and avoiding obstacles, without considering the interactions among various reward components. Starting from the 1st iteration, all tasks were considered; however, the results indicated that this iteration failed to properly balance these tasks, particularly performing poorly in obstacle avoidance. Feedback from this iteration was then provided back to the LLM, leading to improvements in the obstacle avoidance reward function during the 2nd iteration. Not only was the discount factor for the obstacle avoidance reward significantly increased, but a penalty for approaching obstacles was also introduced, substantially enhancing obstacle avoidance performance, though it overlooked the reward weight for formation maintenance. In the 3rd iteration, adjustments were made to increase the reward weight for formation maintenance while simultaneously adjusting the reward weights for obstacle avoidance and reaching the destination by a certain ratio. The final evaluation results demonstrated a 100% success rate and showed excellent formation performance.

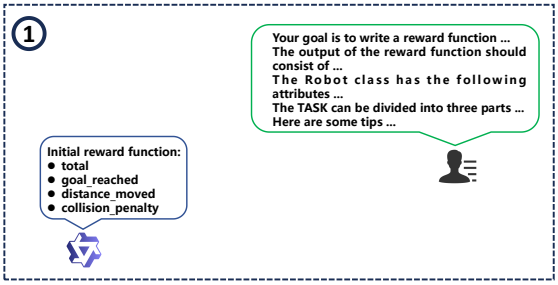
To demonstrate the effectiveness of our approach, we compared it with a human-designed method [10]. Fig. 5 illustrates the reward curves of the reward functions generated by LLM and those designed by humans during the training process, tracking the trend of reward changes from the start of training until the success rate reached 100% during evaluation. The results show that the reward function generated and fine-tuned online by the LLM enabled agents to achieve a 100% success rate more quickly in evaluations.

Specifically, in the reward curves corresponding to the LLM-generated reward functions, the LLM redesigns the reward function in each iteration based on the evaluation of the previous version. This process can lead to substantial differences in the cumulative rewards achieved in the environment between two consecutive iterations. As a result, the reward curves may exhibit significant fluctuations across iterations—for example, a drastic increase in a penalty term may cause a sudden drop in the overall reward signal.

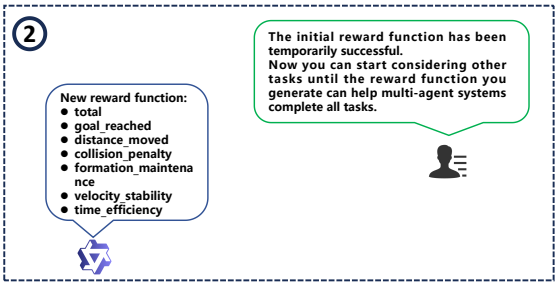
TABLE II: Comparison with other methods

Method	Success rate (%)	Average time (s)	Formation error
ORCA-F	79	19.2	35.2
human-designed	93	14.5	37.4
LLM	**95**	**11.5**	**26.7**

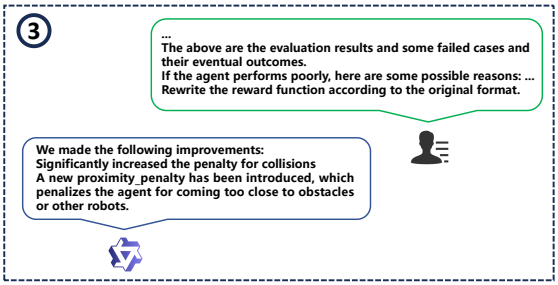
To further validate the universality of our model, we compared our approach with ORCA-F[24] and RL methods based on human-designed reward functions [10] by conducting evaluations, which use three different random seeds, each running for 300 episodes, and calculate the average performance across all runs. The results, as shown in Table II,



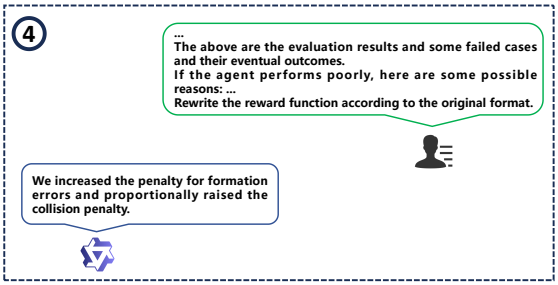
(a) iteration0



(b) iteration1



(c) iteration2



(d) iteration3

Fig. 4: Summary of the conversation with LLM.

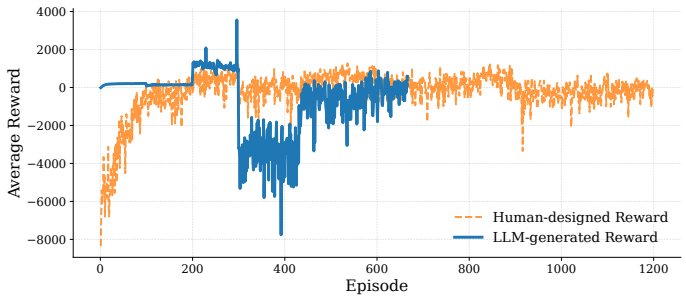


Fig. 5: LLM-Generated vs Human-Designed Reward Functions

indicate that our approach outperforms the human-designed method in terms of success rate, time consumption, and formation error.

B. Simulation and Real-World Deployment

Given the uncertainty regarding the practical applicability of the model, direct deployment in real world could potentially result in damage. Therefore, we first validated the practicality of the model through simulations. To replicate real-world conditions as closely as possible within the simulation environment, both agents and obstacles were configured with physical parameters identical to those of their real-world counterparts. Considering that multi-agent systems rely on Robot Operating System (ROS) for communication, Gazebo, which offers the best integration with ROS, was selected as the simulation platform.

The simulated environment measures 20m $\times$ 20m, with seven obstacles initially positioned within a central 5m $\times$ 5m area. The agents are Mecanum wheel robots capable of omnidirectional movement, while dynamic obstacles are represented by TurtleBot2 robots, which use differential drive wheels. Both the diagonal lengths of the agents and the diameter of the obstacles are approximately 0.35m and both have a maximum speed of 1.25 m/s.

Fig. 6 presents snapshots of critical moments during the simulation. Initially, the three agents are aligned in a straight-line formation. Upon activation, they promptly reconfigure into an equilateral triangle formation. Throughout the simulation, the agents successfully maintain their formation while navigating around obstacles and swiftly re-establish the formation immediately after completing obstacle avoidance maneuvers.

Following successful validation in simulation, we conducted real-world experiments in a 5m $\times$ 5m environment. Each agent uses the OptiTrack motion capture system to obtain its own position and that of nearby obstacles, computing velocity via positional differentiation. Both agents and obstacles are limited to a maximum speed of 0.5m/s. During deployment, each agent runs on an NVIDIA Jetson AGX Orin, with decision-making at 10Hz—invoking the policy every 0.1s, though inference takes only 4ms. A low-level PD controller handles motor speed control at 240Hz.

Fig. 7 showcases snapshots of critical moments from real-world experiments. The agents' performance mirrors that observed in the simulations: they successfully avoid all dynamic obstacles while maintaining an effective formation and reaching the destination. These experimental results demonstrate the efficacy of our approach in real-world applications.

VI. CONCLUSIONS

This paper introduces a method for RL in FCCA that leverages LLM. By generating reward functions with LLM and establishing evaluation metrics for these functions, our approach facilitates more effective online tuning of RL reward functions. When applied to the FCCA problem, our method achieves success rates that are comparable to or even surpass those of human-designed reward functions. The efficacy and practicality of this approach have been confirmed through both simulation and real-world experimentation.

Despite its advantages, this method has notable limitations. For example, reward functions generated by LLM may lose track of initial requirements and early evaluation outcomes after several iterations. When tasks become more numerous and complex, LLM face significant challenges in balancing these tasks effectively.

In future work, we plan to deepen the integration of LLM with real-world multi-agent systems. By adopting methods analogous to those outlined in [25], we aim to improve real-world performance and minimize the discrepancies between simulated and real-world environments.

REFERENCES

[1] J. van den Berg, J. Snape, S. J. Guy, and D. Manocha, "Reciprocal collision avoidance with acceleration-velocity obstacles," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3475–3482, 2011.

[2] Z. Pan, C. Zhang, Y. Xia, H. Xiong, and X. Shao, "An improved artificial potential field method for path planning and formation control of the multi-uav systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 3, pp. 1129–1133, 2022.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[4] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

[5] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," *Journal of Machine Learning Research*, vol. 21, no. 178, pp. 1–51, 2020.

[6] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[7] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[8] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. WU, "The surprising effectiveness of ppo in cooperative multi-agent games," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, pp. 24 611–24 624. Curran Associates, Inc., 2022.

[9] Z. Sui, Z. Pu, J. Yi, and S. Wu, "Formation control with collision avoidance through deep reinforcement learning using model-guided demonstration," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 6, pp. 2358–2372, 2021.

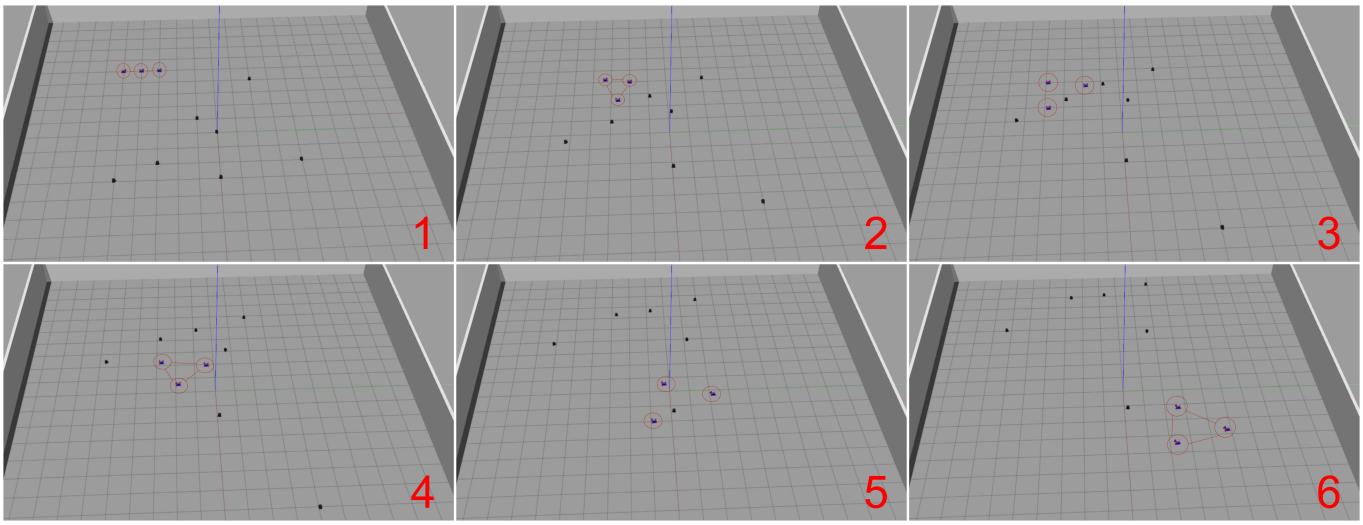


Fig. 6: Snapshots of simulation

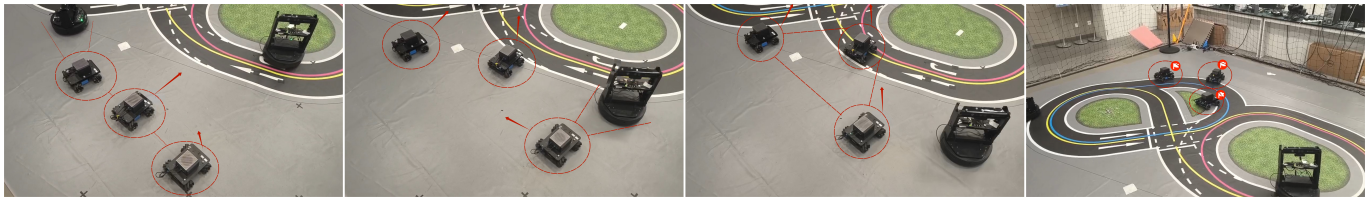


Fig. 7: Snapshots of real-world deployment

[10] Y. Yan, X. Li, X. Qiu, J. Qiu, J. Wang, Y. Wang, and Y. Shen, "Relative distributed formation and obstacle avoidance with multi-agent reinforcement learning," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 1661–1667, 2022.

[11] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[12] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji, "Executable code actions elicit better LLM agents," in *Forty-first International Conference on Machine Learning*, 2024.

[13] J. Wang, T. Wang, W. Cai, L. Xu, and C. Sun, "Boosting efficient reinforcement learning for vision-and-language navigation with open-sourced llm," *IEEE Robotics and Automation Letters*, vol. 10, no. 1, pp. 612–619, 2025.

[14] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu, "Text2Reward: Reward Shaping with Language Models for Reinforcement Learning," *arXiv e-prints*, 2023.

[15] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar, "Eureka: Human-level reward design via coding large language models," *arXiv preprint arXiv:2310.12931*, 2023.

[16] S.-L. Dai, K. Lu, and J. Fu, "Adaptive finite-time tracking control of nonholonomic multirobot formation systems with limited field-of-view sensors," *IEEE Transactions on Cybernetics*, vol. 52, no. 10, pp. 10 695–10 708, Oct. 2022.

[17] B. Zhou, H. Xu, and S. Shen, "Racer: Rapid collaborative exploration with a decentralized multi-uav system," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1816–1835, 2023.

[18] Z. Peng, J. Wang, D. Wang, and Q.-L. Han, "An overview of recent advances in coordinated control of multiple autonomous surface vehicles," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 2, pp. 732–745, Feb. 2021.

[19] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[20] L. Quan, L. Yin, C. Xu, and F. Gao, "Distributed swarm trajectory optimization for formation flight in dense environments," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 4979–4985, 2022.

[21] S. L. M. J. P. A. John Schulman, Philipp Moritz, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[22] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv e-prints*, 2016.

[23] A. Yang *et al.*, "Qwen2.5 Technical Report," *arXiv e-prints*, 2024.

[24] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 285–292, 2017.

[25] Y. J. Ma, W. Liang, H.-J. Wang, S. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman, "Dreureka: Language model guided sim-to-real transfer," *arXiv preprint arXiv:2406.01967*, 2024.