ReMAC: Large Language Model-Driven Reward Design for Multi-Agent Manipulation Collaboration

Pengyi Li
College of Intelligence and Computing
Tianjin University
lipengyi@tju.edu.cn

Hongyao Tang
College of Intelligence and Computing
Tianjin University
tanghongyao@tju.edu.cn

Yifu Yuan
College of Intelligence and Computing
Tianjin University
yuanyf@tju.edu.cn

Jianye Hao†
College of Intelligence and Computing
Tianjin University
jianye.hao@tju.edu.cn

Abstract

Multi-agent collaboration, such as in multi-robot systems, often relies on carefully crafted reward functions. These functions are crucial for learning collaborative policies. However, designing efficient reward functions for multi-agent systems remains an open challenge. To bridge this gap, we propose **ReMAC**, a novel large language model-driven **Re**ward generation framework for **M**ulti-**A**gent **C**ollaboration. ReMAC employs a hierarchical approach to generate and optimize multi-agent reward functions: The upper level maintains and iteratively optimizes a population of reward functions from both team-level and individual-agent perspectives. The lower level applies multi-agent reinforcement learning algorithms (MARL) to learn collaborative policies. This hierarchical design ensures efficient learning and optimization of multi-agent policies. Motivated by recent advances in robotics, especially in embodied AI, we observe that existing multi-agent benchmarks fall short in supporting collaborative manipulation tasks. To bridge this gap, we design the Multi-Agent Manipulation Collaboration benchmark, **ManiCraft**, aiming to advance research on robotic manipulation in the MARL community. Experimental results demonstrate that ReMAC successfully constructs high-quality reward functions that outperform even those manually designed by human experts. The visualization videos are available at https://remac-manicraft.github.io.

1 Introduction

Collaborative manipulation with multiple robotic arms requires precise coordination [1–5]. When multiple robots jointly lift, place, or assemble objects, their individual actions become tightly coupled. Even small mismatches in timing or force can lead to internal wrench buildup or coordination failure. Learning such coordinated behavior automatically is difficult because each robot receives only partial observations and must align its motion with others [6–8].

Multi-Agent Reinforcement Learning (MARL) [9–16] provides a natural framework for learning these complex collaborative behaviors [17–19]. In this paradigm, each robot is modeled as an agent that interacts not only with the environment but also with its partners, and all agents improve their policies through interactive experience [20]. However, learning effective collaborative behaviors in

†Corresponding authors: Jianye Hao

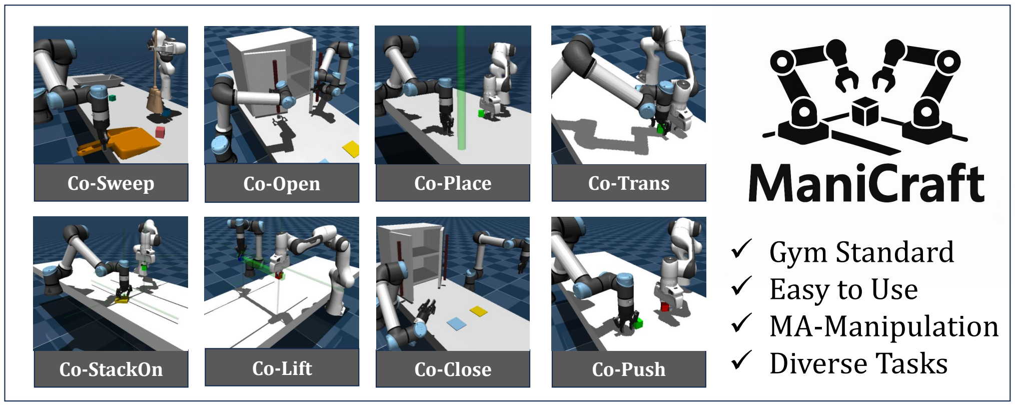

Figure 1: ManiCraft: A Multi-Agent Manipulation Benchmark for Collaborative Policy Learning.

MARL remains a persistent challenge [21–25], and one critical reason is the design of the reward function. Sparse team-level rewards often lead to unstable learning, while individual per-agent rewards can incentivize selfish or even conflicting behaviors [26]. Designing high-quality reward functions often requires expert involvement, making the process time-consuming and challenging. Consequently, reward design remains one of the central challenges in MARL [27].

Early works typically focus on improving existing reward functions via reward shaping [28–31] or credit assignment [32–35]. Designing reward functions from scratch remains largely underexplored. Recently, Large language models (LLMs) have gained significant attention, demonstrating human-level performance in areas such as code generation, planning, and reasoning [36–39]. Some works, such as L2R [40], Eureka [41], and R* [42] employ LLMs to design rewards from scratch for single-agent problems, while others like CriticGPT [43] and RLAIF [44] leverage LLMs to provide preference for training reward models. However, research on reward design for multi-agent settings remains limited [45–48]. Unlike single-agent scenarios, multi-agent tasks require jointly defining both individual- and team-level rewards, which greatly increases the complexity of reward formulation.

To bridge this gap, we propose a novel LLM-driven framework for **Re**ward generation in **M**ulti-**A**gent **C**ollaboration (ReMAC). ReMAC leverages the domain knowledge and coding capabilities of LLMs to generate structured reward functions for MARL. Specifically, the LLM first analyzes the individual skills required by each agent and the coordination demands at the team level. Based on this analysis, ReMAC constructs two types of rewards: *agent-level* and *team-level*, which are then combined to produce the final reward for each agent. To improve reward quality, we maintain a reward population $\mathbb{P}_{\mathrm{R}}$, where each individual comprises both agent- and team-level reward functions. For every reward function, we instantiate a corresponding MARL agent, forming a MARL population $\mathbb{P}_{\mathrm{MARL}}$. The team policies in $\mathbb{P}_{\mathrm{MARL}}$ interact with the environment to generate experiences. Each experience is then labeled with rewards by the reward population and stored in a shared replay buffer for learning. At regular intervals, the best-performing team is summarized and fed back to the LLM, which reflects on the design from skill, individual, and team perspectives. Based on this reflection, the LLM generates improved reward functions to replace the suboptimal ones in $\mathbb{P}_{\mathrm{R}}$, which are then used for subsequent policy training.

To evaluate the effect of reward design on low-level collaboration, a dedicated benchmark for multi-agent manipulation is essential. However, the current MARL community lacks such a benchmark for low-level policy learning and evaluation. Existing suites, such as MA-MuJoCo [49], focus on locomotion coordination, while RoCo [47] emphasizes high-level scheduling and planning. To fill this gap, we introduce ManiCraft, a new and challenging benchmark for collaborative robotic manipulation. ManiCraft is built on MuJoCo with mocap for end-effector control and includes 11 manipulation tasks of varying difficulty. We carefully design the observation space, action space, and reward structure to ensure that each task can be effectively learned by current MARL algorithms. ManiCraft follows the Gym interface standard, providing a lightweight and reproducible environment for development and evaluation. We evaluate ReMAC on ManiCraft, and the results show that the reward functions generated by ReMAC consistently outperform strong human-designed baselines.

Our contributions are summarized as follows: 1) We propose an LLM-driven reward generation framework, ReMAC, which efficiently designs multi-agent reward functions that are competitive with or better than human-designed reward functions. 2) we design a reward function population from both the agent-level and team-level perspectives, and optimizes these functions across three dimensions: skill, individual, and team, enabling efficient reward function optimization. 3) We introduce the ManiCraft benchmark to address the current lack of diverse multi-manipulation collaborative tasks in the MARL community. To the best of our knowledge, ManiCraft is the first benchmark specifically designed for collaborative manipulation tasks in MARL, with a focus on low-level policy learning.

2 Background

Multi-Agent RL: We consider a fully cooperative multi-agent task, which can be modeled as a *Decentralized Markov decision process* (Dec-MDP) [50] by a tuple: $\langle \mathcal{N}, \mathcal{S}, \mathcal{U}, \mathcal{T}, \mathcal{R}, \gamma \rangle$. Here, $\mathcal{N} = \{1, \cdots, N\}$ denotes the set of N agents. In a Dec-MDP, the complete state of the environment $s_t \in \mathcal{S}$ is fully observable to the agents at each time step t. Each agent i uses a stochastic policy π_i to select actions $u_t^i \sim \pi^i(\cdot|s_t) \in \mathcal{U}^i$, resulting in a joint action $u_t = \{u_t^i\}_{i=1}^N \in \mathcal{U}$. After executing the joint action u_t in state s_t, the environment transitions to the next state s_{t+1} according to the transition function $\mathcal{T}(s_t, u_t)$, and the policies receive the reward(s) r_t from the reward function $\mathcal{R}(s_t, u_t)$, $\gamma \in [0, 1)$ is a discount factor. We denote the joint policy as $\pi = \{\pi^1, \cdots, \pi^N\} \in \Pi$, where Π is the joint policy space. In cooperative MARL, the collaborative team aims to find a joint policy that maximizes the total expected discounted return, denoted as $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$. MARL algorithms vary, with some focusing on communication [51–53, 52, 54], non-stationarity [55–58], and credit assignment [59, 33, 34, 60], diversity [61–64, 58], exploration [65–67, 58], and convergence properties [68–70]. In this paper, we aim to design reward functions in code form that guides MARL to learn the collaborative policies.

Reward Design and Shaping: Various methods have been proposed to construct high-quality reward signals. In Inverse RL, reward functions are learned from expert demonstrations [71–76]. Preference-based methods [77–82] leverage human feedback and preference data to guide the learning process. Additionally, methods like trial-and-error manual design [83, 84] and evolutionary algorithms [85, 86] optimize reward functions using predefined templates, relying on domain knowledge from experts. Some works employ LLMs to generate reward function code. L2R [40] generates reward functions based on task descriptions. Eureka [41] uses reward function population for iterative improvement. R* [42] decomposes reward design into structural evolution and parameter optimization. Other works focus on reward shaping [87, 28, 29, 88–90] to enhance exploration or collaboration.

LLM for Multi-Agent System: Recent works leverage LLMs to strengthen multi-agent reasoning, communication, and decision making [45, 91]. DyLAN [46] dynamically dispatches LLM agents for cooperative reasoning and coding. FAMA [92] uses a centralized critic to guide agents in free-form text negotiations for optimal joint policies. Other studies show LLM agents can achieve numerical consensus through iterative dialogue [93] or employ a rudimentary Theory-of-Mind to infer teammates' hidden states and intentions [94]. γ-Bench [95] demonstrates that chain-of-thought prompting steadily improves GPT performance in cooperative games. MetaGPT [96] encodes human Standard Operating Procedures into multi-agent LLM pipelines with role-based task decomposition and cross-verification. Beyond text-only environments, LLMs are integrated into embodied frameworks: CoELA [97] combines LLM-based memory, planning, and chat channels so agents can discuss and execute household tasks; SMART-LLM [98] decomposes high-level language instructions into coalition-level robot plans; RoCo [47] equips each robot arm with an LLM-based planner for collision-free coordination. Co-NavGPT [48] dispatches a single LLM to coordinate multi-robot exploration.

3 Method

This section provides an overview of ReMAC framework. We first introduce the optimization process of ReMAC. Then, we provide a detailed description of how to construct individual and team rewards. Finally, we present how to optimize the reward functions and use them to guide policy training.

3.1 Overview

ReMAC leverages the broad domain knowledge, coding capabilities, and reasoning abilities of LLMs to design reward functions for multi-agent systems. In brief, ReMAC designs both agent-level and

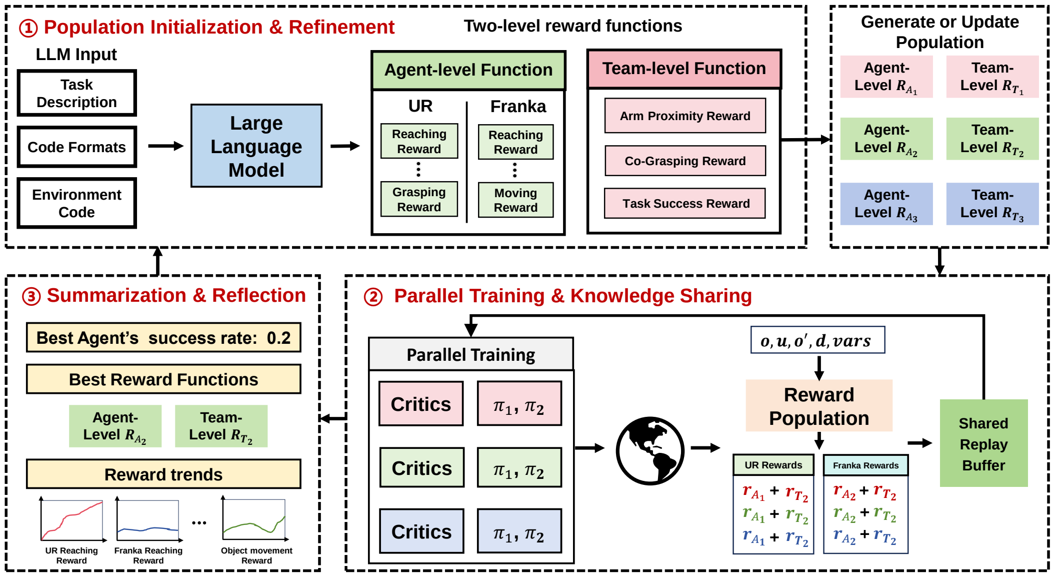

Figure 2: The overview of ReMAC. The process contains three steps: ① The task description, code templates, and environment code are taken as inputs to the LLM. The LLM generates both the agent-level reward function and the team-level reward function. Repeat this process n times to form a reward population. ② For each reward function pair, the corresponding MARL critics and policy teams are instantiated and trained through interactions with the environment. ③ At regular intervals, the reward functions associated with the best-performing policies are selected. The training details are summarized and fed back into the LLM for reflection to improve the reward functions.

team-level rewards to guide agents in learning coordinated policies. To achieve efficient optimization of reward functions, ReMAC employs an evolutionary paradigm, wherein a reward population is constructed and improved through iterative evolutionary processes. The overall optimization process of ReMAC is illustrated in Figure 2. Specifically, ReMAC consists of three key steps:

Multi-Agent Reward Population Construction. ReMAC begins by providing the LLM with the task description, environment code, and predefined prompts (e.g., reward function templates and design tips). Based on these inputs, the LLM identifies the role of each agent and generates a pair of reward functions: agent-level rewards for guiding individual skill learning and team-level rewards for promoting coordination. This process is repeated n times to construct a reward population $\mathbb{P}_\mathrm{R}$.

Parallel Training and Knowledge Sharing. For each reward function pair in $\mathbb{P}_\mathrm{R}$, we initialize a MARL instance composed of policies and critics for all agents. These instances collectively form a MARL population $\mathbb{P}_\mathrm{MA}$. Each team in $\mathbb{P}_\mathrm{MA}$ interacts with the environment to collect experiences, which are stored in a shared replay buffer $\mathcal{D}$. Each collected experience is re-labeled by its corresponding reward function and then used for MARL training and optimization.

Periodic Summarization and Reflection. At regular intervals, we select the top-performing teams and summarize their corresponding reward functions, success rates, and the trends of each reward components during train-

Algorithm 1 ReMAC Framework

```
1:  Initialization: Task description $L$, environment code $M$, coding LLM $\mathbb{LLM}$, designed prompt $p$
2:  Hyperparameters: maximum steps $T_\text{total}$, pop size $n$, evolution frequency $T_\text{evo}$
3:  Stage I: Initialize population $\mathbb{P}_\mathrm{R}$ & $\mathbb{P}_\mathrm{MARL}$:
4:  $\{\mathrm{f}^1_\text{reward}, \cdots, \mathrm{f}^n_\text{reward}\} = \mathbb{LLM}(L, M, p)$
5:  Initialize a MARL instance for each $\mathrm{f}^i_\text{reward}$
6:  for step t = 1 to $T_\text{total}$ do
7:      Stage II: Interaction & Learning
8:      Interaction and label rewards with $\mathbb{P}_\mathrm{R}$
9:      Store experiences in $D$
10:     Optimize MARL in parallel
11:     Stage III: Summarization & Reflection
12:     if $t \% T_\text{evo} == 0$ then
13:         Select $\mathrm{f}^\text{best}_\text{reward}$ with training details $d$ for reflection.
14:         Update population $\mathbb{P}_\mathrm{R} = \mathbb{LLM}(\mathrm{f}^\text{best}_\text{reward}, d, p)$
```

ing. This information is fed back to the LLM for reflection, allowing it to further refine and improve the reward population.

The above process is performed iteratively until the maximum number of interaction steps is reached. To provide a clearer illustration of the algorithm, we present its pseudocode in Algorithm 1. Next, we provide a detailed introduction of the multi-agent reward function generation, as well as policy learning and reward function optimization.

3.2 Reward Generation from Individual and Team Perspectives

Unlike single-agent problems, reward design in multi-agent systems must account for both individual skill acquisition and coordination among agents and their behaviors. This requires more fine-grained reward design and credit assignment.

To address these challenges, ReMAC leverages the strong reasoning capabilities and domain knowledge of LLMs. Given a task description $\mathcal{T}$, ReMAC proceeds in two steps: 1) Analyzes the skills each agent needs to master in order to complete the task, and constructs corresponding agent-level reward functions. 2) Identifies the necessary inter-agent constraints for coordination, and builds team-level reward functions accordingly.

We formalize both agent-level and team-level reward functions. The agent-level function outputs a list of total rewards (one for each agent) and a reward dictionary, whereas the team-level function outputs a single total reward and a corresponding dictionary. The total reward is used for policy learning, while the reward dictionary provides a basis for analyzing the trends of the reward modules. Besides, each reward function consists of multiple reward modules, each targeting a specific skill or collaboration objective. For example, a reaching reward that guides the robotic arm to a target position, or a collaborative grasping reward that encourages agents to grasp an object simultaneously. Based on the above design, the guiding reward for each agent is formulated as the sum of individual and team components, i.e. $r_i = r^i_{\text{Individual}} + r_{\text{Team}}$, facilitating more efficient credit assignment and coordinated policy learning.

3.3 Multi-Agent Policy Learning & Reward Evolution

Constructing a single pair of reward functions for policy learning is inefficient and prone to suboptimal solutions. To solve the problem, we construct n pairs of reward functions, forming a reward population $\mathbb{P}_{\text{R}}$. For each pair of reward functions in $\mathbb{P}_{\text{R}}$, a corresponding MARL instance is initialized, thereby forming a MARL population $\mathbb{P}_{\text{MARL}}$. Each team policy in $\mathbb{P}_{\text{MARL}}$ interacts with the environment to collect experiences. For each experience, rewards are computed using the reward population $\mathbb{P}_{\text{R}}$, labeling n rewards per experience. These different rewards guide the learning of different MARL individual within $\mathbb{P}_{\text{MARL}}$. This data sharing approach significantly improves sample efficiency. Additionally, considering the computational intensity of population-based training, we employ parallel training to substantially reduce time overhead.

Every T_{evo} environment steps, we select the best team and summarize its success rate, the reward functions it relies on, and the trends of each reward modules. This information is fed back to the LLM for reflection. To achieve efficient optimization of reward functions, we guide the analysis of the current reward function from three perspectives:

- **Skill perspective** focuses on analyzing whether each reward module successfully guides the skill learning, often requiring adjustments to the internal implementation of modules.
- **Individual perspective** analyzes whether the agent-level rewards are comprehensive or redundant, involving adding necessary guidance or removing unnecessary disruptive rewards.
- **Team perspective** analyzes whether the team-level reward effectively promotes collaboration among agents, which may involve optimization of the collaboration module.

Based on these three levels, LLM optimizes both the agent-level and team-level reward function by adding, removing, or adjusting reward modules. Ultimately, the reward functions generated through LLM reflection will replace non-optimal reward functions in $\mathbb{P}_{\text{R}}$. Besides, we continue optimization based on the best-performing MARL instance to avoid learning from scratch.

Table 1: Tasks included in ManiCraft and their descriptions

Task	Description
Co-Sweep-Easy	Panda holds the broom to sweep one cube into the dustpan held by UR.
Co-Sweep-Mid	Panda holds the broom to sweep two cubes into the dustpan held by UR.
Co-Sweep-Hard	Panda holds the broom to sweep three cubes into the dustpan held by UR.
Co-Push	The Panda and UR push the cubes at their sides together.
Co-Stack-On	The Panda places the cube from its side onto the coaster next to the UR.
Co-Trans	The UR grasps the cube next to the Panda.
Co-Open	Two URs work together to open the cabinet door.
Co-Close	Two URs work together to close the cabinet door.
Co-Place	Move the object next to the Panda to the target position next to the UR.
Co-Lift	UR and Panda work together to lift a rectangular object.
Co-Grasp	UR and Panda work together to grasp a rectangular object.

4 ManiCraft Benchmark

Recent advancements in the robotics field, particularly in embodied intelligence [99, 100], have been remarkable. However, we observe that the MARL community lacks a benchmark for multi-agent manipulation tasks aimed at low-level collaborative policy learning [101, 102, 19, 49, 103, 104]. To bridge the gap, We propose **ManiCraft**, a benchmark that the following key features:

- **Diverse MA-Manipulation Tasks**: A diverse set of collaborative manipulation tasks designed to facilitate low-level coordination policy learning.
- **Easy to Use & Extend**: Implemented following the Gym standard [105], with each task implemented in a single file, making it easy to use and extend.
- **Fine-grained design for MARL**: Carefully designed action space, state spaces and reward functions to ensure each task is learnable by MARL algorithms.

Specifically, ManiCraft is developed based on MuJoCo [106] and utilizes MoCap for end-effector pose control. We design 11 manipulation collaborative tasks, which typically require the coordination of a UR robotic arm and a Franka Panda robotic arm, or the coordination of two UR robotic arms. Each robot is mounted on opposite sides of a table, with the target objects to be placed on the table. Detailed task descriptions and settings are provided in Table 1. Besides, ManiCraft also supports the rapid construction of collaborative scenarios involving more than two robotic arms, and we plan to release more coordination tasks in the future. Below, we present the design of the action space, state space, and reward functions in ManiCraft.

Action Space Design. The action space of each agent is defined as a 2-tuple consisting of the end-effector's positional delta in 3D space and a normalized torque value applied by the gripper fingers. Each action is bounded within the range $[-1, 1]$. For some tasks, we extend the action space to include rotation control via Euler angles.

State Space Design. In our current setup, all agents have global observations. All task-related states are encapsulated within the observations, such as the end-effector pose of the gripper, the gripper's opening and closing size, the arm's velocity, and the position of the target object. In the future, we consider incorporating local observations to create more challenging collaborative scenarios.

Manually Designed Reward Functions. To ensure that each task can be handled by current MARL algorithms, we carefully design the reward functions and demonstrate that, for each task, the MARL algorithms can achieve a certain success rate. The basic principle behind our reward design is to construct reward components, ranging from 0 to 1, based on factors such as distance and grasping decisions, and combine them with different weights. Through extensive testing, we select the most efficient reward function configurations.

In summary, ManiCraft is a multi-agent manipulation benchmark tailored for the MARL community to facilitate the development of low-level coordination policies. We refer readers to Appendix D for additional implementation and design details of ManiCraft. The subsequent sections provide experimental validation and analysis using ManiCraft.

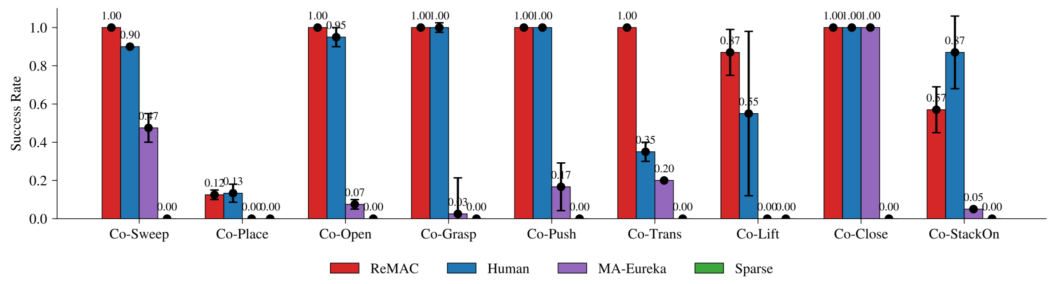

Figure 3: Performance comparison between ReMAC and other baselines on ManiCraft. ReMAC achieves performance comparable to, and in some cases surpassing, that of algorithms guided by meticulously human-designed reward functions.

5 Experiments

5.1 Experiment Setup

We conduct experiments on ManiCraft to demonstrate the effectiveness of ReMAC. At the same time, we provide videos in the link to visualize the results. The benchmark includes a diverse set of tasks, each requiring different types of collaborative policies to succeed. For a fair comparison, all algorithms adopt MASAC as the MARL backbone, and all LLM-based methods use GPT-4o as the language model. MASAC follows the centralized training with decentralized execution (CTDE) paradigm. It maintains an individual policy for each agent, along with a centralized critic to guide policy learning. The detailed network architecture and training hyperparameters are provided in Appendix B and C.

Baselines: We consider the following three baselines: 1) MASAC with human-designed reward functions, where the reward is manually crafted through trial-and-error to guide learning effectively. The detailed reward function designs are provided in Appendix D. 2) MASAC with sparse rewards, where agents receive a reward only upon successful task completion. 3) Multi-agent extension of Eureka, where we adapt the original Eureka framework to the MARL setting by using a single reward function to guide all agents collectively. Hyperparameters are kept consistent across all methods to eliminate confounding factors.

Evaluation Metric: We primarily compare the task success rate under the same number of environment steps, which reflects the sample efficiency of different algorithms. All statistics are obtained from 5 independent runs. We report the average with 95% confidence regions.

5.2 Performance Evaluation

We first compare performance across 9 different tasks in the ManiCraft benchmark. As shown in Figure 3, ReMAC demonstrates performance on par with, and in some tasks superior to, MARL algorithms using human-designed reward functions. The MARL algorithms trained with sparse rewards consistently fail to learn effective collaborative policies across all tasks. Besides, ReMAC outperforms the multi-agent extension of Eureka in both efficiency and performance. This advantage stems from two key factors: (i) the construction of reward functions from both agent-level and team-level perspectives, and (ii) the ability of ReMAC to more effectively leverage experience collected from different teams.

We present the learning curves of different algorithms in Figure 4. We can observe that ReMAC achieves sample efficiency comparable to or even better than that of manually designed reward functions. MA-Eureka is only able to learn effective collaborative policies on relatively simple tasks, such as Co-Close. When the task difficulty increases even slightly, MA-Eureka tends to fail. In contrast, ReMAC demonstrates stable learning across a broader range of tasks. However, we also observe that on certain tasks, such as Co-Place, which involve more intricate or temporally dependent coordination, both ReMAC and human-designed rewards struggle to facilitate efficient learning. This may be attributed to the challenge of decomposing complex tasks, particularly those with long-horizon dependencies and intricate inter-agent interactions. In these cases, ReMAC performs worse than

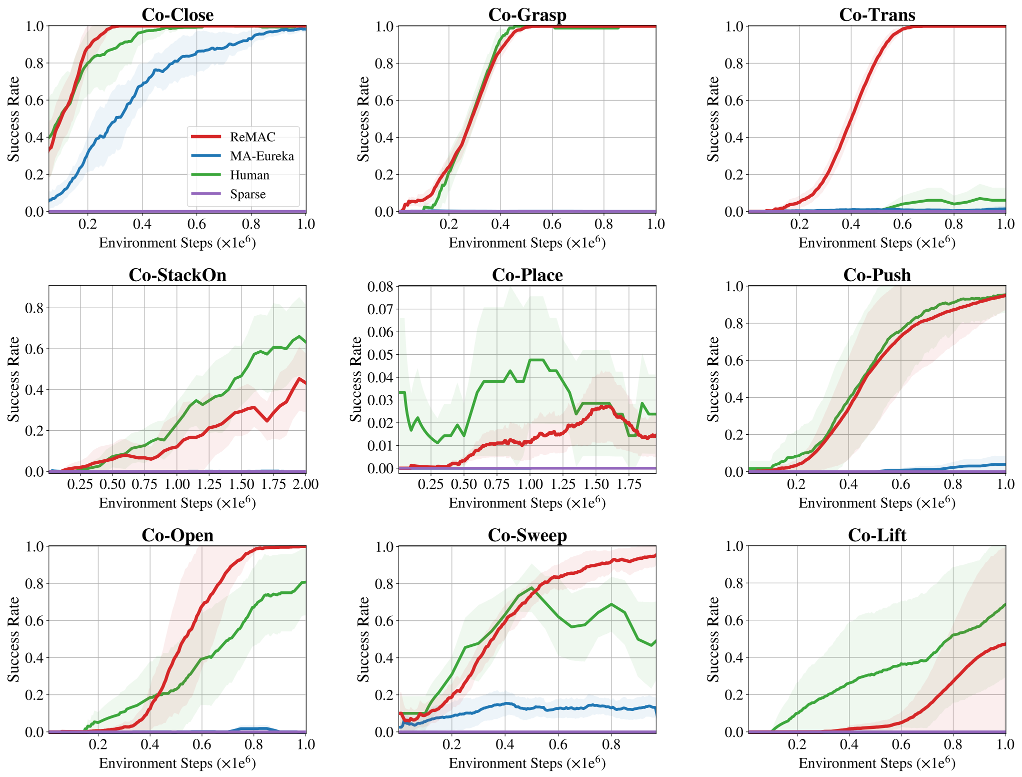

Figure 4: Training curves on various tasks. ReMAC significantly outperforms other baselines and surpasses human-designed rewards in most tasks.

human-designed rewards. One reason is that ReMAC relies on trial-and-error optimization, which leads to more frequent failures in complex tasks. These failures reduce sample efficiency and hurt final performance. Addressing above limitations will be an important direction for future work.

5.3 Ablation & Analysis

In this section, we perform ablation studies to analyze several core components of ReMAC. Specifically, we seek to answer the following three questions: Q1: Does constructing reward functions from both the agent-level and team-level perspectives lead to more effective learning? Q2: Is population-level iteration necessary for improving reward function optimization? Q3: Does experience sharing among individuals in ReMAC contribute to learning efficiency?

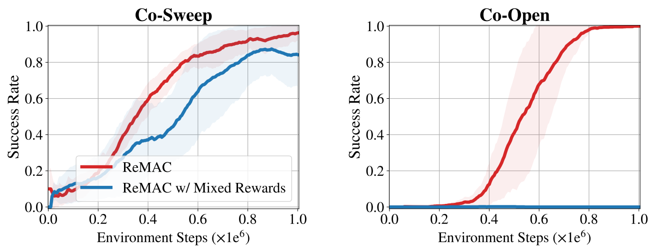

Figure 5: Ablation on Dual-Level Reward Construction

To answer Q1, we analyze the necessity of constructing reward functions from both the agent-level and team-level perspectives. We compare our approach with a unified reward design, which treats the multi-agent system as a single-agent task by constructing team rewards to guide the learning of all agents. As shown in Figure 5, ReMAC demonstrates higher efficiency compared to variants that rely on single team rewards. This advantage primarily stems from its decoupled design, which explicitly allocates credit by decomposing the reward, thereby guiding policy learning more effectively.

To answer Q2, we analyze the necessity of maintaining a reward population. As shown in Figure 6, removing the population leads to a significant performance drop in ReMAC. This is primarily because population-based optimization provides a diverse set of reward functions, which facilitates more effective exploration. In contrast, relying on iterative optimization with a single reward function tends to increase the risk of falling into suboptimal solutions. Therefore, population-based optimization is necessary for improving the efficiency of multi-agent reward function optimization.

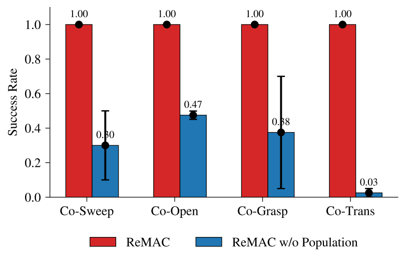

Figure 6: Ablation study on population

To answer Q3, we conduct an ablation study on knowledge sharing in ReMAC. In this setting, knowledge is no longer shared across teams, each team in the population is trained solely based on its own interaction experiences. As shown in Figure 7, ReMAC w/o Sharing has a noticeable decline in both final performance and convergence speed. Without experience sharing, the diversity of experiences available to each individual significantly decreases, which in turn hinders policy learning and optimization. Besides, we present the generated reward functions for the CoStackOn task. ReMAC decomposes the complex collaboration into meaningful components, both UR and Panda receive reaching and object moving rewards, while the team reward encourages inter-arm coordination and joint placement.

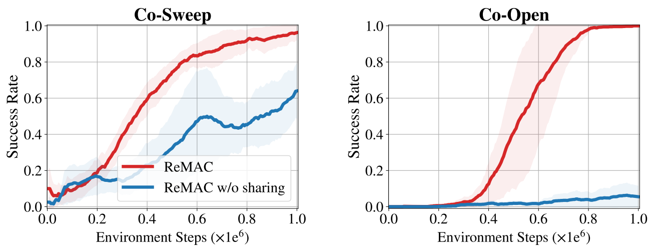

Figure 7: Ablation study on knowledge sharing

Agent-level Reward Function

```
# Calculate distances
panda_to_cube_dist, ur_to_coaster_dist,
    cube_to_coaster_dist = ...
# Panda reward components
grasp_reward = 1.0 if is_panda_grasped else 0.0
reach_reward_panda = np.exp(-panda_to_cube_dist)
move_to_goal_reward = np.exp(-cube_to_coaster_dist)
panda_reward = 0.5 * grasp_reward + 1.0 *
    reach_reward_panda + 2.0 * move_to_goal_reward
# UR reward components
reach_reward_ur = np.exp(-ur_to_coaster_dist)
ur_reward = 0.5 * reach_reward_ur + 2.0 *
    move_to_goal_reward
```

Team-level Reward Function

```
distance = np.linalg.norm(cube_position - coaster_position)
success_reward = 100.0 if success else 0.0
combined_grasp_on_coaster_reward = 1.0 if is_panda_grasped
    and is_object_on_coaster and (distance < 0.05) else 0.0
team_proximity_reward = np.exp(-distance)  # Encourage
    reducing distance to coaster
# Team-level reward
team_reward = 50.0 * combined_grasp_on_coaster_reward +
    success_reward + 1.0 * team_proximity_reward
```

6 Conclusion

To enable efficient automatic reward generation in multi-agent reinforcement learning (MARL), we propose ReMAC, a large language model (LLM)-based framework for multi-agent reward design. ReMAC adopts a two-layer architecture. The upper layer focuses on reward optimization by leveraging the LLM's broad domain knowledge and coding capabilities. It generates a population of reward candidates from both the agent-level and team-level perspectives. This population is iteratively refined based on reflective feedback from skill, individual, and team dimensions. The lower layer employs MARL to learn coordinated policies, enabling efficient experience sharing among agents. To advance research in the MARL community, we introduce ManiCraft, a benchmark suite of diverse multi-agent manipulation tasks that emphasizes low-level coordination policy learning. Experimental results on ManiCraft show that ReMAC's automatically generated rewards outperform expert-designed reward functions, significantly improving both learning efficiency and final policy performance.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant Nos. 624B2101, 62422605, 92370132), National Key Research and Development Program of China (Grant No. 2024YFE0210900), and Xiaomi Young Talents Program of Xiaomi Foundation.

References

[1] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic. Dual arm manipulation—a survey. *Robotics and Autonomous Systems*, 2012.

[2] Z. Feng, G. Hu, Y. Sun, and J. Soon. An overview of collaborative robotic manipulation in multi-robot systems. *Annual Reviews in Control*, 2020.

[3] S. Erhart and S. Hirche. Internal force analysis and load distribution for cooperative multi-robot manipulation. *IEEE Transactions on Robotics*, 2015.

[4] A. Zhai, J. Wang, H. Zhang, G. Lu, and H. Li. Adaptive robust synchronized control for cooperative robotic manipulators with uncertain base coordinate system. *ISA Transactions*, 2022.

[5] H. Haghshenas, A. Hansson, and M. Norrlöf. Time-optimal path tracking for cooperative manipulators: A convex optimization approach. *Control Engineering Practice*, 2023.

[6] M. Abbas, J. Narayan, and S. K. Dwivedy. A systematic review on cooperative dual-arm manipulators: modeling, planning, control, and vision strategies. *International Journal of Intelligent Robotics and Applications*, 2023.

[7] Z. Feng, G. Hu, Y. Sun, and J. Soon. An overview of collaborative robotic manipulation in multi-robot systems. *Annual Reviews in Control*, 2020.

[8] A. Ghorbanpour. Cooperative robot manipulators dynamical modeling and control: An overview. *Dynamics*, 2023.

[9] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint*, 2017.

[10] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, and etc. Dota 2 with large scale deep reinforcement learning. *CoRR*, 2019.

[11] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 2019.

[12] X. Lyu, A. Baisero, Y. Xiao, and C. Amato. A deeper understanding of state-based critics in multi-agent reinforcement learning. *arXiv preprint*, 2022.

[13] T. Yang, W. Wang, H. Tang, J. Hao, Z. Meng, H. Mao, D. Li, W. Liu, Y. Chen, Y. Hu, C. Fan, and C. Zhang. An efficient transfer learning framework for multiagent reinforcement learning. In *NeurIPS*, 2021.

[14] Y. Zheng, J. Hao, Z. Zhang, Z. Meng, and X. Hao. Efficient multiagent policy optimization based on weighted estimators in stochastic cooperative environments. *J. Comput. Sci. Technol.*, 2020.

[15] W. Wang, T. Yang, Y. Liu, J. Hao, X. Hao, Y. Hu, Y. Chen, C. Fan, and Y. Gao. Action semantics network: Considering the effects of actions in multiagent systems. In *ICLR*, 2020.

[16] P. Hernandez-Leal, B. Kartal, and M. E. Taylor. A survey and critique of multiagent deep reinforcement learning. *J. Auton. Agents Multi-Agent Syst.*, 2019.

[17] L. Matignon, L. Jeanpierre, and A. Mouaddib. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *AAAI*, 2012.

[18] J. Orr and A. Dutta. Multi-agent deep reinforcement learning for multi-robot applications: A survey. *Sensors*, 2023.

[19] S. Gu, J. G. Kuba, Y. Chen, Y. Du, L. Yang, A. Knoll, and Y. Yang. Safe multi-agent reinforcement learning for multi-robot control. *Artificial Intelligence*, 2023.

[20] Z. Ning and L. Xie. A survey on multi-agent reinforcement learning and its application. *Journal of Automation and Intelligence*, 2024.

[21] M. Liu, M. Zhou, W. Zhang, Y. Zhuang, J. Wang, W. Liu, and Y. Yu. Multi-agent interactions modeling with correlated policies. *arXiv preprint*, 2020.

[22] Y. Wen, Y. Yang, R. Luo, J. Wang, and W. Pan. Probabilistic recursive reasoning for multi-agent reinforcement learning. *arXiv preprint*, 2019.

[23] T. Yang, H. Tang, C. Bai, J. Liu, J. Hao, Z. Meng, and P. Liu. Exploration in deep reinforcement learning: A comprehensive survey. *CoRR*, 2021.

[24] Y. Zheng, Z. Meng, J. Hao, and Z. Zhang. Weighted double deep multiagent reinforcement learning in stochastic cooperative environments. In *PRICAI*, 2018.

[25] Y. Zheng, Z. Meng, J. Hao, Z. Zhang, T. Yang, and C. Fan. A deep bayesian policy reuse approach against non-stationary agents. In *NeurIPS*, 2018.

[26] M. Hasan and R. Niyogi. Reward specifications in collaborative multi-agent learning: a comparative study. In *SIGAPP*, 2024.

[27] D. Huh and P. Mohapatra. Multi-agent reinforcement learning: A comprehensive survey. *arXiv preprint*, 2023.

[28] L. Wang, Y. Zhang, Y. Hu, W. Wang, C. Zhang, Y. Gao, J. Hao, T. Lv, and C. Fan. Individual reward assisted multi-agent reinforcement learning. In *ICML*, 2022.

[29] Y. Du, L. Han, M. Fang, J. Liu, T. Dai, and D. Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. *NeurIPS*, 2019.

[30] N. Jaques, A. Lazaridou, E. Hughes, C. Gulcehre, P. Ortega, D. Strouse, J. Z. Leibo, and N. De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *ICML*, 2019.

[31] P. Li, H. Tang, T. Yang, X. Hao, T. Sang, Y. Zheng, J. Hao, M. E. Taylor, W. Tao, Z. Wang, et al. Pmic: Improving multi-agent reinforcement learning with progressive mutual information collaboration. *ICML*, 2022.

[32] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint*, 2017.

[33] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *ICML*, 2018.

[34] M. Zhou, Z. Liu, P. Sui, Y. Li, and Y. Y. Chung. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. *NeurIPS*, 2020.

[35] J. Li, K. Kuang, B. Wang, F. Liu, L. Chen, F. Wu, and J. Xiao. Shapley counterfactual credits for multi-agent reinforcement learning. In *KDD*, 2021.

[36] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. In *ICRA*, 2023.

[37] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, and Others. Do as I can, not as I say: Grounding language in robotic affordances. In *CORL*, 2022.

[38] Y. Wang, Z. Xian, F. Chen, T. Wang, Y. Wang, K. Fragkiadaki, Z. Erickson, D. Held, and C. Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation. In *ICML*, 2024.

[39] L. Wang, Y. Ling, Z. Yuan, M. Shridhar, C. Bao, Y. Qin, B. Wang, H. Xu, and X. Wang. Gensim: Generating robotic simulation tasks via large language models. In *ICLR*, 2024.

[40] T. Xie, S. Zhao, C. Henry Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu. Text2reward: Reward shaping with language models for reinforcement learning. In *ICLR*, 2024.

[41] Y. Jason Ma, W. Liang, G. Wang, D. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. Eureka: Human-level reward design via coding large language models. In *ICLR*, 2024.

[42] P. Li, J. Hao, H. Tang, Y. Yuan, J. Qiao, Z. Dong, and Y. Zheng. R*: Efficient reward design via reward structure evolution and parameter alignment optimization with large language models. In *ICML*, 2025.

[43] J. Liu, Y. Yuan, J. Hao, and Others. Enhancing robotic manipulation with ai feedback from multimodal large language models. *arXiv preprint*, 2024.

[44] H. Lee, S. Phatale, H. Mansoor, K. R. Lu, T. Mesnard, J. Ferret, C. Bishop, E. Hall, V. Carbune, and A. Rastogi. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. 2023.

[45] C. Sun, S. Huang, and D. Pompili. Llm-based multi-agent reinforcement learning: Current and future directions. *CoRR*, 2024.

[46] Z. Liu, Y. Zhang, P. Li, Y. Liu, and D. Yang. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *CoRR*, 2023.

[47] Z. Mandi, S. Jain, and S. Song. Roco: Dialectic multi-robot collaboration with large language models. In *ICRA*, 2024.

[48] B. Yu, H. Kasaei, and M. Cao. Co-navgpt: Multi-robot cooperative visual semantic navigation using large language models. *arXiv preprint*, 2023.

[49] B. Peng, T. Rashid, C. Schroeder de Witt, P. Kamienny, P. Torr, W. Böhmer, and S. Whiteson. Facmac: Factored multi-agent centralised policy gradients. *NeurIPS*, 2021.

[50] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Math. Oper. Res.*, 2002.

[51] J. Foerster, I. A. Assael, N. De Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *NeurIPS*, 2016.

[52] C. Zhang and V. Lesser. Coordinating multi-agent reinforcement learning with limited communication. In *AAMAS*, 2013.

[53] W. Kim, J. Park, and Y. Sung. Communication in multi-agent reinforcement learning: Intention sharing. In *ICLR*, 2020.

[54] C. Zhu, M. Dastani, and S. Wang. A survey of multi-agent deep reinforcement learning with communication. *AAMAS*, 2024.

[55] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht. Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv preprint*, 2019.

[56] H. Nekoei, A. Badrinaaraayanan, A. Sinha, M. Amini, J. Rajendran, A. Mahajan, and S. Chandar. Dealing with non-stationarity in decentralized cooperative multi-agent deep reinforcement learning via multi-timescale learning. In *Conference on Lifelong Learning Agents*, 2023.

[57] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. De Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint*, 2017.

[58] P. Li, J. Hao, H. Tang, Y. Zheng, and X. Fu. Race: improve multi-agent reinforcement learning with representation asymmetry and collaborative evolution. In *ICML*, 2023.

[59] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint*, 2017.

[60] A. Wong, T. Bäck, A. V. Kononova, and A. Plaat. Deep multiagent reinforcement learning: Challenges and directions. *Artificial Intelligence Review*, 2023.

[61] C. Li, T. Wang, C. Wu, Q. Zhao, J. Yang, and C. Zhang. Celebrating diversity in shared multi-agent reinforcement learning. *NeurIPS*, 2021.

[62] M. Bettini, R. Kortvelesy, and A. Prorok. Controlling behavioral diversity in multi-agent reinforcement learning. *arXiv preprint*, 2024.

[63] Z. Liu, C. Yu, Y. Yang, Z. Wu, Y. Li, et al. A unified diversity measure for multiagent reinforcement learning. *NeurIPS*, 2022.

[64] S. Hu, C. Xie, X. Liang, and X. Chang. Policy diagnosis via measuring role diversity in cooperative multi-agent rl. In *ICML*, 2022.

[65] I. Liu, U. Jain, R. A. Yeh, and A. Schwing. Cooperative exploration for multi-agent deep reinforcement learning. In *ICML*, 2021.

[66] L. Zheng, J. Chen, J. Wang, J. He, Y. Hu, Y. Chen, C. Fan, Y. Gao, and C. Zhang. Episodic multi-agent reinforcement learning with curiosity-driven exploration. *NeurIPS*, 2021.

[67] J. Hao, T. Yang, H. Tang, C. Bai, J. Liu, Z. Meng, P. Liu, and Z. Wang. Exploration in deep reinforcement learning: From single-agent to multiagent domain. *TNNLS*, 2023.

[68] J. G. Kuba, R. Chen, M. Wen, Y. Wen, F. Sun, J. Wang, and Y. Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint*, 2021.

[69] Y. Zhong, J. G. Kuba, X. Feng, S. Hu, J. Ji, and Y. Yang. Heterogeneous-agent reinforcement learning. *JMLR*, 2024.

[70] 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. *NeurIPS*, 2022.

[71] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

[72] D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. In *IJCAI*, 2007.

[73] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, 2008.

[74] J. Ho and S. Ermon. Generative adversarial imitation learning. In *NeurIPS*, 2016.

[75] L. Yu, J. Song, and S. Ermon. Multi-agent adversarial inverse reinforcement learning. In *ICML*, 2019.

[76] S. Liu and M. Zhu. Distributed inverse constrained reinforcement learning for multi-agent systems. *NeurIPS*, 2022.

[77] T. Kaufmann, P. Weng, V. Bengs, and E. Hüllermeier. A survey of reinforcement learning from human feedback. *arXiv preprint*, 2023.

[78] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *NeurIPS*, 2017.

[79] K. Lee, L. M. Smith, and P. Abbeel. Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training. In *ICML*, 2021.

[80] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox. Correcting robot plans with natural language feedback. In *RSS*, 2022.

[81] L. Guan, K. Valmeekam, and S. Kambhampati. Relative behavioral attributes: Filling the gap between symbolic goal specification and reward learning from human preferences. In *ICLR*, 2023.

[82] S. Kang, Y. Lee, M. Kim, J. Oh, S. Chong, and S. Yun. Dpm: Dual preferences-based multi-agent reinforcement learning. 2024.

[83] S. Booth, W. Bradley Knox, J. Shah, S. Niekum, P. Stone, and A. Allievi. The perils of trial-and-error reward design: Misdesign through overfitting and invalid task specifications. In *AAAI*, 2023.

[84] W. Bradley Knox, A. Allievi, H. Banzhaf, F. Schmitt, and P. Stone. Reward (mis)design for autonomous driving. *Artif. Intell.*, 2023.

[85] A. Faust, A. G. Francis, and D. Mehta. Evolving rewards to automate reinforcement learning. *arXiv preprint*, 2019.

[86] S. Niekum, A. G. Barto, and L. Spector. Genetic programming for reward function search. *IEEE Trans. Auton. Ment. Dev.*, 2010.

[87] P. Ladosz, L. Weng, M. Kim, and H. Oh. Exploration in deep reinforcement learning: A survey. *Inf. Fusion*, 2022.

[88] J. Hu, Y. Sun, H. Chen, S. Huang, Y. Chang, L. Sun, et al. Distributional reward estimation for effective multi-agent deep reinforcement learning. *NeurIPS*, 2022.

[89] B. Liu, Z. Pu, Y. Pan, J. Yi, Y. Liang, and D. Zhang. Lazy agents: A new perspective on solving sparse reward problem in multi-agent reinforcement learning. In *ICML*, 2023.

[90] D. E. Hostallero, D. Kim, S. Moon, K. Son, W. J. Kang, and Y. Yi. Inducing cooperation through reward reshaping based on peer evaluations in deep multi-agent reinforcement learning. In *AAMAS*, 2020.

[91] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang. Large language model based multi-agents: A survey of progress and challenges. In *IJCAI*, 2024.

[92] O. Slumbers, D. H. Mguni, K. Shao, and J. Wang. Leveraging large language models for optimised coordination in textual multi-agent reinforcement learning, 2024.

[93] H. Chen, W. Ji, L. Xu, and S. Zhao. Multi-agent consensus seeking via large language models. *arXiv preprint*, 2023.

[94] H. Li, Y. Q. Chong, S. Stepputtis, J. Campbell, D. Hughes, M. Lewis, and K. Sycara. Theory of mind for multi-agent collaboration via large language models. *EMNLP*, 2023.

[95] J. Huang, E. J. Li, M. H. Lamand T. Liang, W. Wang, Y. Yuan, W. Jiao, X. Wang, Z. Tu, and M. R. Lyu. How far are we on the decision-making of llms? evaluating llms' gaming ability in multi-agent environments. *arXiv preprint*, 2024.

[96] S. Hong, X. Zheng, J. Chen, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. Yau, Z. Lin, L. Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint*, 2023.

[97] H. Zhang, W. Du, J. Shan, Q. Zhou, Y. Du, J. B. Tenenbaum, T. Shu, and C. Gan. Building cooperative embodied agents modularly with large language models. *arXiv preprin*, 2023.

[98] S. S. Kannan, V. L. Venkatesh, and B. Minl. Smart-llm: Smart multi-agent robot task planning using large language models. In *IROS*, 2024.

[99] A. Gupta, S. Savarese, S. Ganguli, and L. Fei-Fei. Embodied intelligence via learning and evolution. *Nature communications*, 2021.

[100] N. Roy, I. Posner, T. Barfoot, P. Beaudoin, Y. Bengio, J. Bohg, et al. From machine learning to robotics: Challenges and opportunities for embodied intelligence. *arXiv preprint*, 2021.

[101] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NeurIPS*, 2017.

[102] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 2021.

[103] K. Kurach, A. Raichuk, P. Stańczyk, M. Zając, O. Bachem, L. Espeholt, C. Riquelme, D. Vincent, M. Michalski, O. Bousquet, et al. Google research football: A novel reinforcement learning environment. In *AAAI*, 2020.

[104] M. Samvelyan, T. Rashid, C. S. De Witt, G. Farquhar, N. Nardelli, T. G . Rudner, C. Hung, P. H. Torr, J. Foerster, and S. Whiteson. The starcraft multi-agent challenge. *arXiv preprint*, 2019.

[105] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *CoRR*, 2016.

[106] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033, 2012.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: This work addresses the challenge of reward function design in multi-agent reinforcement learning by proposing an LLM-based reward generation framework. In addition, we introduce the first multi-agent manipulation benchmark ManiCraft specifically designed for learning collaborative policies. The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We provide a discussion of the limitations in Appendix A.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: Our work focuses on empirical evaluation and does not provide theoretical proofs or formal analysis. We leave the development of theoretical guarantees and formal analysis as future work.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide the detailed pseudocode, experimental hyperparameter settings and the designed prompts.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: We commit to releasing the code upon the public availability of the paper.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide the detailed experimental settings in both the main experiment section and the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Following the experimental settings of prior work, i.e., Eureka, we conduct each experiment with 5 independent runs and report error bars to reflect statistically significant variability.

   Guidelines:

   - The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: These details are provided in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes, the research conducted in this paper fully conforms with the NeurIPS Code of Ethics in all respects.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

Justification: This paper presents work whose goal is to advance the field of MARL. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, all creators and original owners of the assets used in the paper are properly credited and fully respected.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Comprehensive details are presented in both the main body of the paper and the appendix.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are employed for reward generation and improvement, and comprehensive details are provided in the main body of the paper as well as in the appendix. All components of the proposed methods are independently designed and original, with no dependence on LLMs.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Limitation & Future Work

First, ReMAC represents an initial exploration into using LLMs for generating multi-agent reward functions, but it currently lacks thorough investigation and optimization regarding component design and convergence guarantees. Second, the proposed ManiCraft benchmark currently focuses on two-agent collaboration tasks, without addressing coordination among three or more agents. Therefore, developing a more rigorous theoretical foundation for ReMAC, improving its architectural components, and extending ManiCraft to support multi-agent collaboration beyond two agents is a valuable direction for future research.

B Training Details

All experiments run on a system equipped with an NVIDIA GTX 2080 Ti GPU and an Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz. Notably, all training is performed on the CPU rather than the GPU, with training time ranging from 12 to 24 hours depending on CPU computational efficiency and total steps.

B.1 Implementation of Baselines

For MASAC, we extend the SAC implementation provided by Stable-Baselines3 [1] to fit the CTDE (Centralized Training with Decentralized Execution) framework. Each agent maintains an individual policy and a critic, where the critic receives global states and actions, and learns from the rewards available to the corresponding agent. Notably, in our current setup, all agents have access to global states rather than operating under partial observability (i.e., not a POMDP setting). Detailed network architecture and hyperparameter configurations are provided in Appendix C.

Next, we detail the implementation of algorithms trained under different reward functions.

Human: *Reward Function*. A carefully crafted reward function based on extensive human trial-and-error. At each step, it returns individual rewards for each agent, composed of the agent's own reward combined with a shared team reward. The detailed reward design is provided in Section D. *Policy Learning*. Policy optimization is performed using the MASAC algorithm.

Sparse: *Reward Function*. The reward function returns 1 only upon successful task completion, and 0 at all other time steps. *Policy Learning*. Policy optimization is conducted using the MASAC algorithm.

MA-Eureka: The original single-agent Eureka maintains a population of reward functions and learns policies using PPO. To ensure a fair comparison, we implement Multi-Agent Eureka within our framework, with the following characteristics: (1) Reward Function Format: A single reward function to compute per-agent rewards; (2) Learning method: Use MASAC instead of PPO; (3) Data Usage: Consistent with Eureka's original setting [2], experience is not shared among individuals in the reward function population.

ReMAC (Ours): (1) Reward Function Format: ReMAC generates two reward functions based on the designed prompt. The first is used to compute rewards related to the individual skills each agent needs to acquire, while the second designs the collaborative rewards required among agents. The final reward for each agent is obtained by combining these two components; (2) Learning method: Use MASAC; (3) Experience is shared among individuals in the population.

Both MA-Eureka and ReMAC adopt a population size of 5. The evolution frequency is set to either 200,000 or 500,000 steps, depending on task difficulty. For more challenging tasks, 500,000 steps are used, as individuals often fail to achieve any success within 200,000 steps, making it difficult to effectively differentiate their performance. For other tasks, 200,000 steps are sufficient. During each evolution phase, the best-performing individual is selected based on fitness, and reward function optimization is subsequently carried out using the LLM reflection.

[1] https://github.com/DLR-RM/stable-baselines3
[2] https://github.com/eureka-research/Eureka

B.2 Parallel Training Framework

Since our algorithm is based on SAC (an off-policy RL algorithm that emphasizes sample efficiency and requires intensive training), we implement a parallelized version of ReMAC to reduce training time. Specifically, we maintain a central server responsible for environment interaction across the entire population. In addition, we deploy n parallel workers. The sampled data is distributed to all workers, each of which performs RL training upon receiving the data. After training, the updated network parameters are returned to the server to update the population.

This parallel framework significantly reduces training time, making it comparable to training a single RL agent. The only additional overhead comes from inter-process communication.

C Hyperparameter Setting

The hyperparameter settings for ReMAC are shown in Table 2. To ensure fairness in comparison, all algorithms are configured with identical hyperparameters.

Table 2: Details of hyperparameters.

Parameter	Value
Optimizer	Adam
Init exploration steps	10000
Learning rate	$3e-4$
Batch size	128
γ	0.8
Hidden size	256
Layer size	3
τ	0.05
log-std-max	2
log-std-min	-20
Replay buffer size	1000000

D Benchmark Details

ManiCraft is developed based on MuJoCo. For each robotic arm, we assign an anchor point to the gripper and use Mocap for motion tracking and control. Although we considered using inverse kinematics (IK) for control, it significantly reduces simulation sampling speed. Therefore, we ultimately adopt Mocap for more efficient execution.

The following outlines the key characteristics of each task and the corresponding reward function. Our reward functions are largely inspired by the formulation methodology employed in ManiSkill3 [3].

Task 1: Co-Sweep-Easy Task:

- **Task Description**: Panda holds the broom to sweep one cube (Red Cube) into the dustpan held by UR.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions and velocities of the UR and Panda arms, the gripper opening size, the position of the object, the position of the broom's bottom end, and the position of the dustpan's bottom end.
- **Reward Function**: The distance between the broom and the red cube is denoted as $d_{\text{Broom\_Red}}$, and the distance between the dustpan and the red cube is $d_{\text{Dustpan\_Red}}$. A successful completion yields a reward of $r_{\text{Success}} = 100$; otherwise, it is 0. The final reward functions are defined as:

$$r_{\text{UR}} = 1 - \tanh(d_{\text{Dustpan\_Red}}) + r_{\text{Success}},$$
$$r_{\text{Panda}} = 1 - \tanh(d_{\text{Broom\_Red}}) + r_{\text{Success}}.$$

[3]https://github.com/haosulab/ManiSkill

- **Characteristic**: This task is relatively simple, with the main challenge lying in aligning the broom, dustpan, and cube to ensure the cube can be successfully swept into the dustpan.

Task 2: Co-Sweep-Mid Task:

- **Task Description**: Panda holds the broom to sweep two cubes into the dustpan held by UR.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions and velocities of the UR and Panda arms, the gripper opening size, the position of the object, the position of the broom's bottom end, and the position of the dustpan's bottom end.
- **Reward Function**: The distance between the broom and the red cube is denoted as $d_{\text{Broom\_Red}}$, and the distance to the green cube as $d_{\text{Broom\_Green}}$. Similarly, the distance between the dustpan and the red cube is denoted as $d_{\text{Dustpan\_Red}}$, and the distance to the green cube as $d_{\text{Dustpan\_Green}}$. A successful completion yields a reward of $r_{\text{Success}} = 100$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{UR\_red\_distance}} &= 1 - \tanh(d_{\text{Dustpan\_Red}}), \\ r_{\text{Panda\_red\_distance}} &= 1 - \tanh(d_{\text{Broom\_Red}}), \\ r_{\text{UR\_green\_distance}} &= 1 - \tanh(d_{\text{Dustpan\_Green}}), \\ r_{\text{Panda\_green\_distance}} &= 1 - \tanh(d_{\text{Broom\_Green}}). \end{aligned}$$

  We first focus on manipulating the red cube. If the red cube has not yet been successfully completed, the reward is guided as follows: $[r_{\text{UR\_red\_distance}}, r_{\text{Panda\_red\_distance}}]$. If the red cube has already been successfully manipulated, and the green cube has not yet been completed, the rewards are $[r_{\text{UR\_green\_distance}}, r_{\text{Panda\_green\_distance}}]$. When the green cube is successfully manipulated, indicating that the entire task is complete, the reward is: $[r_{\text{Success}}, r_{\text{Success}}]$.
  Characteristic: This task presents a moderate level of difficulty, with an emphasis on the correct sequence of operations to ensure that both objects are placed into the dustpan.

Task 3: Co-Sweep-Hard Task:

- **Task Description**: Panda holds the broom to sweep three cubes into the dustpan held by UR.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions and velocities of the UR and Panda arms, the gripper opening size, the position of the object, the position of the broom's bottom end, and the position of the dustpan's bottom end.
- **Reward Function**: The distance between the broom and the red cube is denoted as $d_{\text{Broom\_Red}}$, the distance to the green cube as $d_{\text{Broom\_Green}}$, and the distance to the blue cube as $d_{\text{Broom\_Blue}}$. Similarly, the distance between the dustpan and the red cube is denoted as $d_{\text{Dustpan\_Red}}$, and the distance to the green cube as $d_{\text{Dustpan\_Green}}$, and the distance to the blue cube as $d_{\text{Dustpan\_Blue}}$. A successful completion yields a reward of $r_{\text{Success}} = 100$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{UR\_red\_distance}} &= 1 - \tanh(d_{\text{Dustpan\_Red}}), \\ r_{\text{Panda\_red\_distance}} &= 1 - \tanh(d_{\text{Broom\_Red}}), \\ r_{\text{UR\_green\_distance}} &= 1 - \tanh(d_{\text{Dustpan\_Green}}), \\ r_{\text{Panda\_green\_distance}} &= 1 - \tanh(d_{\text{Broom\_Green}}), \\ r_{\text{UR\_blue\_distance}} &= 1 - \tanh(d_{\text{Dustpan\_Blue}}), \\ r_{\text{Panda\_blue\_distance}} &= 1 - \tanh(d_{\text{Broom\_Blue}}). \end{aligned}$$

  We first focus on manipulating the red cube. If the red cube has not yet been successfully manipulated, the reward is: $[r_{\text{UR\_red\_distance}}, r_{\text{Panda\_red\_distance}}]$. If the red cube has been completed, we shift our attention to the green cube. If the green cube has not yet been successfully manipulated, the reward is: $[r_{\text{UR\_green\_distance}}, r_{\text{Panda\_green\_distance}}]$. Once both the red and green cubes have been completed, we focus on the blue cube. If the blue cube has not yet been successfully manipulated, the reward is: $[r_{\text{UR\_blue\_distance}}, r_{\text{Panda\_blue\_distance}}]$. Finally, when all three cubes have been successfully placed into the dustpan, indicating task completion, the reward is: $[r_{\text{Success}}, r_{\text{Success}}]$. **Characteristic**: This task is highly challenging compared to the previous two, as it involves a longer temporal horizon and requires clearing more cubes within the same number of steps.

Task 4: Co-Push Task:

- **Task Description**: The Panda and UR push the cubes at their sides together.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions and velocities of the UR and Panda arms, the gripper opening size, the position of the red cube and the green cube.
- **Reward Function**: The distance between the UR arm's end-effector and the green cube is denoted as $d_{\text{UR\_Green}}$, the distance between the Panda arm's end-effector and the red cube is denoted as $d_{\text{Panda\_Red}}$, the distance between the Panda arm's end-effector and the UR arm's end-effector $d_{\text{Panda\_UR}}$, and the distance between the red cube and the green cube is denoted as $d_{\text{Red\_Green}}$. A successful completion yields a reward of $r_{\text{Success}} = 20$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{UR\_green\_distance}} &= 1 - \tanh(d_{\text{UR\_Green}}), \\ r_{\text{Panda\_red\_distance}} &= 1 - \tanh(d_{\text{Panda\_Red}}), \\ r_{\text{Red\_green\_distance}} &= 1 - \tanh(d_{\text{Red\_Green}}), \\ r_{\text{Panda\_UR\_distance}} &= 1 - \tanh(d_{\text{Panda\_UR}}). \end{aligned}$$

  The reward for the UR arm is defined as $r_{\text{UR\_green\_distance}} + r_{\text{Red\_green\_distance}} + r_{\text{Panda\_UR\_distance}} + r_{\text{Success}}$, and the reward for the Panda arm is defined as $r_{\text{Panda\_red\_distance}} + r_{\text{Red\_green\_distance}} + r_{\text{Panda\_UR\_distance}} + r_{\text{Success}}$. **Characteristic**: This task places greater emphasis on the exploration of collaborative strategies between the two robotic arms.

Task 5: Co-Stack-On:

- **Task Description**: The Panda places the cube from its side onto the coaster next to the UR.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions and velocities of the UR and Panda arms, the gripper opening size, the positions of the cube and the coaster.
- **Reward Function**: The distance between the UR arm's end-effector and the coaster is denoted as $d_{\text{UR\_Coaster}}$, the distance between the Panda arm's end-effector and the cube is denoted as $d_{\text{Panda\_Cube}}$, the distance between the Panda arm's end-effector and the UR arm's end-effector $d_{\text{Panda\_UR}}$, and the distance between the cube and the coaster is denoted as $d_{\text{Cube\_Coaster}}$. A successful completion yields a reward of $r_{\text{Success}} = 20$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{UR\_coaster\_distance}} &= 1 - \tanh(5 \cdot d_{\text{UR\_Coaster}}), \\ r_{\text{Panda\_cube\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Panda\_Cube}}), \\ r_{\text{Cube\_coaster\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Cube\_Coaster}}), \\ r_{\text{Panda\_UR\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Panda\_UR}}). \end{aligned}$$

  The reward for the UR arm is defined as $r_{\text{UR\_coaster\_distance}} + r_{\text{Panda\_UR\_distance}} + r_{\text{Cube\_coaster\_distance}} + r_{\text{Success}}$, and the reward for the Panda arm is defined as $r_{\text{Panda\_cube\_distance}} + r_{\text{Panda\_UR\_distance}} + r_{\text{Cube\_coaster\_distance}} + r_{\text{Success}}$. **Characteristic**: This task poses a high level of difficulty, as it requires the Panda arm to lift the cube and the UR arm to position the coaster within the Panda's reachable range. Achieving successful coordination necessitates finely tuned reward signals.

Task 6: Co-Trans:

- **Task Description**: The UR grasps the cube next to the Panda.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions and velocities of the UR and Panda arms, the gripper opening size, the position of the cube.
- **Reward Function**: The distance between the UR arm's end-effector and the cube is denoted as $d_{\text{UR\_Cube}}$, When the cube's y-axis position is below 0.5, it becomes reachable by the UR. Thus, a success reward $r_{\text{Panda\_Success}} = 1$ for the Panda is obtained when the cube is placed at a position where its $y < 0.5$. After doing so, the Panda must return to its initial position to avoid potential collisions, with $d_{\text{Panda\_Start}}$ representing the distance from the starting point. The distance between the cube's position along the y-axis and the boundary at 0.5 is defined as $d_{\text{Cube\_Boundary}}$. A successful completion yields a reward of $r_{\text{Success}} = 20$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{UR\_cube\_distance}} &= 1 - \tanh(5 \cdot d_{\text{UR\_Cube}}), \\ r_{\text{Panda\_cube\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Panda\_Cube}}) \quad \text{if Panda\_Success is False, else } 1, \\ r_{\text{Panda\_success}} &= 1 \quad \text{if Panda\_Success is True, else } 0, \\ r_{\text{Panda\_start\_distance}} &= 2 - 2 \cdot \tanh(5 \cdot d_{\text{Panda\_Start}}) \quad \text{if Panda\_Success is True, else } 0, \\ r_{\text{Cube\_boundary\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Cube\_Boundary}}) \quad \text{if Panda\_Success is False, else } 1, \end{aligned}$$

  The reward for the UR arm is defined as $r_{\text{UR\_cube\_distance}} + r_{\text{Success}}$, and the reward for the Panda arm is defined as $r_{\text{Panda\_cube\_distance}} + r_{\text{Panda\_start\_distance}} + r_{\text{Panda\_success}} + r_{\text{Cube\_boundary\_distance}} + r_{\text{Success}}$. **Characteristic**: This task is relatively challenging. It first requires the Panda arm to place the cube within the UR arm's reachable area and then move away; otherwise, a collision may occur, preventing the UR from successfully grasping the cube.

Task 7: Co-Open:

- **Task Description**: Two URs work together to open the cabinet door.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions and velocities of the two URs, the gripper opening size, the rotation angles of both URs, the positions of the left and right door handles, and the joint positions ($qpos$).
- **Reward Function**: The distance between the left UR arm's end-effector and the left door handle is denoted as $d_{\text{Left\_UR\_Handle}}$, the distance between the right UR arm's end-effector and the right door handle is denoted as $d_{\text{Right\_UR\_Handle}}$, the distance between the left handle and right handle is denoted as $d_{\text{Left\&Right\_Handle}}$, the distance between the left handle and its fully opened y-position as $d_{\text{Left\_Open}}$, and the distance between the right handle and its fully opened y-position as $d_{\text{Right\_Open}}$. A successful completion yields a reward of $r_{\text{Success}} = 20$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{Left\_UR\_handle\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Left\_UR\_Handle}}), \\ r_{\text{Right\_UR\_handle\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Right\_UR\_Handle}}), \\ r_{\text{Left\_open}} &= 1 - \tanh(5 \cdot d_{\text{Left\_Open}}), \\ r_{\text{Right\_open}} &= 1 - \tanh(5 \cdot d_{\text{Right\_Open}}), \\ r_{\text{Left\&Right\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Left\&Right\_Handle}}). \end{aligned}$$

  The reward for the UR arm is defined as $r_{\text{Left\_UR\_handle\_distance}} + r_{\text{Left\_open}} + r_{\text{Left\&Right\_distance}} + r_{\text{Success}}$, and the reward for the Panda arm is defined as $r_{\text{Right\_UR\_handle\_distance}} + r_{\text{Right\_open}} + r_{\text{Left\&Right\_distance}} + r_{\text{Success}}$. **Characteristic**: A key characteristic of this task is the low friction of the gripper, which makes it prone to slipping when grasping the door handle. As a result, precise control is required to successfully complete the door-opening operation.

Task 8: Co-Close:

- **Task Description**: Two URs work together to close the cabinet door.

- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1) + 3 (change in Euler angles, range from -1 to 1)
- **State Space**: The 3D positions and velocities of the two URs, the gripper opening size, the rotation angles of both URs, the positions of the left and right door handles, and the joint positions ($qpos$).
- **Reward Function**: The distance between the left UR arm's end-effector and the left door handle is denoted as $d_{\text{Left\_UR\_Handle}}$, the distance between the right UR arm's end-effector and the right door handle is denoted as $d_{\text{Right\_UR\_Handle}}$, the distance between the left handle and right handles denoted as $d_{\text{Left\&Right\_Handle}}$, the initial distance between the left handle and right handles denoted as $d_{\text{Init\_Left\&Right\_Handle}}$. A successful completion yields a reward of $r_{\text{Success}} = 20$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{Left\_UR\_handle\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Left\_UR\_Handle}}), \\ r_{\text{Right\_UR\_handle\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Right\_UR\_Handle}}), \\ r_{\text{Left\&Right\_change\_distance}} &= 5 \cdot (d_{\text{Init\_Left\&Right\_Handle}} - d_{\text{Left\&Right\_Handle}}). \end{aligned}$$

The reward for the UR arm is defined as $r_{\text{Left\_UR\_handle\_distance}} + r_{\text{Left\&Right\_change\_distance}} + r_{\text{Success}}$, and the reward for the Panda arm is defined as $r_{\text{Right\_UR\_handle\_distance}} + r_{\text{Left\&Right\_change\_distance}} + r_{\text{Success}}$. **Characteristic**: This task is relatively simple and typically does not require complex manipulation. However, it involves controlling a higher-dimensional action space.

Task 9: Co-Place:

- **Task Description**: Move the object next to the Panda to the target position next to the UR.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions and velocities of the UR and Panda arms, the gripper opening size, the position of the cube, the target position.
- **Reward Function**: The distance between the UR arm's end-effector and the cube is denoted as $d_{\text{UR\_Cube}}$, the distance between the Panda arm's end-effector and the cube is denoted as $d_{\text{Panda\_Cube}}$, the distance between the cube and the target position is denoted as $d_{\text{Cube\_Target}}$, the distance between the UR arm's end-effector position and the Panda arm's end-effector position $d_{\text{UR\_Panda}}$. A successful completion yields a reward of $r_{\text{Success}} = 20$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{UR\_cube\_distance}} &= 1 - \tanh(5 \cdot d_{\text{UR\_Cube}}), \\ r_{\text{Panda\_cube\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Panda\_Cube}}), \\ r_{\text{Cube\_target}} &= 5 \cdot (1 - \tanh(5 \cdot d_{\text{Cube\_Target}})), \\ r_{\text{UR\_Panda\_distance}} &= 1 - \tanh(5 \cdot d_{\text{UR\_Panda}}). \end{aligned}$$

The reward for the UR arm is defined as $r_{\text{UR\_cube\_distance}} + r_{\text{Cube\_target}} + r_{\text{UR\_Panda\_distance}} + r_{\text{Success}}$, and the reward for the Panda arm is defined as $r_{\text{Panda\_cube\_distance}} + r_{\text{Cube\_target}} + r_{\text{UR\_Panda\_distance}} + r_{\text{Success}}$. **Characteristic**: This task is relatively difficult and consists of multiple steps. First, the Panda arm must place the object within the UR arm's reachable area; then, the UR arm is responsible for delivering the object to the target location. Effective collaboration is essential, as poor coordination often leads to collisions and task failure due to blockage.

Task 10: Co-Lift:

- **Task Description**: UR and Panda work together to lift a rectangular object.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions of the UR and Panda arms, the gripper opening size, the target position for the UR to grasp, and the target position for the Panda to grasp.

- **Reward Function**: The distance between the UR arm's end-effector and its target position is denoted as $d_{\text{UR\_Target}}$, the distance between the Panda arm's end-effector and its target position is denoted as $d_{\text{Panda\_Target}}$, If the Panda successfully grasps its target position, it receives a reward of 1; otherwise, the reward is 0. The same applies to the UR. The initial height of the object is denoted as H_{Init}. The height of the current target position for the UR to grasp is denoted as $H_{\text{UR\_Target}}$. The height of the current target position for the Panda to grasp is also denoted as $H_{\text{Panda\_Target}}$. A successful completion yields a reward of $r_{\text{Success}} = 20$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{UR\_target\_distance}} &= 1 - \tanh(5 \cdot d_{\text{UR\_Target}}), \\ r_{\text{Panda\_target\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Panda\_Target}}), \\ r_{\text{Panda\_grasped}} &= 1 \quad \text{if Panda\_Grasped is True, else } 0, \\ r_{\text{UR\_grasped}} &= 1 \quad \text{if UR\_Grasped is True, else } 0, \\ r_{\text{UR\_Lift}} &= 10 \cdot \max(H_{\text{UR\_Target}} - H_{\text{Init}}, 0), \\ r_{\text{Panda\_Lift}} &= 10 \cdot \max(H_{\text{Panda\_Target}} - H_{\text{Init}}, 0). \end{aligned}$$

  The reward for the UR arm is defined as $r_{\text{UR\_target\_distance}} + r_{\text{UR\_grasped}} + r_{\text{UR\_Lift}} + r_{\text{Success}}$, and the reward for the Panda arm is defined as $r_{\text{Panda\_target\_distance}} + r_{\text{Panda\_grasped}} + r_{\text{Panda\_Lift}} + r_{\text{Success}}$. **Characteristic**: This task is highly challenging, as it requires both grippers to simultaneously grasp their respective target points and lift the object together into the air to achieve success. It places a strong demand on precise coordination between the two arms.

Task 11: Co-Grasp:

- **Task Description**: UR and Panda work together to grasp a rectangular object.
- **Action Space**: 3 (change in end-effector pose, range from -1 to 1) + 1 (gripper openness, range from -1 to 1)
- **State Space**: The 3D positions of the UR and Panda arms, the gripper opening size, the target position for the UR to grasp, and the target position for the Panda to grasp.
- **Reward Function**: The distance between the UR arm's end-effector and its target position is denoted as $d_{\text{UR\_Target}}$, the distance between the Panda arm's end-effector and its target position is denoted as $d_{\text{Panda\_Target}}$, If the Panda successfully grasps its target position, it receives a reward of 1; otherwise, the reward is 0. The same applies to the UR. A successful completion yields a reward of $r_{\text{Success}} = 20$, otherwise, it is 0. The reward components are defined as follows:

$$\begin{aligned} r_{\text{UR\_target\_distance}} &= 1 - \tanh(5 \cdot d_{\text{UR\_Target}}), \\ r_{\text{Panda\_target\_distance}} &= 1 - \tanh(5 \cdot d_{\text{Panda\_Target}}), \\ r_{\text{Panda\_grasped}} &= 1 \quad \text{if Panda\_Grasped is True, else } 0, \\ r_{\text{UR\_grasped}} &= 1 \quad \text{if UR\_Grasped is True, else } 0. \end{aligned}$$

  The reward for the UR arm is defined as $r_{\text{UR\_target\_distance}} + r_{\text{UR\_grasped}} + r_{\text{Success}}$, and the reward for the Panda arm is defined as $r_{\text{Panda\_target\_distance}} + r_{\text{Panda\_grasped}} + r_{\text{Success}}$. **Characteristic**: This task presents a moderate level of difficulty, as it requires each gripper to secure its designated target point. While less demanding than full joint manipulation, it still necessitates a reasonable degree of coordination.

E Prompt Design

This section presents the detailed prompt design used in ReMAC, which consists of the following three components:

1. **Initial Prompt**: The initial prompt is primarily used to define the role of the LLM, specify the target task, provide the required code format, and outline the success criteria.

Agent-level Reward Function

```
You are a reward engineer whose goal is to design reward
functions to solve multi-agent reinforcement learning tasks
as effectively as possible.
You need to create two reward functions, get_team_level_reward
and get_agent_level_reward, to guide multi-agent
collaboration.
- get_agent_level_reward should focus on the individual rewards
of each agent within the team.  It should guide each agent to
learn various skills.
- get_team_level_reward should focus on the team's overall
reward.  It should use a goal-condition reward policy to
encourage team collaboration.
The task is:  {Task_description}
The format for get_agent_level_reward should be:
{agent_level_reward_function}.
The format for get_team_level_reward should be:
{team_level_reward_function}.
The success criteria is:
{success_criteria}
```

2. **Population Generation and Evolution Prompts**: The prompt is used to guide both the generation and refinement of reward functions.

Agent-level Reward Function

```
Please strictly follow them to write the
”agent_level_reward_function“ and "team_level_reward_function"
above separately, using the keys from the list below as
function inputs. However, do not introduce any keys that are
not present in the list.
parameter_info
The code output should be formatted as a python code string:
"“‘python ... “‘". The return variables must be consistent
with those provided in the given functions. You should
neither add nor remove variables, nor modify their names.
Repeatedly verify that all input variables in the function
definition exist in the list, ensuring no errors in naming or
the introduction of new variables.
Each reward function must include the following:
(1) Reward Components: Add or remove certain components. If
there are no modifications, please provide a brief reason.
for example, add xxx reward component to encourage the agent
to do xxx and apply a weight x for better xxx
(2) Reward Weights: Adjust the weight of certain reward
components or change the reward coefficients. If there are
no modifications, please provide a brief reason. for example,
change the reaching reward weight from 1.0 to 5.0 for better
xxx
(3) Reward Calculation: Modify the reward calculation
methods. If there are no modifications, please provide a
brief reason. for example, change the reaching or catching
reward calculation method or add exp to xxx reward component
to encourage the agent to do xxx
Carefully analyze the skills that each agent needs to learn
in the task, as well as the collaboration skills required at
the team level. Provide a detailed analysis, and explicitly
specify in the reward functions how each reward component
contributes to achieving these objectives.
Some helpful tips for writing the reward function code:
(1) Please do not simply transform the reward components or
adjust the hyperparameters;
(2) Make sure the type of each input variable is correctly
specified; All the necessary information is provided in the
function inputs, and "self" is neither referenced nor called;
(3) It is necessary to adjust some parameters of the existing
reward function, such as scaling the reward for grasping,
scaling the proximity reward, or scaling the success reward;
(4) If you choose to transform a reward component, then
you must also introduce a temperature parameter inside the
transformation function;
(5) First separate rewards into the agent-level and team-level
rewards, which can reduce the difficulty of credit assignment;
(6) Ensure there is shaping guidance for the robot
end-effector reaching the object, as well as for the object
reaching the target position;
(7) Rewards closer to task success should have higher weights,
for example, moving the object toward the goal should be
weighted more than merely approaching it.
```

3. **Reflection Prompts**. It is used to improve reward functions based on feedback information.

Agent-level Reward Function

```
Based on the above reward function, the current MARL policy's
success rate is win_rate,
The changes in different rewards (accumulated component
rewards) during the learning process at evo_steps
environment steps (recorded at intervals of 50,000 steps):
reward_learning_curves.
Please carefully analyze the policy feedback.  Some helpful
tips for analyzing the policy feedback:
(1) Considering multi-agent collaboration, first analyze
which agent is underperforming and which skills are not being
learned, leading to the issue.
(2) If the issue lies with a specific skill, rewrite the
reward component associated with that skill.
(3) If a specific agent is underperforming, consider rewriting
or improving its agent-level reward.
(4) If there are issues with the team reward, rewrite or
improve the team-level reward function.
(5) If the success rate remains at zero, check if the issue is
due to reward design.  If so, rewrite all reward functions.
(6) If the reward is excessively large, it may need to be
appropriately scaled to avoid learning issues.
Finally, verify that all necessary reward components are
included and computed correctly, for example, the arm's
reaching reward is mistakenly based on the distance between
the object and the target.
```