Speaking the Language of Teamwork: LLM-Guided Credit Assignment in Multi-Agent Reinforcement Learning

Muhan Lin[§†] **Shuyang Shi**[§◇] **Yue Guo**[‡] **Vaishnav Tadiparthi**[¶]
Behdad Chalaki[♠] **Ehsan Moradi Pari**[¶] **Simon Stepputtis**[ǀ] **Woojun Kim**[‡]
Joseph Campbell[†] **Katia Sycara**[‡]

Abstract

Credit assignment, the process of attributing credit or blame to individual agents for their contributions to a team's success or failure, remains a fundamental challenge in multi-agent reinforcement learning (MARL), particularly in environments with sparse rewards. Commonly-used approaches such as value decomposition often lead to suboptimal policies in these settings, and designing dense reward functions that align with human intuition can be complex and labor-intensive. In this work, we propose a novel framework where a large language model (LLM) generates dense, agent-specific rewards based on a natural language description of the task and the overall team goal. By learning a potential-based reward function over multiple queries, our method reduces the impact of ranking errors while allowing the LLM to evaluate each agent's contribution to the overall task. Through extensive experiments, we demonstrate that our approach achieves faster convergence and higher policy returns compared to state-of-the-art MARL baselines.

1 Introduction

From coordinating autonomous vehicles [Shalev-Shwartz et al., 2016, Zhang et al., 2024b] in traffic systems [Wiering et al., 2000, Chu et al., 2019] to managing distributed networks, multi-agent reinforcement learning (MARL) enables agents to learn policies through interaction. It is common practice in MARL to learn decentralized policies over local observations to avoid exponential growth of the joint state-action space, but this creates non-stationarity as agents' policies evolve simultaneously.

The centralized-training-decentralized-execution (CTDE) paradigm [Lowe et al., 2017, Kim et al., 2023, Kim and Sung, 2023] mitigates this by using global state and joint action information during training. However, a central challenge is the credit assignment problem [Foerster et al., 2018, Rashid et al., 2020]: attributing team outcomes to individual agents. MARL involves multiple agents whose actions collectively influence the *team reward*, making contributions hard to separate.

Credit assignment can be implicitly addressed with value decomposition methods [Rashid et al., 2020, Sunehag et al., 2017, Son et al., 2019] in CTDE. These approaches decompose the team value into a (possibly nonlinear) combination of per-agent values. Despite their successes [Wang et al.,

---

[§] This work was conducted during the author's previous affiliation with School of Computer Science, Carnegie Mellon University, Pittsburgh, USA; [†] Department of Computer Science, Purdue University, West Lafayette, USA; [◇] 1X, Palo Alto, USA; [‡] School of Computer Science, Carnegie Mellon University, Pittsburgh, USA; [¶] Honda Research Institute, Ann Arbor, USA; [♠] Torc Robotics, Ann Arbor, USA (This work was conducted during the author's previous affiliation with Honda Research Institute, Ann Arbor, USA); [ǀ] Virginia Tech, Blacksburg, USA; **Correspondence:** `lin2265@purdue.edu`

2nd Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2025).

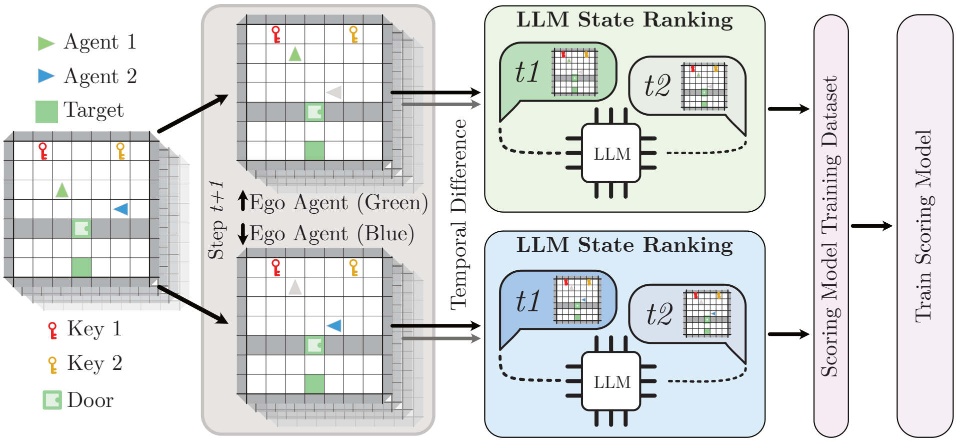

Figure 1: Overview of our method LCA: We first generate the agent-specific encodings of state observations, and then prompt an LLM to execute pairwise state ranking from each agent's perspective in the contexts of collaboration. Specifically, if ranking state pairs in Agent 1's perspective, Agent 1 will be encoded as the "ego" agent and other agents as "teammates" in the observation, allowing the LLM to differentiate them with the language-based observation description. The individual rewards trained with such agent-specific ranking results properly handle the credit assignment in MARL. We test our approach in the grid world and pistonball environments.

2020],they struggle in sparse-reward settings with infrequent or delayed feedback [Liu et al., 2023]. Such drawback limits the application of these methods, as sparse reward settings remain exceedingly common largely due to the difficulty of crafting dense, value-aligned reward functions [Leike et al., 2018, Skalse et al., 2022, Knox et al., 2023, Booth et al., 2023].

Recent work shows that large language models (LLMs) can generate preference rankings to learn dense reward functions [Lee et al., 2023]. While effective in single-agent sparse-reward settings [Lin et al., 2024], their role in MARL credit assignment remains unclear. This work seeks to answer the question: **can LLMs assign credits in MARL by generating informative, agent-specific rewards based on natural language descriptions of tasks and goals?**

In this paper, we propose **L**LM-guided **C**redit **A**ssignment **(LCA)**, a novel framework that integrates LLMs into the MARL training process to facilitate credit assignment with sparse rewards. Our approach extracts team objectives and key task steps from existing team rewards, uses an LLM to rank agents' actions, and prefers those most contributive to success. The rankings train dense potential-based per-agent rewards, addressing both credit assignment and reward sparsity. Our work makes the following key contributions:

1. An LLM-based method that generates dense, agent-specific rewards from natural language team goals and sparse team rewards.
2. Empirical results showing LCA improves returns and convergence over baselines, even with smaller, error-prone LLMs..

2 Related Works

Credit assignment in multi-agent reinforcement learning remains a fundamental challenge, especially in environments with sparse team rewards. Traditional approaches fall into two classes: value decomposition [Sunehag et al., 2017, Rashid et al., 2020, Du et al., 2019, Foerster et al., 2018] and algorithmic variants such as the gradient-descent decomposition [Su et al., 2020]. Hybrid methods also exist. Kapoor et al. [2024] adapts the partial reward decoupling into MAPPO [Yu et al., 2022] to eliminate contributions from other irrelevant agents based on attention. Wen et al. [2022] utilizes transformer with PPO loss [Schulman et al., 2017] adapted to the value decomposition idea.

The methods above improve credit assignment but degrade with sparse or delayed rewards. For example, Liu et al. [2023] shows the poor performance of QMIX [Rashid et al., 2020] with sparse rewards. However, designing dense rewards is difficult given the complexity of tasks [Leike et al., 2018, Knox et al., 2023, Booth et al., 2023]. Social-influence-based rewarding calculates dense individual rewards [Jaques et al., 2019] but requires estimation of teammates' behavior models, adding training overhead.

In single-agent RL, dense rewards are often generated via Reinforcement Learning with Human Feedback (RLHF) [Christiano et al., 2017] and Reinforcement Learning with AI Feedback (RLAIF) [Lee et al., 2023], successfully applied to domains like text summarization and dialogue generation [Ziegler et al., 2020], but not designed for multi-agent settings. [Zhang et al., 2024a] shows one direction of LLM-generated dense rewards for multi-agent credit assignment. Utilizing the coding capabilities of LLM, this method iteratively queries LLM to generate multiple reward functions with high density and refine them gradually in the training process. However, this method can risk LLM hallucination problems, which can cause misleading rewards due to inconsistent rankings or other ranking errors. Considering these problems, our method extends the potential-based RLAIF [Lin et al., 2024], which mitigates hallucination through multi-query approach, from single-agent to MARL credit assignment.

3 Background

Multi-Agent Reinforcement Learning: We consider a fully cooperative Markov Game [Matignon et al., 2012], which generalizes the Markov Decision Process (MDP) to multi-agent settings where multiple agents interact in a shared space and collaborate by maximizing a common reward function. A fully cooperative Markov Game is represented by the tuple $(N, S, \{A_i\}_{i=1}^N, P, R, \gamma)$, where N is the number of agents, S represents the set of global states, $\{A_i\}$ is the action space for each agent, and $P(s'|s, a_1, \dots, a_N)$ describes the probability of transitioning from one state to another based on the joint actions of all agents. The agents share a reward function $R(s, a_1, a_2, \dots, a_N)$, which assigns a common reward based on the state-action pairs. The objective is for the agents to collaboratively learn policies that maximize the cumulative discounted reward, where γ denotes the discount factor.

Preference-Based Reinforcement Learning: The underlying framework of our work is preference-based reinforcement learning, where preference labels over agent behaviors are used to train reward functions for RL policy training [Christiano et al., 2017, Ibarz et al., 2018, Lee et al., 2021a,b]. Given a pair of states (s_a, s_b), an annotator labels preference $y \in \{0, 1\}$ to indicate which state is closer to the task goal: $y = 0$ if s_a is ranked higher than s_b, and $y = 1$ if s_b is ranked higher than s_a.

We introduce a parameterized state-scoring function σ_ψ, often referred to as the potential function and typically identified with the reward model r_ψ. Based on this, the probability that the s_a is ranked higher than s_b is computed with the standard Bradley-Terry model [Bradley and Terry, 1952],

$$P_\psi[s_a \succ s_b] = \frac{\exp(\sigma_\psi(s_a))}{\exp(\sigma_\psi(s_a)) + \exp(\sigma_\psi(s_b))} = \text{sigmoid}(\sigma_\psi(s_a) - \sigma_\psi(s_b)), \tag{1}$$

Utilizing a preference dataset $\mathcal{D} = \{(s_a, s_b, y) | s_a, s_b \in \mathcal{S}\}$, preference-based RL trains the state-scoring model σ_ψ via minimizing the cross-entropy loss. This process aims to maximize the score difference between higher-ranked and lower-ranked states:

$$\mathcal{L} = -\mathbb{E}_{(s_a, s_b, y)\sim\mathcal{D}}\Big[\mathbb{I}\{y = (s_a \succ s_b)\} \log P_\psi[s_a \succ s_b] + \mathbb{I}\{y = (s_b \succ s_a)\} \log P_\psi[s_b \succ s_a]\Big], \tag{2}$$

with $\mathbb{I}\cdot$ as the indicator function. This framework is applied in both RLHF and RLAIF, where the outputs of the state-scoring model are directly used as rewards. The primary difference between these approaches lies in the choice of annotator—either a human or a large language model (LLM).

Using LLMs for preference labeling reduces human labor but with inevitable ranking errors, resulting in misleading rewards and inefficient training. One critical source of errors is inconsistent rankings on the same state pairs across multiple prompting trials when the LLM is uncertain about their preference. It is proven that formulating potential-based RLAIF rewards as $r(s_t, s_{t+1}) = \sigma_\psi(s_{t+1}) - \sigma_\psi(s_t)$, instead of $\sigma_\psi(s_t)$, causes $r(s_t, s_{t+1})$ to converge to 0 as LLM uncertainty increases [Lin et al., 2024]. Such uninformative reward effectively mitigates the negative impact of inconsistent rankings.

4 Method

Existing RLAIF approaches [Lin et al., 2024] do not lend themselves well to multi-agent settings when ranking joint state-actions. Consider a two-agent scenario in which the agents perform actions with conflicting contributions toward the team goal: one positive and one negative. The positive reward from a beneficial action that contributes to the team's success is canceled out by the negative reward from another agent. This results in an ambiguous state which is difficult for an LLM to rank when considering both agents, ultimately resulting in a sparse rather than dense reward function. In contrast, our LCA approach seeks to decompose the joint preference ranking into individual preference rankings for the purpose of learning individual reward functions, overcoming this issue.

4.1 LLM-based Reward Decomposition

Describing Team Goals from Team Rewards: Given that not all environments provide explicit, natural language descriptions of states, goals, or sub-goals, this information can be inferred from the team reward structure by investigating a trajectory sampled beforehand. Without loss of generalization, we assume that there exists one team reward function, $r_t(s_i)$, from the environment, which is usually sparse (We assume it does not include step penalty and is not finely hand-crafted). Therefore, on a trajectory randomly sampled without a limit of max steps - which means it ends when the team task is completed - there are only a few states s_i where $r_t(s_i) \neq 0$. If $r_t(s_i) > 0$, s_i should be a key landmark of completing the team task. If $r_t(s_i) < 0$, it would be critical to avoid this state s_i. Therefore, the natural language description of such s_i following the order they appear on the sampled trajectory can provide LLM enough information about how the agent team should complete the task, which will be critical information for agent-specific state ranking.

Agent-specific State Ranking: We prompt an LLM to implicitly assign credits to each agent separately by ranking state pairs based on the agent's own actions from that agent's perspective. We first generate an agent-specific encoding o^i of the observation o of a state s by labeling the agent i itself as the "ego" and any other agent as the "teammate", allowing the LLM and state-scoring models to identify which agent they are evaluating. Given any state transition (s, a, s'), where $a = \langle a_1, \dots, a_n \rangle$ and n is the number of agents, the LLM generates a preference label for agent i as:

$$y^i(s, a, s') = y^i(o^i, a_i, o'^i).$$

The LLM is then prompted to reason from agent i's perspective to determine whether the agent's action a_i between these two states, o^i and o'^i, is appropriate for collaboration. If agent i performs a correct action while another agent j performs an incorrect one—a scenario where single-agent-style RLHF struggles to generate a single ranking—this method assigns:

$$y^i(s, a_i, s') = (o'^i \succ o^i) = (s' \succ s),$$

$$y^j(s, a_j, s') = (o^j \succ o'^j) = (s \succ s').$$

LLM-Guided Individual Reward Training: Given that the LLM implicitly assigns credit by generating differentiated rankings for each agent i: $\mathcal{D}^i = \{(s_a, s_b, y^i) | s_a, s_b \in \mathcal{S}\}$, these rankings can be used to train individual state-scoring models $\sigma^i(o^i)$. Denoting the total number of times each state pair (s_a, s_b) is ranked in $\mathcal{D}^i$ as N_{query}, the loss function for each individual state-scoring model will be

$$\begin{aligned}
\mathcal{L}^i &= -\mathbb{E}_{(s_a,s_b,y^i)\sim\mathcal{D}^i}\Bigg[\mathbb{E}_{N_{query}}\Bigg[\mathbb{I}\{y^i = (s_a \succ s_b)\}\log P^i_\psi[o^i_a \succ o^i_b] + \\
&\quad \mathbb{I}\{y^i = (s_b \succ s_a)\}\log P^i_\psi[o^i_b \succ o^i_a]\Bigg]\Bigg], \\
&= -\mathbb{E}_{(s_a,s_b,y^i)\sim\mathcal{D}^i}\Bigg[conf\{y^i = (s_a \succ s_b)\}\log(sigmoid(\sigma^i_\psi(o^i_a) - \sigma^i_\psi(o^i_b))) + \\
&\quad conf\{y^i = (s_b \succ s_a)\}\log(sigmoid(\sigma^i_\psi(o^i_b) - \sigma^i_\psi(o^i_a)))\Bigg].
\end{aligned} \tag{3}$$

The confidence of ranking s_a higher than s_b in the perspective of agent i, $conf\{y^i = (s_a \succ s_b)\} = \frac{N^i(s_a \succ s_b)}{N_{query}(s_a, s_b)}$, where $N^i(s_a \succ s_b)$ denotes the number of times LLM ranks s_a higher than s_b in the view of agent i, and $N_{query}(s_a, s_b)$ denotes the total number of ranking times on s_a and s_b. The individual reward will be formulated as

$$r_i(s, a_i, s') = \sigma^i_\psi(o'^i) - \sigma^i_\psi(o^i) \tag{4}$$

except the case where the agent i stays still without taking an actual action and the reward will be 0.

This reward function generalizes potential-based rewards from single-agent to multi-agent settings, while maintaining the claims in [Lin et al., 2024] that the RLAIF loss encodes ranking confidence, and that inconsistent rankings, implying that the confidence of two possible ranking results over a state pair are closer, can drive the individual reward towards zero with the loss function of the state-scoring model in Eq. 3. Intuitively, this means that the individual reward functions are robust to ranking errors stemming from high uncertainty when each state-action pair is ranked multiple times.

Additionally, it is unnecessary to train one reward function for each agent if agents are homogeneous with the same individual task. Since these agents take the same, exchangeable role in the team, for a transition (s_a, a, s_b) with encoded observation o^i_a, o^i_b for agent i, there must exist another transition (s'_a, a, s'_b) with encoded observation o'^j_a, o'^j_b for agent j such that $o^i_a = o'^j_a, o^i_b = o'^j_b$. The loss function for agent i's state-scoring model over the preference dataset $\mathcal{D}^i$ can be written as

$$\begin{aligned}\mathcal{L}^i &= -\mathbb{E}_{(s_a,s_b,y^i)\sim\mathcal{D}^i}\Big[\mathbb{I}\{y^i = (o^i_a \succ o^i_b)\}\log P^i_\psi[o^i_a \succ o^i_b] + \mathbb{I}\{y^i = (o^i_b \succ o^i_a)\}\log P^i_\psi[o^i_b \succ o^i_a]\Big],\\ &= -\mathbb{E}_{(s'_a,s'_b,y^i)\sim\mathcal{D}^i}\Big[\mathbb{I}\{y^i = (o'^j_a \succ o'^j_b)\}\log P^i_\psi[o'^j_a \succ o'^j_b] + \mathbb{I}\{y^i = (o'^j_b \succ o'^j_a)\}\log P^i_\psi[o'^j_b \succ o'^j_a]\Big].\end{aligned} \tag{5}$$

If agent i and j share $y^i, \mathcal{D}^i, P^i_\psi$, which means they share the ranking dataset and the state-scoring model, $\mathcal{L}^i$ will be directly transformed to $\mathcal{L}^j$. Therefore, homogeneous agents with the same tasks can be grouped together and share the same reward function. The single reward function can handle the credit assignment among them and gives distinct individual rewards by taking differentiated observations in their own view over the current state. We only need to train different reward functions for heterogeneous agents or homogeneous ones with different pre-assigned tasks. In this case, agents with the same actions may have distinct rewards, which cannot be handled by one single reward model.

4.2 Prompt Designs for Agent-specific State Ranking Reflecting Collaboration

Although we decompose joint state-action rankings into individual rankings, the ranking for each agent is not the same as in a single-agent setting. The LLM reasons from the "ego" agent's perspective but must evaluate collaboration with teammates to assign credit correctly. This section explains how to achieve this via prompt design.

During collaboration, each agent's policy depends on the states and actions of other agents. We design our prompt to make this dependency understandable by LLMs. We consider two types of collaboration dependencies, where the specific type present in an environment is determined by the task content:

1. **Behavior dependence:** Teammates' current state and latest action influence the ego's current action choice.
2. **Task dependence:** The "ego" agent needs to change its task steps according to others' task requirements.

The Two-Switch and Victim-Rubble environments introduced in the experiment section 5.1 are two examples corresponding to these collaboration dependencies respectively. We introduce prompt designs for the above two dependency types separately with these two examples.

4.2.1 *Prompt Design for Behavior Dependence*

In the Two-Switch environment (see Sec. 5.1 for description), the optimal teamwork requires two agents to trigger different switches and unlock the door. Without guidance or communication, the "ego" agent typically observes which switch its teammate approaches and chooses the other. However, if the teammate hesitates and fails to commit, both agents may enter a deadlock, continually adapting to each other's latest action. Such behavior introduces non-stationarity, since from the ego agent's perspective the teammate's policy shifts rapidly as it updates. In addition, this kind of behavior dependency can create sub-optimal policies where one agent is "lazy" and does not contribute to the team's success [Liu et al., 2023].

The agent-specific LLM-generated rankings in our method address these issues by instructing the LLM to **assume teammates follow optimal policies**. This encourages each agent to act optimally without falling into deadlocks or compensating for teammate indecision or laziness. To achieve this, besides offering the team target, key steps, the environment, and current states of all agents, we add the following constraint to our prompt:

Assuming the "teammate" agent will take the best action for the team at this step, *does the current action taken by the "ego" agent help ... from the view of team?*

With this prompt, the LLM understands that it should rank state pairs based on whether the "ego" has made the optimal decision, without being influenced by the teammate's subsequent actions.

4.2.2 *Prompt Design for Task Dependence*

In the Victim-Rubble environment (see Sec. 5.1 for description), optimal teamwork requires two agents to adjust the order of their task steps based on their teammate's needs. For example, if the agents start in the center room, the orange agent should prioritize removing the rubble in the right rooms as it blocks access to a victim which the green agent must heal. To achieve this, besides offering the description of the team target, key steps, the environment and the current states of agents, the prompt should first describe dependencies between agents' tasks:

```
The green agent always prioritizes rescuing victims whose path is free of any rubble
    , waiting for the orange agent to remove rubbles and clear paths.
```

Then describe the current dependency constraints:

```
Rubble1: Chamber5 (8,1), **blocking the only passage** between Chamber5 and 3 from
    the side of Chamber5
Rubble2: Chamber5 (9,2), **blocking the only passage** reaching Chamber4 which
    contains one Victim
```

And also tell LLM which role the "ego" agent takes:

```
You are the orange agent at Chamber3 (4,3)
```

Combining these information, the LLM can identify the next rubble the orange agent should first remove. Part of the example response is as follows:

```
The next step for the orange agent should be to clear the path to Chamber4 so that
    the green agent can rescue the victim.
```

5 Experiments

We tested LCA in three multi-agent collaboration scenarios without inter-agent communication, outer access to policy models, or state transition models. Fig. 2 shows the layouts.

5.1 Experiment Setup

Grid World. We examine two multi-agent collaboration scenarios within Grid World [Swamy et al., 2024]: **Two-Switch** and **Victim-Rubble**.

In the Two-Switch variant, two agents (green and orange triangles) start from random positions in the upper room and at least one should navigate to the target (green rectangle) in the lower room. There is one locked door that blocks the agent's way to the goal and only opens when both switches have been triggered. To unlock the door, the agents must move to the switches, face and trigger them. Therefore, agents are expected to distribute switches between each other, coordinating by observing each other's positions since they cannot communicate.

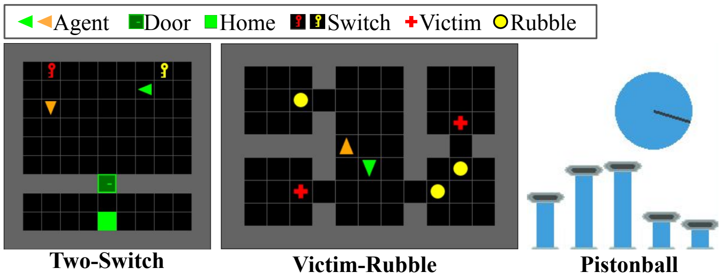

Figure 2: Grid world environments with Two-Switch (left), Victim-Rubble (middle) and Pistonball (right) variants from left to right.

In Victim-Rubble, the green agent must heal all victims (red crosses) and the orange agent must clear all rubble (yellow circles) as fast as possible. One victim lies at the end of a corridor blocked by two rubble pieces; another victim is freely accessible, and one rubble piece blocks nothing.

Pistonball. We also investigate the Pistonball environment from PettingZoo [Terry et al., 2021]. There are five pistons, which are five independent agents, moving vertically to push a ball from the rightmost point to the leftmost in minimal steps.

We compare our approach with the following baselines:

- **MAPPO with the default team reward** This is the vanilla case of MARL where no explicit credit assignment is done. In grid world variants, each agent receives 1 on task completion or handling a switch/victim/rubble, 0 otherwise, with step penalty $-n/n_{max}$, where n is the step count and n_{max} is the episode's maximum time steps. In Pistonball, all earn $+1$ if the ball reaches left, with -0.1 step penalty.
- **MAPPO with the default team reward plus individual rewards** This baseline assigns credits in a naive way with default hand-crafted individual rewards. In the Two-Switch variant, the default individual reward is 1 if the agent triggers a switch / goal. In the Victim-Rubble variant, it is 1 if the orange agent clears rubble or green heals a victim. There are no simple individual rewards in the Pistonball variant, which thus does not have this baseline.
- **QMIX and VDN with the default team reward** These two classical value-decomposition baselines decompose the team reward described above into individual Q values for credit assignment [Rashid et al., 2020, Sunehag et al., 2017].

Team rewards often fail to discourage poor behaviors, while naive individual rewards remain sparse. Our dense rewards address both. Specifically speaking, 1) In Two-Switch: Team rewards grant all agents +1 when a switch is triggered by any one. If the orange agent learns this first, green may remain idle, letting the orange trigger both switches and still earning +2. This increases team steps. Our rewards immediately penalize agents once they act improperly. 2) In Victim-Rubble: Team reward gives both agents $+1$ if green heals an accessible victim, even if orange ignores blocking rubble. Our rewards penalize orange's failure. 3) In Pistonball: Team rewards penalize all pistons if the ball moves right, even if some act correctly. There are no straightforward individual rewards without extensive tuning. Our rewards target only the responsible piston.

These collaborative settings highlight the advantage of dense individual rewards. We use IPPO [Schulman et al., 2017] for policy training with random initialization. We randomly sampled sequential state pairs for potential-based rewards. Since the agents in Two-Switch are homogeneous with the same individual tasks, a single state-scoring model is trained with 4400 state pairs for two agents. Similarly, one state-scoring model is trained with 1000 state pairs and shared by five homogeneous agents in Pistonball. Two state-scoring models are trained for the two heterogeneous agents in Victim-Rubble and each takes 2000 state pairs.

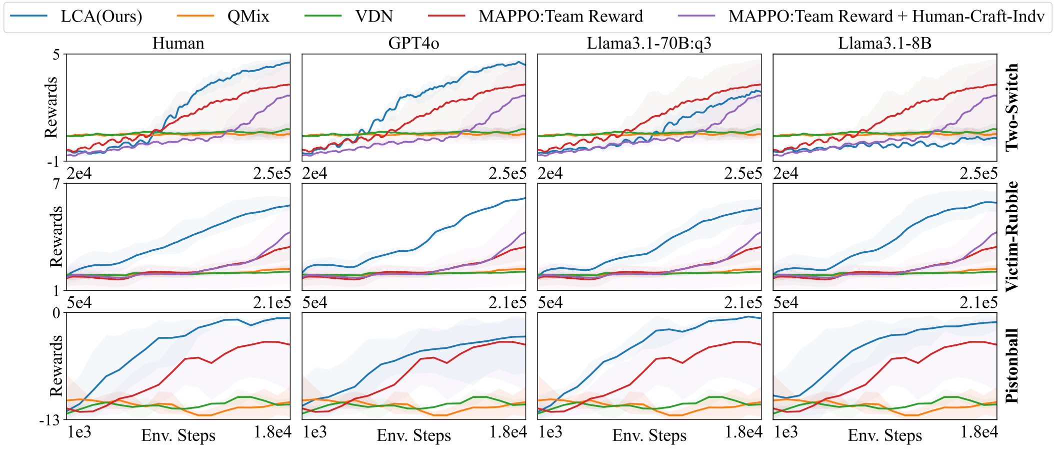

Figure 3: The average learning curves with reward functions trained from single LLM ranking per state pair in the Two-Switch, Victim-Rubble and Pistonball environments over 3 random seeds, with the return variance visualized as shaded areas. The training returns shown on the y-axis are the sum of the default team reward and the individual rewards, indicating that while different methods employ distinct reward models to train policies, their performance is measured under the same reward function for comparability.

5.2 Single-Query Evaluation

We first evaluate the performance of our method using a single query to the LLM to rank each sampled state pair. In each environment, we train our state-scoring models with the human ranking-heuristic function, which serves as an estimated ground-truth ranking based on human heuristics, and evaluate them against 3 LLMs: GPT-4 [Achiam et al., 2023], and two versions of Llama-3.1 [Touvron et al., 2023]—one small and fast version with 70B parameters, referred to as q3_K_M, and another with 8B parameters. Then the potential-difference rewards based on state-scoring models above are employed to train 3 RL policies with random seeds and initializations for each method. The results, as well as the baseline performance, are shown in Fig. 3.

In the Two-Switch variant, our method with human heuristics and GPT4o achieves the optimal return $(5 - step\_penalty)$ in 250k training steps with faster learning speed and less variance than baselines. In this single-query experiment, it is normal to observe that policies trained with the quantized Llama3.1-70B:q3 learn more slowly and the rewards from Llama3.1-8B generating noisy outputs fail to train a useful policy according to [Lin et al., 2024]. They can be further improved with multiple ranking queries per state pair, particularly Llama 3.1-70B:q3, which outperforms the baselines with just two queries, as demonstrated in the next section on multi-query experiments.

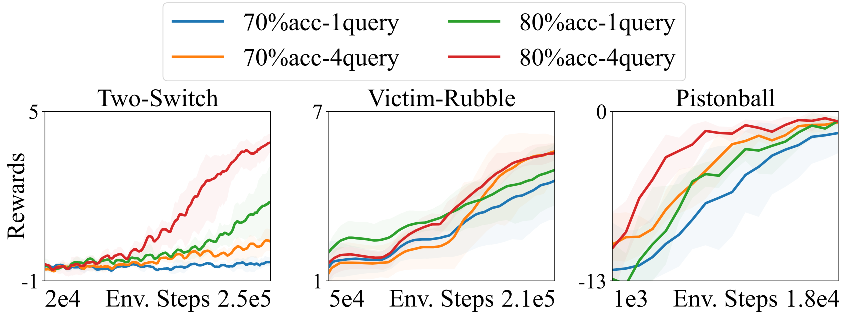

Figure 4: The learning curves with reward functions trained from four-query synthetic experiments over 3 random seeds.

In the Victim-Rubble variant, the default reward easily fails to reach a high return in 210k training steps while LCA with human, GPT4o, Llama3.1-70B:q3 and Llama3.1-8B rankings converges much faster and reaches the optimal reward ($7 - step\_penalty$). GPT4o-reward rollout over an episode in Appendix A shows that LCA effectively decomposes sparse team rewards into dense informative individual rewards. However, the imperfect human ranking heuristic causes our method to learn slightly more slowly than with GPT4o. The human ranking heuristic in this environment forces the green agent to always first save the accessible victim and the orange agent to always first remove the rubbles blocking passages. However, on certain trajectories from suboptimal policies during training, the orange agent may encounter harmless rubble before clearing other rubble, making immediate removal more efficient than returning later. Llama3.1-70B:q3 can have similar ranking flaws. Such ranking flaws may lead to some local optimality and slightly slow down the training speed.

In the Pistonball variant, the baselines fail with the sparse vanilla team reward, while our method with human, Llama3.1-8B and -70B:q3 learn the optimal policy much faster with less variance in 18k training steps. Compared with other LLMs, GPT4o struggles a bit to understand the introduced physical mechanism, so it slows down the training process slightly but still trains some useful policies.

Due to the ϵ-greedy controller PPO methods do not adopt, QMIX and VDN can sometimes start training with a higher return, where $\epsilon = 1$, than IPPO-based LCA and MAPPO. However, sparse rewards cause QMIX and VDN to learn slowly and fail to reach significant returns within LCA's limited training steps, though they learn much faster after a few hundred thousand training steps exceeding LCA training time. We also tried LIIR [Du et al., 2019] and encountered similar consequences, so we ignore its results here.

5.3 Multi-Query Evaluation

This section verifies that our method successfully inherits the robustness of potential-based rewards to noisy preference labels, extending the multi-query approach from single-agent scenarios to MARL. The multi-query approach is to query about ranking over each state pair in the ranking dataset multiple times to handle LLM-ranking inconsistencies for small but fast LLMs generating errors.

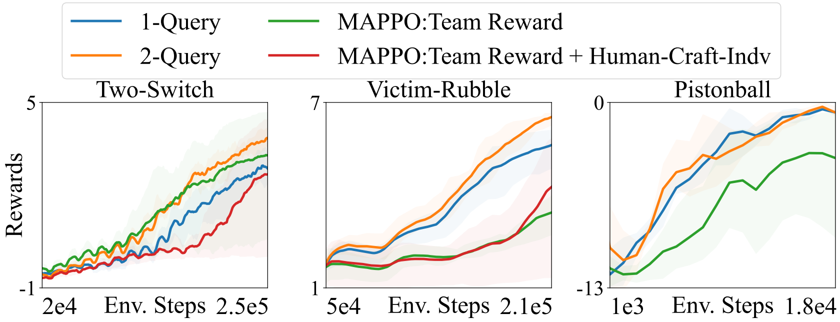

Figure 5: The learning curves with rewards trained from two-query with Llama3.1-70B:q3 over 3 random seeds.

5.3.1 *Synthetic Ranking Evaluation*

To evaluate LCA's robustness, we synthesized ranking datasets with 70% and 80% accuracy and simulated ranking results with four queries per state pair across three environments. These rankings have correctness between 60% (near random guessing) and 90% (high accuracy), providing a comprehensive assessment of LCA's performance. The four-query ranking datasets are synthesized based on four copies of the human-ranking datasets by randomly flipping a specific percentage of rankings. The data are used to train state-scoring models separately, based on which we obtain multi-query potential-based rewards. Fig. 4 shows the resulting policy learning curves averaged over 3 random seeds. We can see the four-query rankings significantly improve the training speed and returns in all environments, especially the Two-Switch variant. In this scenario, the four-query rankings of 80% correctness dramatically raise the training returns to the optimal. The policy with rewards from single-query rankings of 70% correctness fails, while the four-query rankings of 70% accuracy considerably improve the individual reward quality and train some useful policies.

5.3.2 *LLM Two-Query Evaluation*

As discussed above, the q3 version of the Llama3.1-70B is faster and more accessible than the full-sized one but generates more errors and has a flawed performance when training credit-aware individual rewards using a single query per state pair. This section shows that the learning speed can be accelerated with less variance and the training return can be raised to the optimal if using one more query to rank each state pair, as demonstrated by the learning curves in Fig. 5. In the Pistonball environment, since the policy trained with single-query Llama3.1-70B:q3 rankings is already with the fastest learning speed, least variance and optimal training returns, the improvement from the multi-query approach is limited and the two-query variation remains on par with it.

6 Conclusions

This work leverages LLMs to handle the critical challenge of credit assignment in MARL in environments with sparse rewards. This LCA method decomposes sparse team rewards into dense, agent-specific ones by using LLM to evaluate each agent's actions in the contexts of collaboration. The potential-based reward-shaping mechanism mitigates the impact of LLM hallucination, enhancing the robustness and reliability of our method. Our extensive experiments demonstrate that multi-agent collaboration policies trained with our LLM-guided individual rewards achieve faster convergence and higher policy returns compared to state-of-the-art MARL baselines. Experiments also show the resilience of LCA to ranking errors. Therefore, without significant performance degradation, LCA is applicable to smaller and more accessible language models.

7 Limitations and Future Work

Despite its strong performance, LCA has several limitations. Scaling to high-dimensional states increases training complexity, requiring more ranking data and longer LLM query times. It also assumes that LLMs know if a state is better for a specific agent while accounting for its teammates' behavior. However, this can be challenging in robot manipulation and other complex tasks.

LCA is theoretically compatible with pixel-level observations and continuous, multi-modal state spaces, but these scenarios are beyond the scope of this work. It is promising that future extensions integrating vision-language models (VLMs) can generalize LCA effectively to such settings.

Acknowledgement

We would like to acknowledge the support from Honda under grant 58629.1.1012949, from DARPA under ANSR grant FA8750-23-2-1015 and Prime Contract No. HR00112490409, as well as from ONR under CAI grant N00014-23-1-2840.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Serena Booth, W Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 5920–5929, 2023.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE transactions on intelligent transportation systems*, 21(3): 1086–1095, 2019.

Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, 31, 2018.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International conference on machine learning*, pages 3040–3049. PMLR, 2019.

Aditya Kapoor, Benjamin Freed, Howie Choset, and Jeff Schneider. Assigning credit with partial reward decoupling in multi-agent proximal policy optimization, 2024. URL `https://arxiv.org/abs/2408.04295`.

Woojun Kim and Youngchul Sung. An adaptive entropy-regularization framework for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 16829–16852. PMLR, 2023.

Woojun Kim, Whiyoung Jung, Myungsik Cho, and Youngchul Sung. A variational approach to mutual information-based coordination for multi-agent reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pages 40–48, 2023.

W Bradley Knox, Alessandro Allievi, Holger Banzhaf, Felix Schmitt, and Peter Stone. Reward (mis) design for autonomous driving. *Artificial Intelligence*, 316:103829, 2023.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Lu, Thomas Mesnard, Colton Bishop, Victor Carbune, and Abhinav Rastogi. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

Kimin Lee, Laura Smith, and Pieter Abbeel. Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training. *arXiv preprint arXiv:2106.05091*, 2021a.

Kimin Lee, Laura Smith, Anca Dragan, and Pieter Abbeel. B-pref: Benchmarking preference-based reinforcement learning. *arXiv preprint arXiv:2111.03026*, 2021b.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*, 2018.

Muhan Lin, Shuyang Shi, Yue Guo, Behdad Chalaki, Vaishnav Tadiparthi, Ehsan Moradi Pari, Simon Stepputtis, Joseph Campbell, and Katia Sycara. Navigating noisy feedback: Enhancing reinforcement learning with error-prone language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

Boyin Liu, Zhiqiang Pu, Yi Pan, Jianqiang Yi, Yanyan Liang, and Du Zhang. Lazy agents: a new perspective on solving sparse reward problem in multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 21937–21950. PMLR, 2023.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning, 2020. URL `https://arxiv.org/abs/2003.08839`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. *Advances in Neural Information Processing Systems*, 35:9460–9471, 2022.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, pages 5887–5896. PMLR, 2019.

Jianyu Su, Stephen Adams, and Peter A. Beling. Value-decomposition multi-agent actor-critics, 2020. URL `https://arxiv.org/abs/2007.12306`.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Gokul Swamy, Christoph Dann, Rahul Kidambi, Zhiwei Steven Wu, and Alekh Agarwal. A minimaximalist approach to reinforcement learning from human feedback. *arXiv preprint arXiv:2401.04056*, 2024.

J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, Niall Williams, Yashas Lokesh, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15032–15043. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/7ed2d3454c5eea71148b11d0c25104ff-Paper.pdf`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062*, 2020.

Muning Wen, Jakub Grudzien Kuba, Runji Lin, Weinan Zhang, Ying Wen, Jun Wang, and Yaodong Yang. Multi-agent reinforcement learning is a sequence modeling problem, 2022. URL `https://arxiv.org/abs/2205.14953`.

Marco A Wiering et al. Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158, 2000.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games, 2022. URL `https://arxiv.org/abs/2103.01955`.

Alex Zhang, Ananya Parashar, and Dwaipayan Saha. A simple framework for intrinsic reward-shaping for rl using llm feedback, 2024a.

Ruiqi Zhang, Jing Hou, Florian Walter, Shangding Gu, Jiayi Guan, Florian Röhrbein, Yali Du, Panpan Cai, Guang Chen, and Alois Knoll. Multi-agent reinforcement learning for autonomous driving: A survey. *arXiv preprint arXiv:2408.09675*, 2024b.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences, 2020. URL `https://arxiv.org/abs/1909.08593`.

A Individual-Reward Rollout over an Episode

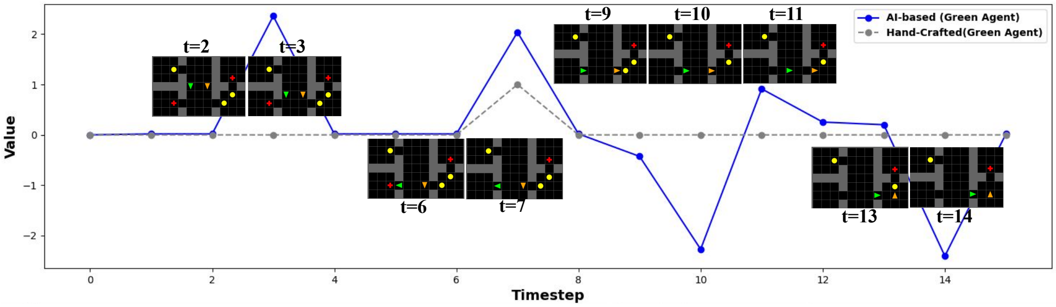

Figure 6: Rolling out individual rewards (blue line) over states of an episode from time steps 0 to 15 in the Victim-Rubble environment. The individual rewards here are the potential-based rewards trained with single-query GPT4o rankings.

We plotted the green agent's individual rewards at states from a continuous episode in the Victim-Rubble environment. The individual rewards here are trained with single-query GPT4o rankings. Compared with the default sparse team reward (grey line), we can see that LCA successfully generates dense individual rewards evaluating individual actions in the contexts of collaboration. Besides giving positive rewards when the green agent makes significant progress (ie. saving victims) like simple hand-crafted reward functions do, LCA also rewards the green agent when it makes a critical turn or movement to the correct target (t=3, 11 in Fig. 6). Meanwhile, LCA individual rewards punish the green agent not only when it takes the wrong action, but also when its teammate makes significant progress but it does nothing special (t=10, 14). It seems that the LLM tends to push the green agent to make progress, effectively avoiding lazy agents.

B Experiment Details for Reproducibility

B.1 Model Architecture

Grid World In both variants, the policy model employs separate actor and critic networks, each composed of three convolutional layers followed by a fully connected layer that maps the flattened feature vector to the output space. The convolutional stack consist of 16 2×2 filters, followed by 2×2 pooling, and finally 32 2×2 filters, and finally 64 2×2 filters. The architectures of the scoring models used in each variant are summarized in Table 1.

	Two-Switch	Victim-Rubble
Conv	conv 16, (2,2), pool (2,2), conv 32, (2,2), conv 64, (2,2)	conv 16, (2,2), conv 32, (2,2), conv 64, (2,2), conv 128, (2,2) conv 256, (2,2)
FC hidden	256, 128, 64, 16	1024, 512, 256, 128, 64, 16

Table 1: Scoring model architectures for the Grid World variants. The table lists the number of output channels and kernel sizes for each convolutional layer, along with the number of nodes in each fully connected hidden layer.

Pistonball Both actor and critic networks of the policy model and the scoring model use a fully connected network with one hidden layer of 64 nodes.

B.2 Hyperparameters

The hyperparameters for training the scoring models and PPO policies were tuned manually, with details summarized in Table 2, 3.

Hyperparameter	Two-Switch	Victim-Rubble	Pistonball
Learning Rate	0.0028	0.0013	0.0003
Batch Size	1024	4096	256
Num. SGD Epochs	4	8	5
Minibatch Size	128	256	32
Clipping Prameter	0.2	0.2	0.2
VF Clip Parameter	10.0	10.0	10.0
VF Coeff.	0.5	0.5	1
KL Coeff.	0.5	0.5	0.2
Entropy Coeff.	0.01	0.02	0.005
GAE	0.8	0.8	0.95
Discount	0.99	0.99	0.99

Table 2: PPO hyperparameters for all agents in all three environments.

Hyperparameter	Two-Switch	Victim-Rubble	Pistonball
SM. LR. Schedule	[0.001, 10] [0.0008, 200]	[0.000004, 20] [0.0000001, 250]	[0.000004, 20], [0.00004, 500]
SM. Batch Size	16	32	16
SM. Epochs	200	250	500

Table 3: Scoring model training hyperparameters for all agents in all three environments. The learning rate schedule is specified by listing each learning rate value alongside the final epoch to which it is applied.

C Full LLM Prompt

LCA adopts the overall prompt structure from Lin et al. [2024]. An example prompt for the Two-Switch environment is as follows, and those for other environments can be accessed here: LCA Prompts.

Preamble

Environment Description:

- Layout: The environment consists of a 9x9 grid divided into 2 chambers. Chamber1 occupies the top 6x9 section, and Chamber2 spans the bottom 2x9 section of the grid. A passage connects Chamber1 and Chamber2. A door at Chamber1 connects Chamber1 and Chamber2. Two agents are required to move to a clinic in Chamber2 this environment as soon as possible. Every time the Agent can only move for one grid in one of four directions, up/down/left/right.
- Coordinate System: In this 9x9 grid, points are labeled (x, y), with (0, 0) at the bottom left and (9, 9) at the top right. Therefore, the coordinates span from (0,3) to (8,8) in Chamber1, from the bottom left corner to the upper right corner of this chamber ((8,8) is inclusive in this chamber). The coordinates span from (0,0) to (8,2) from the bottom left corner to the upper right corner of Chamber2 ((8,2) is inclusive in this chamber).
- Mechanics:
- Door: The door to Chamber2 unlocks only when both of the two keys in Chamber1 have been triggered. A key can only be triggered when an agent is one-step away from it. To save time, we expect two agents to move to two different keys separately. If both keys have not been triggered in the given state, you should determine which key the agent I query about should move to for the sake of this team, and then evaluate whether the agent is moving closer to it. If only one key has not been triggered, only the agent closer to this key should move to this key, while the other agent should move to the door, preparing to get through the passage.

Sample to annotate

Q:
State[a]:
The "ego" agent: Chamber1 (5,8)
The "teammate" agent: Chamber1 (5,7)
Clinic: Chamber2 (4,0)
Passage to Chamber2: Chamber1 (4,2)
Door at Chamber1 (4,2): locked

Redkey: Chamber1 (1,8)
4 steps between Redkey and the "ego" agent
5 steps between Redkey and the "teammate" agent

Yellowkey: Chamber1 (7,8)
2 steps between Yellowkey and the "ego" agent
3 steps between Yellowkey and the "teammate" agent

No keys have been triggered.
The "ego" agent's action: moved to (6,8)

Ending

Assuming the "teammate" agent will take the best action for the team at this step, does the action taken by the "ego" agent in State[a] help it save the future steps for progressing towards the goal from the view of team? (Yes/No)

A: Let's think step by step.

The LLM should be able to give the reasoning process similar to this one:

Few-shot exemplar

Q:
State[a]:
The "ego" agent: Chamber1 (5,8)
The "teammate" agent: Chamber1 (5,7)
Clinic: Chamber2 (4,0)
Passage to Chamber2: Chamber1 (4,2)
Door at Chamber1 (4,2): locked

Redkey: Chamber1 (1,8)
4 steps between Redkey and the "ego" agent
5 steps between Redkey and the "teammate" agent

Yellowkey: Chamber1 (7,8)
2 steps between Yellowkey and the "ego" agent
3 steps between Yellowkey and the "teammate" agent

No keys have been triggered.
The "ego" agent's action: moved to (6,8)

Assuming the "teammate" agent will take the best action for the team at this step, does the action taken by the "ego" agent in State[a] help it save the future steps for progressing towards the goal from the view of team? (Yes/No)

A: Let's think step by step.

To determine if the action taken by the "ego" agent helps save future steps for progressing towards the goal from the view of the team, let's analyze the situation:

1. **Current State**: The "ego" agent is at (5,8) in Chamber1, and the teammate is right below at (5,7). The clinic is located at (4,0) in Chamber2, which means both agents need to move down through the passage at (4,2) once the door is unlocked. The red key is at (1,8), and the yellow key is at (7,8).

2. **Goal**: Unlock the door by triggering both keys and then proceed to the clinic in Chamber2.

3. **Analysis**:
- Considering the positions of the keys and the need for both agents to move efficiently towards them, we can infer a strategy where one agent goes for the red key and the other for the yellow key. So there are two possible plans for these two agents in State[a]:

Plan 1: the "ego" agent moves to Redkey (4 steps), and the "teammate" agent moves to Yellowkey (3 steps). Given two agents act at the same time, the time for triggering both keys will be 4.
Plan 2: the "ego" agent moves to Yellowkey (2 steps), and the "teammate" agent moves to Redkey (5 steps). Given two agents act at the same time, the time for triggering both keys will be 5.

Since Plan 1 needs less time, the optimal choice now is that the "ego" agent moves to Redkey, and the "teammate" agent moves to Yellowkey.

- Given the "ego" agent's initial position at (5,8), moving right could be seen as aligning with heading towards the yellow key at (7,8). This is inconsistent with the optimal choice.

5. **Conclusion**: No, the action taken by the "ego" agent in State[a] does not help save future steps for progressing towards the goal from the view of the team.