Improved Multi-Agent Collaboration with Multi-Turn Reinforcement Learning

Shuo Liu
Khoury College of Computer Sciences
Northeastern University
Boston, MA, 02115, USA
liu.shuo2@northeastern.edu

Tianle Chen
Khoury College of Computer Sciences
Northeastern University
Boston, MA, 02115, USA
chen.tianle@northeastern.edu

Christopher Amato
Khoury College of Computer Sciences
Northeastern University
Boston, MA, 02115, USA
c.amato@northeastern.edu

Abstract

A large amount of work has been done in Multi-Agent Systems (MAS) for modeling and solving problems with multiple interacting agents. However, most LLMs are pretrained independently and not specifically optimized for coordination. Existing LLM fine-tuning frameworks rely on individual rewards, which require complex reward designs for each agent to encourage collaboration. To address these challenges, we model LLM collaboration as a cooperative Multi-Agent Reinforcement Learning (MARL) problem. We develop a multi-agent, multi-turn algorithm, Multi-Agent Group Relative Policy Optimization (MAGRPO), to solve it, building on current RL approaches for LLMs as well as MARL techniques. Our experiments on coding collaboration demonstrate that fine-tuning MAS with MAGRPO enables agents to generate high-quality responses efficiently through effective cooperation. Our approach opens the door to using other MARL methods for LLMs and highlights the associated challenges.

1 Introduction

Leveraging billions of parameters and extensive pre-training on large-scale datasets, state-of-the-art LLMs have demonstrated remarkable capabilities across diverse domains Grattafiori et al. [2024], Achiam et al. [2023], Anil et al. [2025]. To adapt to specific applications or align with human preferences, fine-tuning has emerged as a critical secondary training stage. Compared to supervised fine-tuning, Reinforcement Learning (RL) enables more generalizable learning for complex, multi-turn tasks through human-aligned reward design, making it an important technique for fine-tuning Ouyang et al. [2022], Guo et al. [2025], Ziegler et al. [2020].

Likewise, Multi-Agent Systems (MAS) have been extensively studied over the past decades, with substantial progress in modeling and solving problems involving multiple agents Littman [1994], Shoham and Leyton-Brown [2009], Stone and Veloso [2000]. In particular, advances in cooperative MAS have demonstrated strong potential for enabling effective collaboration in distributed settings, such as games, robotics, and traffic control Samvelyan et al. [2019], Vinyals et al. [2017], Berner et al. [2019], Amato et al. [2016], Wiering [2000], Liu et al. [2022]. These developments motivate the application of MAS principles and techniques to LLM collaboration, where multiple LLMs working together can solve more complex tasks in a more robust and efficient manner.

Multi-Turn Interactions in Large Language Models Workshop NeurIPS 2025

There has been some recent work on coordinating multiple LLMs. Some approaches implement coordination at the inference stage, enabling agents to interact through debate, discussion, or verification Du et al. [2023], Wu et al. [2023a], Lifshitz et al. [2025]. These methods operate at the prompt level, with fixed models that are not tuned toward coordination-centric objectives. The agents may have conflicting answers or spread incorrect information to other participants, limiting performance Cemri et al. [2025], Estornell and Liu [2024]. Moreover, the design of effective prompts remains difficult and unclear. Other approaches fine-tune agents independently with individual or role-conditioned rewards. However, they require carefully curated rewards for each individual or role Slumbers et al. [2024], Liu et al. [2025a], Subramaniam et al. [2025], and, as independent learning methods, lack convergence guarantees Tan [1993].

In this paper, we model LLM collaboration as a cooperative MARL problem Albrecht et al. [2024] and formalize it as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) Oliehoek and Amato [2016]. In LLM collaboration, multiple trainable LLMs generate responses synchronously based on their individual prompts. The external environment evolves according to the joint responses until the dialog ends. This general model allows a wide range of problems to be modeled and solved using versions of MARL algorithms. Following the efficient practice of Group Relative Policy Optimization (GRPO) Guo et al. [2025], we propose Multi-Agent GRPO (MAGRPO) that trains LLMs in a multi-turn setting. MAGRPO leverages centralized group-relative advantages for joint optimization, while preserving decentralized execution for each agent. Our method builds off of state-of-the-art LLM approaches in GRPO and MARL approaches for centralized training and decentralized execution, such as MAPPO Yu et al. [2022]. Our experiments show that MAGRPO develops various cooperation schemes, improving response efficiency with high quality.

Our contributions can be summarized as follows: (i) We model the LLM collaboration as a cooperative MARL problem, where multiple LLMs cooperate to generate joint responses; (ii) We implement the MAGRPO algorithm, which optimizes agent cooperation through aligned rewards while maintaining decentralized execution to maintain efficiency; (iii) Our experiments demonstrate that fine-tuning with MAGRPO improves both response efficiency and quality in coding collaboration; (iv) We provide a detailed analysis of the limitations of existing approaches and outline open challenges in applying MARL to LLM collaboration.

2 Related Work

2.1 Test-Time Multi-Agent Interaction

Recent work employs multiple agents with specialized roles interacting through diverse pipelines at test-time to enhance response quality. In multi-agent debate, agents iteratively formulate positions by reviewing other agents' outputs, where the final decision or answer is determined by majority voting or a summarizer Du et al. [2023], Chan et al. [2023], Liang et al. [2024]. Role-based approaches allocate tasks across specialized agents Wu et al. [2023a], Qian et al. [2024], Hong et al. [2024]. For example, an agent may function as a verifier to assess the correctness of outputs Skreta et al. [2023], Lifshitz et al. [2025], Setlur et al. [2025], while another may act as a macro-planner to orchestrate workers' responses. However, these multi-agent frameworks rely on prompt-level interactions among agents, often leading to ineffective communication and computational inefficiency. Moreover, the design of effective prompts and role assignment remains unclear, as prompts usually fail to reliably guide agent behavior, enforce role adherence, or support coherent coordination across tasks. These limitations motivate us to fine-tune LLMs in MAS to improve their cooperation.

2.2 Multi-Agent Fine-Tuning

Recent work has explored fine-tuning LLMs to improve their performance across diverse domains, e.g., arithmetic reasoning, navigation, and hidden-role games Ma et al. [2025], Slumbers et al. [2024], Sarkar et al. [2025]. These approaches typically employ individual rewards or rewards conditioned on specific roles Park et al. [2025], Liu et al. [2025a], Subramaniam et al. [2025]. Such reward structures often require careful manual specification, and their underlying rationale is rarely well justified. The misaligned or conflicting incentives can hinder effective coordination. Moreover, these methods lack convergence guarantees, as each agent learns independently in a non-stationary environment where other agents are simultaneously updating their policies. In this paper, we focus on cooperative scenarios, where LLMs are jointly trained with interpretable, human-aligned rewards.

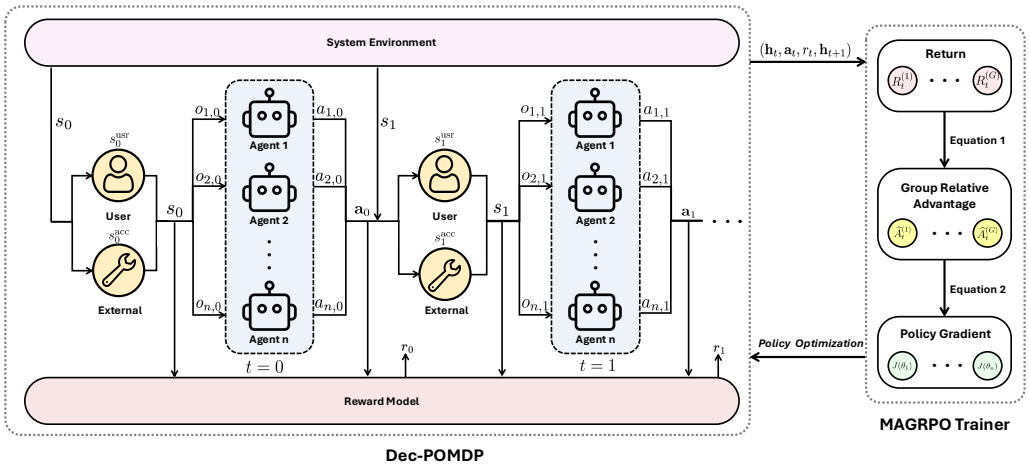


Figure 1: Illustration of Dec-POMDP and our MAGRPO algorithm.

3 Cooperative MARL for LLM Collaboration

Since LLMs can be viewed as a special class of agents, we leverage advances in MAS to facilitate their collaboration. We model LLM collaboration as a cooperative MARL problem and outline its unique challenges. We formalize this problem as a Dec-POMDP, as shown in Figure 1.

3.1 LLM Collaboration

LLM collaboration refers to the problems where LLMs cooperatively solve a class of tasks in MAS. The tasks are specified in natural language and provided to the agents as prompts. Each LLM agent generates a response synchronously based on its individual instructions. The set of these responses jointly forms a solution to the task.

Most tasks cannot be resolved in one turn. Users, external models, or systems validate the solutions and provide additional requirements or suggestions for LLMs. These components also serve as part of the environment for LLM collaboration, whose states may change based on the agents' outputs. The updates are embedded into prompts for subsequent turns. This iterative process continues until the task is successfully completed or a predefined turn limit is reached.

As discussed by a number of companies NVIDIA [2024], Anthropic [2024], a team of agents could be used to generate a complex codebase. The code would be difficult, costly, and time-consuming to generate with a single agent, but a group of LLMs could do so quickly and cheaply. None of these agents is self-interested, but they are trainable in a scheme such as the one discussed below. Using a joint reward allows agents to specialize as needed to complete the task without complex prompt or reward engineering.

3.2 Problem Formalization

We formalize collaboration among LLMs as a subclass of the cooperative MARL problem, considering LLM agents and the types of problems they are solving. This problem is a form of a Dec-POMDP Oliehoek and Amato [2016], which allows cooperation through a joint reward while preserving scalable decentralized control.

Mathematically, our LLM Dec-POMDP is defined by a tuple $\langle \mathcal{I}, \mathcal{S}, \{\mathcal{O}_i\}, \{\mathcal{A}_i\}, R, T, H \rangle$.

- $\mathcal{I} = \{1, \cdots, n\}$ denotes the set of n LLM agents, each instantiated with a pre-trained language model.
- $\mathcal{S}$ denotes the full global state space. At turn t, a full state $s_t = (s_t^{\text{acc}}, s_t^{\text{usr}})$ consists of parts that are accessible in the model and provided to the reward model $s_t^{\text{acc}} \in \mathcal{S}^{\text{acc}}$ (e.g., external models or systems), and the inaccessible user state $s_t^{\text{usr}} \in \mathcal{S}^{\text{usr}}$ which updates over time

but isn't maintainable. In Dec-POMDP, the state is not directly observable by the agents (LLMs).

- $\mathcal{O}_i$ is the observation space for agent i with $\mathcal{O} = \times_i \mathcal{O}_i$ the joint observation space. A local observation $o_{i,t}$ consists of natural language instructions (i.e., prompts), providing a partial and noisy view of s_t.
- $\mathcal{A}_i$ is the action space for agent i with $\mathcal{A} = \times_i \mathcal{A}_i$ the joint action space. A local action $a_{i,t}$ is a response in natural language to the given prompt.
- $R : \mathcal{S}^{\text{acc}} \times \mathcal{A} \to \mathbb{R}$ is the joint reward function implemented via predefined rules or a pretrained reward model. At turn t, the joint rewards r_t are determined by the accessible part of current state s_t^{acc} and the agents' joint action $\mathbf{a}_t = \{a_{1,t}, \cdots, a_{n,t}\}$.
- $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the underlying stochastic state transition function. At turn t, the agents' joint actions $\mathbf{a}_t$ induce a shift to a new state $s_{t+1} \sim T(\cdot|s_t, \mathbf{a}_t)$, which reflects the updates in the user state and the states of external models and systems.
- H is the episode horizon, i.e., the turn limit of the dialog.

In Dec-POMDP, since the states are not directly observed, each agent maintains its local observation-action history $\mathbf{h} = \{h_1, \cdots, h_n\}$ to infer information about state. A solution to a Dec-POMDP is a joint policy that maximizes the expected cumulative reward, $\boldsymbol{\pi}^* = \{\pi_1^*, \cdots, \pi_n^*\} = \arg\max_{\boldsymbol{\pi}} \mathbb{E}_{\boldsymbol{\pi}} \left[\sum_{t=0}^{H-1} R(s_t^{\text{acc}}, \mathbf{a}_t)\right]$. A joint policy is a set of local policies π_i, which conditions on the local observation-action history $h_{i,t} = \{o_{i,0}, a_{i,0}, \ldots, o_{i,t}\}$.

RL methods for Dec-POMDPs have become a popular topic (e.g., Foerster et al. [2024], Lowe et al. [2020], Foerster et al. [2018], Rashid et al. [2018], Wang et al. [2021], Yu et al. [2022], Albrecht et al. [2024]) with methods successful at scaling to large state, action and observation spaces. Many methods use Centralized Training for Decentralized Execution (CTDE), where they use some centralized information during training (e.g., a centralized value function estimate) but are still able to execute in a decentralized manner when training is complete.

3.3 Challenges in LLM Collaboration

LLM collaboration presents unique challenges compared to traditional MARL problems, where LLM agents receive and process tasks through natural language.

3.3.1 Representations in Natural Language

Unlike traditional cooperative MARL agents, LLM agents operate over natural language, receiving instructions and generating responses as sequences of tokens. MARL approaches could model this problem at the token or prompt/response level. At the token level, the number of actions and observations is smaller, but the problem horizon can be very long. At the prompt/response level, the actions and observations space is much larger, but the horizon is much shorter. Moreover, token-level rewards are often uninformative, as both queries and responses must form coherent and semantically meaningful structures. As adopted in prior RL methods Ouyang et al. [2022], Guo et al. [2025], Rafailov et al. [2024], we model each LLM agent's decision-making process as a direct mapping from input instructions to complete responses to enable efficient and stable training. Nevertheless, the best modeling and solution approaches remain an open question.

3.3.2 Training Paradigm

As mentioned above, many MARL methods use centralized training for decentralized execution (CTDE). Unfortunately, standard CTDE methods use centralized value models in the form of centralized critics Foerster et al. [2024], Lowe et al. [2020], Yu et al. [2022] or mixers in value decomposition methods Rashid et al. [2018], Wang et al. [2021]. Such architectures allow additional information and coordination during training but do not scale well to very large action and observation spaces (such as those in our problem). Conversely, Decentralized Training and Execution (DTE) methods Amato [2025] train a set of models, one for each agent in a decentralized manner. DTE approaches are typically more scalable but don't use additional information during training (even when it is available). It is an open question which paradigm to use to maximize performance while maintaining scalability in the LLM collaboration problem. In this paper, we balance decentralized execution with

Algorithm 1: MAGRPO

Require: Dataset $\mathcal{D}$, n pretrained LLMs with policies $\{\pi_{\theta_1}, \cdots, \pi_{\theta_n}\}$, reward model R, generation group size G, learning rate α

for each episode **do**
  Sample a task $\sim \mathcal{D}$
  Initialize observations $o_{i,0}, \forall i \in \mathcal{I}$, according to the task, and $\mathbf{o}_0 = \{o_{1,0}, \cdots, o_{n,0}\}$
  $h^{\mathcal{G}}_{i,0} \leftarrow o_{i,0}, \forall i \in \mathcal{I}$, and $\mathbf{h}^{\mathcal{G}}_0 = \{h^{\mathcal{G}}_{1,0}, \cdots, h^{\mathcal{G}}_{n,0}\}$
  for turn $t = 0$ to $H-1$ **do**
    Generate a group of responses $a^{\mathcal{G}}_{i,t} \leftarrow \pi_{\theta_i}(\cdot|h^{\mathcal{G}}_{i,t}), \forall i \in \mathcal{I}$, where $h^{\mathcal{G}}_{i,t} = \{h^{(1)}_{i,t}, \cdots, h^{(G)}_{i,t}\}$, $a^{\mathcal{G}}_{i,t} = \{a^{(1)}_{i,t}, \cdots, a^{(G)}_{i,t}\}$, and $\mathbf{a}^{\mathcal{G}}_0 = \{a^{\mathcal{G}}_{1,t}, \cdots, a^{\mathcal{G}}_{n,t}\}$
    Obtain a joint reward $r^{\mathcal{G}}_t$ from system
    Receive new observations $o^{\mathcal{G}}_{i,t+1}$, and update history $h^{\mathcal{G}}_{i,t+1} \leftarrow \{h^{\mathcal{G}}_{i,t}, a^{\mathcal{G}}_{i,t}, o^{\mathcal{G}}_{i,t+1}\}, \forall i \in \mathcal{I}$
  end for
  for turn $t = H-1$ to 0 **do**
    Calculate return $R^{(g)}_t \leftarrow \sum_{\tau=t}^{H-1} r^{(g)}_\tau, \forall g \in \mathcal{G}$
    Estimate $\widehat{A}^{(g)}_t, \forall g \in \mathcal{G}$ for each branch B according to Equation 1
    Calculate $J(\theta_i), \forall i \in \mathcal{I}$ for each branch B according to Equation 2
    $\theta_i \leftarrow \theta_i + \alpha \nabla_{\theta_i} J(\theta_i), \forall i \in \mathcal{I}$
  end for
end for
return $\boldsymbol{\pi_\theta} = \{\pi_{\theta_1}, \cdots, \pi_{\theta_n}\}$

centralized training using group-based Monte Carlo estimates. Experiments show the effectiveness of our approach on short-horizon tasks.

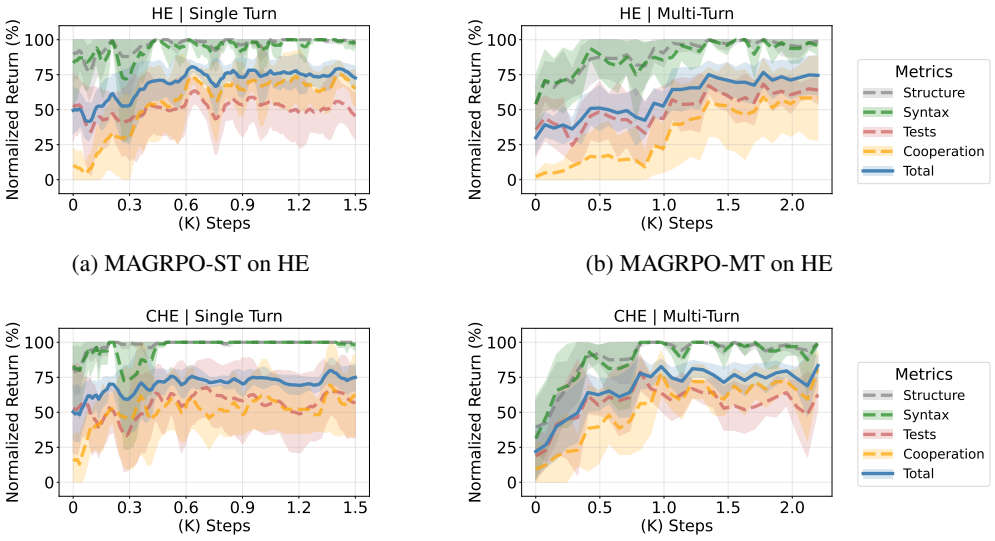
4 MAGRPO

We propose the Multi-Agent GRPO (MAGRPO) algorithm to jointly train LLM agents in MAS while maintaining decentralized execution.

Algorithm 1 shows the procedure of MAGRPO. Given a dataset $\mathcal{D}$ containing task information (e.g., the descriptions of coding problems), n LLMs are optimized, each with a policy parameterized by θ_i and guided by a reward model R. In each episode, a task is sampled from the given dataset $\mathcal{D}$, which is used to construct initial observations $\mathbf{o}_0 = \{o_{1,0}, \cdots, o_{n,0}\}$ and histories $\mathbf{h}_0 = \{h_{1,0}, \cdots, h_{n,0}\}$. Taking inspiration from the single-agent GRPO algorithm Guo et al. [2025], Liu et al. [2025b], at each turn t, each agent takes action by generating a group of responses $a^{\mathcal{G}}_{i,t} = \{a^{(1)}_{i,t}, \cdots, a^{(G)}_{i,t}\}$ following its policy $\pi_i(\cdot|h^{\mathcal{G}}_{i,t})$ based on its observation-action history $h^{\mathcal{G}}_{i,t} = \{h^{(1)}_{i,t}, \cdots, h^{(G)}_{i,t}\}$. The actions of individual agents are aggregated to form a group of joint actions $\mathbf{a}^{\mathcal{G}}_t = \{a^{\mathcal{G}}_{0,t}, \cdots, a^{\mathcal{G}}_{n,t}\}$. The agents receive a group of joint rewards $r^{\mathcal{G}}_t$ for their responses $\mathbf{a}^{\mathcal{G}}_t$), which also conditions on the accessible part of the state $R(\cdot|s^{\text{acc},\mathcal{G}}_t, \mathbf{a}^{\mathcal{G}}_t)$. The joint actions triggers the transition $T(\cdot|s^{\mathcal{G}}_t, \mathbf{a}^{\mathcal{G}}_t)$, where agents receive new observations $o^{\mathcal{G}}_{i,t+1} = \{o^{(1)}_{i,t+1}, \cdots, o^{(G)}_{i,t+1}\}$ and use them to construct histories $h^{\mathcal{G}}_{i,t+1} = \{h^{\mathcal{G}}_{i,t}, a^{\mathcal{G}}_{i,t}, o^{\mathcal{G}}_{i,t+1}\}$. This process continues until terminated at turn H.

We employ stochastic gradient descent to train agents. Without explicit value models, estimating history-action values from a single rollout incurs high variance. To stabilize training, we estimate the expected return of the current state by averaging over a group of Monte Carlo samples $\{R^{(1)}_t, \cdots, R^{(G)}_t\}$ for each branch B. As a result, we are able to generate a centralized estimate (which is common in MARL) without a large value model. For each turn t, the advantage of each joint action in the group is calculated as,

$$\widehat{A}^{(g)}_t = R^{(g)}_t - \frac{1}{G} \sum_{g=1}^{|\mathcal{G}|} R^{(g)}_t, \tag{1}$$



(a) MAGRPO-ST on HE

(b) MAGRPO-MT on HE

(c) MAGRPO-ST on CHE

(d) MAGRPO-MT on CHE

Figure 2: Normalized returns: (a) Structural integrity (dashed grey); (b) Syntax correctness (dashed green); (c) Test pass rate (dashed red); (d) Cooperation (dashed yellow); (e) Total return (solid blue).

where $R_t^{(g)} = \sum_{\tau=t}^{H-1} r_\tau^{(g)}$. Inspired by GRPO Guo et al. [2025], Dr. GRPO Liu et al. [2025b], and MAPPO Yu et al. [2022], the centralized advantage values can be used to update policy π_i (parameterized by θ_i) for each agent i. MAGRPO does not have importance sampling and epsilon clipping since it is on-policy, and the weight of the KL divergence term is set to be 0 to encourage greater policy deviation from the base model,

$$J(\theta_i) = \mathbb{E}_{\mathbf{o}_0 \sim \mathcal{D}, \mathbf{h}^{\mathcal{G}} \sim \boldsymbol{\pi}_{\boldsymbol{\theta}}} \left[\frac{1}{|B|} \frac{1}{|\mathcal{G}|} \sum_{h_i^{\mathcal{G}} \in B} \sum_{g \in \mathcal{G}} \widehat{A}_t^{(g)} \log \pi_{\theta_i}(a_{i,t}^{(g)} | h_{i,t}^{\mathcal{G}}) \right], \tag{2}$$

where the gradient is averaged across all the branches (states) and over the group of generations at t.

5 Experiments

In large-scale software development, numerous developers cooperate to implement complex systems. Employing LLMs as developers is a promising direction, but coordinating them remains challenging due to diverse cooperation schemes and complex failures. In our experiments, we frame the coding collaboration task by having 2 *Qwen2.5-Coder-3B* agents to generate Python functions together, where a helper agent produces auxiliary functions to support a main function generator without direct communication. The outputs from both agents, along with the required libraries, are aggregated into complete code snippets.

5.1 Datasets

We first evaluate MAGRPO on the HumanEval (HE) dataset. HE comprises 164 handwritten programming problems, each containing a natural language description (`prompt`), a function signature (`entry_point`), and a set of unit tests (`test`). To guide learning, we design a level-based reward model that prioritizes fundamental aspects of code generation. Structural integrity verifies the presence and correctness of both main and auxiliary function definitions; syntactic correctness ensures compliance with Python syntax; test pass rate assesses functional correctness based on the proportion of successfully passed unit tests; and a cooperation quality bonus is granted when the main function properly invokes and utilizes the auxiliary function. The rewards are accumulated only when all requirements at each preceding level are satisfied.

However, most problems in HE are not designed for coding collaboration, as certain atomic operations (e.g., `strlen(string)`) can hardly be decomposed in a way that supports meaningful cooperation.

These noisy instances bring instability into the training process or bias it toward inefficient cooperation schemes, such as the main agent merely wrapping the auxiliary function. Thus, we construct a cooperation-oriented code generation dataset, CoopHumanEval (CHE), for our evaluation. CHE includes both original HE problems with the potential for cooperation (e.g., `prime_fib(n)`) and additional handwritten programming problems (e.g., `compare_areas(shapes)`). Its data fields are the same as HE, with problem descriptions in `prompt`, function signatures in `entry_point`, and designed unit tests provided in `test`. The problems in CHE are readily decomposable, and agents can explore more effective cooperation schemes by training with this dataset. Both datasets are split into training and testing sets with a ratio of 25:8.

5.2 Baselines

We adopt the fixed and fine-tuned single model, along with three multi-agent methods built on fixed base models, as our baselines. For the single-agent setting, the *Qwen2.5-Coder-3B* model generates a Python function based on the problem description in the `prompt`, with the function name specified in `entry_point`. We also fine-tune this model on the training set to adapt it to the task. In the multi-agent setting, two *Qwen2.5-Coder-3B* models serve as agents: one generates an auxiliary function, and the other produces the main function. To minimize the influence of prompts on our comparison, we keep the problem description fixed and only add minimal coordination instructions. Specifically, in the naive concatenation scheme, agents are simply informed of their roles and generate outputs in parallel without communication. The sequential pipeline introduces one-way communication, allowing the main agent to respond based on the auxiliary agent's output. The one-round discussion baseline enables bidirectional communication: agents first receive the same prompts as in naive concatenation, then the prompts are augmented with the other's first-turn response in the second turn.

5.3 Results

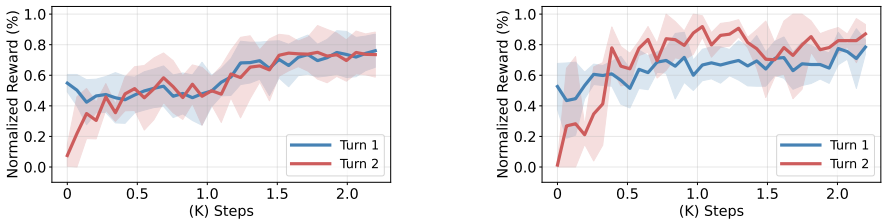
We optimize the interaction between 2 agents with MAGRPO in both single-turn and multi-turn settings, i.e., MAGRPO-ST and MAGRPO-MT. To reduce prompt-induced variance, agents are informed only of the problem description and their roles, using the same initial prompt as in naive concatenation. In MAGRPO-MT, the agents' previous responses are given to an external *Claude-Sonnet-4* model, which provides each agent with feedback comprising functionality analysis, error detection, and revision suggestions.

The performance of MAGRPO-ST and MAGRPO-MT on HE is shown in Figures 2a and 2b. Although MAGRPO-ST improves the syntactical correctness and develops valid cooperation, its test pass rate does not show much progress. As for MAGRPO-MT, agents are initially overwhelmed by the external model's feedback, resulting in even lower initial returns. They gradually adopt the suggestions and improve their returns. However, the improvement in test pass rate is still limited due to noisy entries in the dataset and unreliable feedback. This reflects the complexity and delicacy of coder coordination, where the main agent must accurately infer the functionality of auxiliary modules and trust their correctness without direct communication. As shown in Figures 2c and 2d, MAGRPO-ST and MAGRPO-MT achieve higher overall returns and lower variances when trained on CHE. In MAGRPO-MT, although agents initially struggle to interpret the feedback, the normalized returns gradually increase and eventually surpass those of single-turn training. This indicates that, when trained on a dataset with well-defined cooperative structures and guided by reliable suggestions, agents can learn to incorporate feedback effectively and improve the quality of their responses.

Table 1 presents a performance comparison between MAGRPO and baselines on HE and CHE. Speed is measured in tokens per second on a GeForce RTX 5090, and pass@k is shown in percentage. By GRPO fine-tuning, the performance of *Qwen2.5-Coder-3B* model only improves slightly as the logic of test problems differs substantially from that in the training set. Although the naive concatenation method exhibits high generation speed, it has lower test pass rates than a single model, as the main agent may rely on incorrect assumptions about the auxiliary function. In the sequential pipeline, the main agent can access the auxiliary function and compensate for its weaknesses, improving robustness but at the cost of slower inference. The one-round discussion method involves more communication between agents, but its effectiveness remains limited due to the potential misaligned cross-adaptation issue. MAGRPO-ST incorporates diverse cooperation schemes that improve robustness, thereby obtaining higher unit pass rates. In MAGRPO-MT, an external model can provide additional feedback

Method	Speed		Pass@1		Pass@3		Pass@5		Pass@10	
	HE	CHE	HE	CHE	HE	CHE	HE	CHE	HE	CHE
Fixed Single Model	113.8	125.6	38.7	50.0	54.8	56.3	61.2	62.5	67.7	75.0
Fine-Tuned Single Model	114.9	124.8	45.1	62.5	64.5	68.8	67.7	75.0	70.9	81.0
Naive Concatenation	**194.9**	189.4	42.5	40.1	45.2	43.8	50.6	56.3	64.5	59.2
Sequential Pipeline	99.6	97.4	53.4	57.2	54.8	73.6	62.3	**82.7**	71.2	86.2
One-Round Discussion	82.5	78.3	41.2	40.9	51.6	52.9	61.1	60.3	70.8	75.3
MAGRPO-ST (Ours)	190.0	**192.4**	54.8	68.2	55.6	71.4	58.1	75.4	71.6	81.2
MAGRPO-MT (Ours)	95.2	97.3	**67.9**	**73.2**	**71.0**	**75.0**	**80.6**	81.3	**90.3**	**87.5**

Table 1: Performance of MAGRPO and baselines on speed (tokens/s) and pass@k. Results are averaged over 10 runs; **bold** indicates the best performance for each metric on each dataset.



(a) Self-Evolving

(b) Guided-Learning

Figure 3: MAGRPO-MT rewards in 2 learning modes: (a) Turn 1 (blue); (b) Turn 2 (red).

that further strengthens cooperation. As a result, MAGRPO-MT achieves the highest pass rates and total return across most pass@k metrics.

5.4 Cooperation Schemes and Learning Modes

MAGRPO identifies various cooperation schemes. For example, the auxiliary function handles the core logic, while the main agent adds backup logic or decorations to improve the overall solution. Alternatively, the main agent may act as a coordinator, decomposing the problem and assigning subtasks to the auxiliary agent. In addition, the auxiliary function may also serve as a strategy filter, guiding the main agent to generate code for specific cases. While coordinator and strategy-filter schemes can improve inference efficiency, they are more prone to syntax and logical errors. With limited cooperation-oriented training data, the main agent typically resorts to more conservative roles, i.e., fallback or decoration. Note that these cooperation schemes emerge during training under a relatively simple joint reward, more refined design patterns could be found when training agents to develop large-scale software.

Agents can learn to cooperate through various modes in the multi-turn setting. Figure 3a demonstrates a self-evolving mode, where agents primarily evolve with the tasks themselves without external feedback. At the beginning, the second-turn rewards (red) are lower than the first-turn rewards (blue), since agents struggle to incorporate previous responses effectively. Both curves gradually improve as agents develop cooperative behaviors, but the performance of the second turn is consistently similar to the first turn, suggesting that only providing the previous responses is ineffective or hardly interpreted by the agents. Figure 3b illustrates guided-learning mode, where LLMs leverage external feedback to improve performance. When using *Claude-Sonnet-4* to provide concrete suggestions (e.g., code edits), the performance of the second turn (red) exceeds first turn (blue), and both outperform those in the self-evolving, indicating that appropriate guidance helps agents to refine the response. Due to the computational constraints, most models used in our setup have around 3B parameters and struggle to interpret vague feedback. We hypothesize that larger models with better reasoning abilities could benefit from more implicit guidance.

6 Conclusion

In this paper, we model LLM collaboration as a cooperative MARL problem and formalize it as a Dec-POMDP. We introduce the MAGRPO algorithm to optimize their cooperation with aligned rewards. Our experiments on coding collaboration show that MAGRPO enables agents to generate higher-quality solutions more efficiently through effective coordination. This work highlights the potential of MARL methods for scalable and robust LLM collaboration and encourages future exploration of more cooperation schemes in large-scale software systems.

7 Acknowledgments

This work was partially funded by U.S. National Science Foundation (NSF) grants #2044993 and #2409351. This work used Delta and DeltaAI advanced computing and data resources at the National Center for Supercomputing Applications through allocation CIS250443 and CIS250554 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support program, which is supported by NSF grants #2138259, #2138286, #2138307, #2137603, and #2138296.

We thank Gregory Bauer and Brett Bode for their help with resolving job failure issues on Delta GPUs, and members of the Lab for Learning and Planning in Robotics (LLRP) for valuable discussion.

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.

Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. URL `https://www.marl-book.com`.

Christopher Amato. An initial introduction to cooperative multi-agent reinforcement learning, 2025. URL `https://arxiv.org/abs/2405.06161`.

Christopher Amato, George Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P How, and Leslie P Kaelbling. Policy search for multi-robot coordination under uncertainty. *The International Journal of Robotics Research*, 35(14):1760–1778, 2016. doi: 10.1177/0278364916679611.

Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, et al. Gemini: A family of highly capable multimodal models, 2025. URL `https://arxiv.org/abs/2312.11805`.

Anthropic. How we built a multi-agent research system, 2024. URL `https://www.anthropic.com/engineering/built-multi-agent-research-system`.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning, 2019. URL `https://arxiv.org/abs/1912.06680`.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025. URL `https://arxiv.org/abs/2503.13657`.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate, 2023. URL `https://arxiv.org/abs/2308.07201`.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL `https://arxiv.org/abs/2305.14325`.

Andrew Estornell and Yang Liu. Multi-llm debate: Framework, principals, and interventions. In *Neural Information Processing Systems (NeurIPS)*, 2024. URL https://openreview.net/forum?id=sy7eSEXdPC.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients, 2024. URL https://arxiv.org/abs/1705.08926.

Jakob N. Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness, 2018. URL https://arxiv.org/abs/1709.04326.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024. URL https://arxiv.org/abs/2308.00352.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate, 2024. URL https://arxiv.org/abs/2305.19118.

Shalev Lifshitz, Sheila A. McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers, 2025. URL https://arxiv.org/abs/2502.20379.

Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning Proceedings 1994*, pages 157–163. Morgan Kaufmann, San Francisco (CA), 1994. ISBN 978-1-55860-335-6. doi: https://doi.org/10.1016/B978-1-55860-335-6.50027-1. URL https://www.sciencedirect.com/science/article/pii/B9781558603356500271.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, Wee Sun Lee, and Natasha Jaques. Spiral: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning, 2025a. URL https://arxiv.org/abs/2506.24119.

Shuo Liu, Yunhao Wang, Xu Chen, Yongjie Fu, and Xuan Di. Smart-eflo: An integrated sumo-gym framework for multi-agent reinforcement learning in electric fleet management problem. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 3026–3031, 2022. doi: 10.1109/ITSC55140.2022.9922047.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025b. URL https://arxiv.org/abs/2503.20783.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2020. URL https://arxiv.org/abs/1706.02275.

Hao Ma, Tianyi Hu, Zhiqiang Pu, Boyin Liu, Xiaolin Ai, Yanyan Liang, and Min Chen. Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2410.06101.

NVIDIA. Introduction to llm agents, 2024. URL https://developer.nvidia.com/blog/introduction-to-llm-agents/.

F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, May 2008. ISSN 1076-9757. doi: 10.1613/jair.2447. URL http://dx.doi.org/10.1613/jair.2447.

Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer, 2016. doi: 10.1007/978-3-319-28929-8.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning, 2025. URL https://arxiv.org/abs/2502.18439.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Chatdev: Communicative agents for software development, 2024. URL https://arxiv.org/abs/2307.07924.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024. URL https://arxiv.org/abs/2305.18290.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning, 2018. URL https://arxiv.org/abs/1803.11485.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge, 2019. URL https://arxiv.org/abs/1902.04043.

Bidipta Sarkar, Warren Xia, C. Karen Liu, and Dorsa Sadigh. Training language models for social deduction with multi-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2502.06060.

Amrith Setlur, Nived Rajaraman, Sergey Levine, and Aviral Kumar. Scaling test-time compute without verification or rl is suboptimal, 2025. URL https://arxiv.org/abs/2502.12118.

Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, Cambridge, UK, 2009. ISBN 9780521899437.

Marta Skreta, Naruki Yoshikawa, Sebastian Arellano-Rubach, Zhi Ji, Lasse Bjørn Kristensen, Kourosh Darvish, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. Errors are useful prompts: Instruction guided task programming with verifier-assisted iterative prompting, 2023. URL https://arxiv.org/abs/2303.14100.

Oliver Slumbers, David Henry Mguni, Kun Shao, and Jun Wang. Leveraging large language models for optimised coordination in textual multi-agent reinforcement learning, 2024. URL https://openreview.net/forum?id=1PPjf4wife.

Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000. URL http://www.cs.utexas.edu/users/ai-lab?MASsurvey.

Vighnesh Subramaniam, Yilun Du, Joshua B. Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. Multiagent finetuning: Self improvement with diverse reasoning chains, 2025. URL https://arxiv.org/abs/2501.05707.

Ming Tan. Multi-agent reinforcement learning: Independent versus cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, San Francisco, CA, USA, 1993. Morgan Kaufmann. ISBN 1-55860-307-7.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning, 2017. URL https://arxiv.org/abs/1708.04782.

Bichen Wang, Yuzhe Zi, Yixin Sun, Yanyan Zhao, and Bing Qin. Rkld: Reverse kl-divergence-based knowledge distillation for unlearning personal information in large language models, 2024. URL https://arxiv.org/abs/2406.01983.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning, 2021. URL https://arxiv.org/abs/2008.01062.

Marco A Wiering. Multi-agent reinforcement learning for traffic light control. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'2000)*, pages 1151–1158, Stanford, CA, USA, 2000. Morgan Kaufmann.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023a. URL https://arxiv.org/abs/2308.08155.

Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A. Smith, Mari Ostendorf, and Hannaneh Hajishirzi. Fine-grained human feedback gives better rewards for language model training, 2023b. URL https://arxiv.org/abs/2306.01693.

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.

Pu Zhao, Fei Sun, Xuan Shen, Pinrui Yu, Zhenglun Kong, Yanzhi Wang, and Xue Lin. Pruning foundation models for high accuracy without retraining, 2024. URL https://arxiv.org/abs/2410.15567.

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 Formalization of Multi-Agent Interaction

Many studies adopt Partially Observable Stochastic Games (POSG) to model the LLM interaction in MAS Slumbers et al. [2024], Park et al. [2025], Liu et al. [2025a], Sarkar et al. [2025]. In this section, we show that Dec-POMDP offers special merits compared to POSG in the solution concept in the cooperative settings, thus more suited to model LLM collaboration.

A.1 Dec-POMDP

A Dec-POMDP is defined by $\langle \mathcal{I}, \mathcal{S}, \{\mathcal{O}_i\}, \{\mathcal{A}_i\}, R, T, H \rangle$. At each step t, since an agent cannot directly observe the state s_t, it usually maintains local observation-action history $h_{i,t} = (o_{i,0}, a_{i,0}, \ldots, o_{i,t})$ to infer a belief over the underlying state. Decisions are made according to a local policy $\pi_i : \mathcal{H}_{i,t} \to \Delta(\mathcal{A}_i)$, which maps histories to probability distributions over actions. The set of all local policies forms the joint policy $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_n\}$. In cooperative settings, the objective is to maximize shared cumulative rewards. As proved in Oliehoek et al. [2008], there is always an optimal joint policy in a Dec-POMDP,

$$\boldsymbol{\pi}^* = \underset{\boldsymbol{\pi} \in \boldsymbol{\Pi}}{\operatorname{argmax}} \, \mathbb{E}_{\boldsymbol{\pi}} \left[\sum_{t=0}^{H-1} R(s_t, a_t) \right]. \tag{3}$$

A.2 POSG

A Partially Observable Stochastic Game (POSG), so-called Partially Observable Markov Game (POMG), does not assume cooperative behavior among agents. It can be either a cooperative, competitive, or mixed game. A POSG is defined as $\langle \mathcal{I}, \mathcal{S}, \{\mathcal{A}_i\}, T, \{\mathcal{O}_i\}, O, \{R_i\}, H \rangle$, where each agent has its own reward function $R_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. In POSG, each agent seeks to maximize its individual return under the fixed policies of all others $\boldsymbol{\pi}_{-i}$. The optimal policy $\pi_i^{\circledast}$ for each agent $i \in \mathcal{I}$ is,

$$\pi_i^{\circledast} = \underset{\pi_i \in \Pi_i}{\operatorname{argmax}} \, \mathbb{E}_{\pi_i, \boldsymbol{\pi}_{-i}} \left[\sum_{t=0}^{H-1} R_i(s_t, a_t) \right], \tag{4}$$

The solutions for POSG are Nash Equilibria (NE), where no agents can unilaterally improve their returns by deviating from their policies. Formally, for all $i \in \mathcal{I}$ and any alternative policy $\pi_i \in \Pi_i$, NE satisfy

$$\mathbb{E} \left[\sum_{t=0}^{H-1} R_i(s_t, a_t) \mid \pi_i^{\circledast}, \boldsymbol{\pi}_{-i}^{\circledast} \right] \geq \mathbb{E} \left[\sum_{t=0}^{H-1} R_i(s_t, a_t) \mid \pi_i, \boldsymbol{\pi}_{-i}^{\circledast} \right]. \tag{5}$$

Like Dec-POMDP, the decision-making in POSG is still concurrent (as stochastic games), where all agents act synchronously at each time step. In contrast, turn-based interactions, where agents take turns to act (e.g., chess, Kuhn Poker, tic-tac-toe), are typically modeled as extensive-form games.

A.3 Non-Optimality of POSG Solutions

We illustrate that the solutions of POSG, i.e., NE, may not necessarily lead to joint optimality in cooperative settings.

Consider a one-step matrix game involving 2 agents, where each agent selects an action from the action space $\mathcal{A} = \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}\}$. The joint action profile determines the utility as presented in Table 2.

$a_1 \backslash a_2$	$\mathcal{A}^{(1)}$	$\mathcal{A}^{(2)}$
$\mathcal{A}^{(1)}$	10	7
$\mathcal{A}^{(2)}$	7	0

Table 2: Joint utility matrix of 2 agents.

This matrix game can be potentially decomposed into 2 POSG in Table 3 through reward shaping.

$a_1 \backslash a_2$	$\mathcal{A}^{(1)}$	$\mathcal{A}^{(2)}$
$\mathcal{A}^{(1)}$	(5, 5)	(3, 4)
$\mathcal{A}^{(2)}$	(4, 3)	(0, 0)

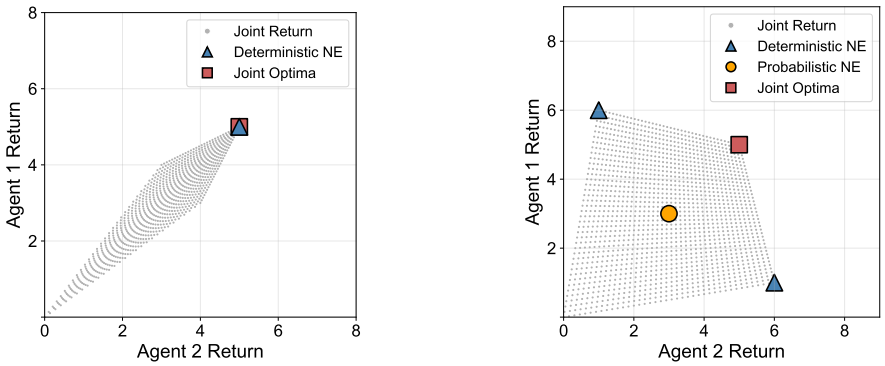
(a) POSG 1

$a_1 \backslash a_2$	$\mathcal{A}^{(1)}$	$\mathcal{A}^{(2)}$
$\mathcal{A}^{(1)}$	(5, 5)	(1, 6)
$\mathcal{A}^{(2)}$	(6, 1)	(0, 0)

(b) POSG 2

Table 3: Return tables of 2 POSG.

In the POSG presented in Table 3a, $(\mathcal{A}^{(1)}, \mathcal{A}^{(1)})$ is a Nash equilibrium (blue triangle in Figure 4a). When $a_1 = \mathcal{A}^{(1)}$, $U_2(\mathcal{A}^{(1)}, \mathcal{A}^{(1)}) > U_2(\mathcal{A}^{(1)}, \mathcal{A}^{(2)})$; when $a_1 = \mathcal{A}^{(2)}$, $U_2(\mathcal{A}^{(2)}, \mathcal{A}^{(1)}) > U_2(\mathcal{A}^{(2)}, \mathcal{A}^{(2)})$. Therefore, the best response for agent 2 is $a_2^{\circledast} = \mathcal{A}^{(1)}$. Similarly, since $U_1(\mathcal{A}^{(1)}, \mathcal{A}^{(1)}) > U_1(\mathcal{A}^{(2)}, \mathcal{A}^{(1)})$, we obtain $a_1^{\circledast} = \mathcal{A}^{(1)}$. This NE also achieves joint optimality with the maximum utility $5 + 5 = 10$ (red square in Figure 4a).



(a) POSG 1 (b) POSG 2

Figure 4: Utility spaces of 2 POSG.

However, certain reward decompositions may yield non-optimal solutions for cooperative games in Table 3, even when POSG solutions reach NE. For the POSG shown in Table 3b, the deterministic NE are $(\mathcal{A}^{(1)}, \mathcal{A}^{(2)}), (\mathcal{A}^{(2)}, \mathcal{A}^{(1)})$ (blue triangles in Figure 4b). When $a_1 = \mathcal{A}^{(1)}$, agent 2 prefers $\mathcal{A}^{(2)}$ as $U_2(\mathcal{A}^{(1)}, \mathcal{A}^{(2)}) > U_2(\mathcal{A}^{(1)}, \mathcal{A}^{(1)})$; when $a_1 = \mathcal{A}^{(2)}$, agent 2 prefers $\mathcal{A}^{(1)}$ since $U_2(\mathcal{A}^{(2)}, \mathcal{A}^{(1)}) > U_2(\mathcal{A}^{(2)}, \mathcal{A}^{(2)})$. Agent 1 faces the same issue. Thus, neither agent can unilaterally improve their utilities by deviating. However, the collective utilities obtained from both policies yield $6 + 1 = 7 < 10$, which are suboptimal compared to the joint optimum (red square in Figure 4b).

In Table 3b, even the probabilistic NE under stochastic policies is still non-optimal. Suppose agent 1 selects $\mathcal{A}^{(1)}$ with probability p, and agent 2 selects $\mathcal{A}^{(1)}$ with probability q, $R_1(\mathcal{A}^{(1)}, \cdot) = 5q + (1 - q) = 4q + 1$, $R_1(\mathcal{A}^{(2)}, \cdot) = 6q$, $R_1(\mathcal{A}^{(1)}, \cdot) = R_1\mathcal{A}^{(2)}, \cdot)$ yields $q = 0.5$; similarly, $R_2(\cdot, \mathcal{A}^{(1)}) = 5p + (1 - p) = 4p + 1$, $R_2(\cdot, \mathcal{A}^{(2)}) = 6p$, $R_2(\mathcal{A}^{(1)}, \cdot) = R_2\mathcal{A}^{(2)}, \cdot)$ yields $p = 0.5$. This probabilistic NE, $\pi_1^{\circledast}(\mathcal{A}^{(1)}) = \pi_1^{\circledast}(\mathcal{A}^{(2)})$, $\pi_2^{\circledast}(\mathcal{A}^{(1)}) = \pi_2^{\circledast}(\mathcal{A}^{(2)})$ leads to overall utilities $3 + 3 = 6 < 10$ (orange circle in Figure 4b).

Although appropriate reward shaping techniques can transform a cooperative game into a POSG like Table 3a to make the NE also jointly optimal, this becomes more challenging when more agents are involved and episodes become longer. We employ Dec-POMDP to avoid the intricate reward engineering and seek the joint optimality.

B Cooperation Schemes

By training the auxiliary and main coders to cooperate under minimal constraints (with only the problem description and their respective roles provided), diverse cooperation schemes naturally emerge. We present 4 representative schemes observed in our models.

B.1 Fallback

The most commonly observed cooperation scheme is the main agent providing a fallback for the auxiliary function. Although prompted to use the auxiliary output, the main agent may still doubt its completeness and reimplement the solution as a backup in case of potential errors.

Listing 1: `compare_one(a,b)`

```
def aux(x, y):
    # Convert x to a float
    if isinstance(x, str):
        x = float(x.replace(",", "."))

    # Vulnerable if y in different types
    if x < y:
        return y
    else:
        return x

def compare_one(a, b):
    try:
      return aux(a, b)
  except Exception:
      # Robust implementation
      if type(a) == type(b):
          if a > b:
                return a
          elif a < b:
                return b
          else:
                return None
      elif type(a) == float and type(b) == str:
          b_float = float(b.replace(",", "."))
          if a > b_float:
                return a
          elif a < b_float:
                return b_float
          else:
                return None
      elif type(a) == str and type(b) == float:
            a_str = str(a.replace(",", "."))
            a_float = float(a_str)
          if a_float > b:
                return a_str
          elif a_float < b:
                return b
          else:
                return None
      else:
          return None
```

For example, the problem `compare_one(a,b)` asks to return the larger of 2 values in its original type (string, float, or int), or `None` if equal. Listing 1 demonstrates how the fallback mechanism provides an alternative in case of runtime errors in the auxiliary implementation. The `aux(x,y)` function converts variable `x` to a float, resulting in an operand type mismatch if `y` is a string. The main agent handles the type error by falling back to its secondary implementation. While the main implementation may still be vulnerable, this mechanism benefits small models that commonly encounter syntax and runtime errors. Despite some redundancy, a proper fallback can enhance code robustness.

B.2 Decorator

When the main agent trusts the auxiliary to accurately handle the core logic, it typically acts as a decorator, adding complementary features to enhance the output. The main agent could handle edge cases for improved robustness or reformat the outputs (e.g., sorting, rounding, and adjusting units). The necessity of such decorating logic depends on the auxiliary agent's functionality and completeness.

Listing 2: `is_palindrome(text)`

```
def aux1(text):
    if len(text) == 1:
        return True
    else:
        return aux1(text[1:-1])

def aux2(text):
    if len(text) <= 1:
        return True
    else:
        return aux2(text[1:-1])

def is_palindrome(text):
    if not text:
        return True

    # Edge case handler
    return text[0] == text[-1] and aux1(text)

    # Redundant decorator
    return text[0] == text[-1] and aux2(text)
```

Listing 2 illustrates a decorator main function and how its utility varies depending on the completeness of the auxiliary implementation. In this problem, 2 agents generate codes together to check if a given string is a palindrome. When paired with `aux1` that only handles the recursion boundary condition of single-character strings, the empty string check of the main serves as a necessary edge case handler. However, when working with `aux2`, which already has a more comprehensive edge case consideration, this handle becomes redundant.

B.3 Coordinator

In large-scale software systems, it would be beneficial to have pipelines for repeated or data-parallel operations (e.g., batch processing, stream transformations). This corresponds to the coordinator cooperation scheme in our models, where the main agent divides the tasks into parts and assigns them to the auxiliary agent.

A simple example involves the main agent acting as an iterator, using a loop (e.g., a `for` loop) to structure the problem. The auxiliary function generates partial solutions within each iteration. These partial results are then aggregated to form the final output. However, this cooperation scheme is unstable, as it depends heavily on the correct functionality of the auxiliary agent. When the auxiliary agent fails to complete its subtask, the entire solution breaks down.

Listing 3: `flip_case(string)`

```
def aux(string: str) -> str:
    result = ""
    for char in string:
        if char.islower():
            result += char.upper()
        elif char.isupper():
            result += char.lower()
        else:
            result += char
    return result

def flip_case(string: str):
    flipped = ""
    for char in string:
        flipped += aux(char)
    return flipped
```

Listing 3 demonstrates a solution to flip the case of characters in a string. The auxiliary function flips the case of each character, while the main function calls this auxiliary function for each character and appends it to the result. This scheme can be extended to more complex scenarios, where subtasks are assigned in a hierarchical structure.

B.4 Strategy Filter

When handling complex problems, the main agent may need to implement logic based on multiple conditions. In such cases, the auxiliary agent can act as a filter for specific branches of logic, often appearing within conditional blocks (e.g., following an `if` statement). This scheme resembles the adaptive control flow in practice. In rule-based pipelines, an auxiliary agent evaluates preconditions (e.g., task types, system status, configurations) and directs workers to execute appropriate subroutines, thereby enhancing project modularity.

Listing 4: `x_or_y(n,x,y)`

```
def aux(n):
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False
    return True

def x_or_y(n, x, y):
    # Check if n is prime
    if aux(n):
        return x
    else:
        return y
```

Listing 4 presents a solution for `x_or_y(n,x,y)` problem, which returns `x` if `n` is prime and `y` otherwise. The auxiliary function handles the primality checking, while the main function is responsible for returning results. The same pattern can also be found in the solutions of `prime_fib(n)`, `factorize(n)`, and `largest_prime_factor(n)`.

C Broader Impacts

Prompt-based coordination is often brittle Estornell and Liu [2024], as agents may fail to follow instructions they were not explicitly trained to interpret. Our method builds on a solid theoretical foundation in cooperative MARL, explicitly optimizing agents for joint optimality. Our work also opens opportunities to enhance existing test-time multi-agent interaction methods by integrating MARL techniques Du et al. [2023], Lifshitz et al. [2025], Wu et al. [2023a], particularly in settings that involve task decomposition and iterative feedback integration.

This work also explores a new perspective on accelerating LLM inference through cooperative MARL. While mainstream acceleration techniques (e.g., knowledge distillation, pruning, and quantization) improve efficiency at the cost of information loss Wang et al. [2024], Zhao et al. [2024], our approach suggests decentralized coordination among specialized agents, thereby alleviating the burden of long-context memory and joint decision-making on a single model. Each agent can focus on a specific subtask, enabling more modular and robust reasoning.

D Limitations and Future Works

Nevertheless, this study is subject to several limitations. First, we focus on homogeneous agents for simplicity, assuming they perform similar tasks despite being assigned different roles, e.g., both the auxiliary agent and main agent are generating Python functions. Future research could explore LLM collaboration among heterogeneous agents with diverse capabilities and functionalities.

Due to computational constraints, we train LLMs with MAGRPO on limited datasets using relatively small-scale language models. When LLM-based coding agents are deployed in larger-scale projects involving multiple files and modules, more diverse and complex cooperation schemes are likely to emerge, which would further demonstrate the potential of decentralized coordination in MAS.

The simplicity of our reward model inevitably leads to narrow reward signals and potential reward hacking. As suggested by many research studies and industrial practice Uesato et al. [2022], Wu et al. [2023b], designing more expressive and fine-grained reward models (e.g., multi-aspect rewards, process-supervised rewards) is essential for better aligning agent cooperation with human preferences.