LEARNING DECENTRALIZED LLM COLLABORATION WITH MULTI-AGENT ACTOR CRITIC

Shuo Liu, Tianle Chen, Ryan Amiri, Christopher Amato
Khoury College of Computer Sciences
Northeastern University
Boston, MA, USA
{liu.shuo2,chen.tianle,amiri.ry,c.amato}@northeastern.edu

ABSTRACT

Recent work has explored optimizing LLM collaboration through Multi-Agent Reinforcement Learning (MARL). However, most MARL fine-tuning approaches rely on predefined execution protocols, which often require centralized execution. Decentralized LLM collaboration is more appealing in practice, as agents can run inference in parallel with flexible deployments. Also, current approaches use Monte Carlo methods for fine-tuning, which suffer from high variance and thus require more samples to train effectively. Actor-critic methods are prevalent in MARL for dealing with these issues, so we developed Multi-Agent Actor-Critic (MAAC) methods to optimize decentralized LLM collaboration. In this paper, we analyze when and why these MAAC methods are beneficial. We propose two MAAC approaches, **CoLLM-CC** with a **C**entralized **C**ritic and **CoLLM-DC** with **D**ecentralized **C**ritics. Our experiments across writing, coding, and game-playing domains show that Monte Carlo methods and CoLLM-DC can achieve performance comparable to CoLLM-CC in short-horizon and dense-reward settings. However, they both underperform CoLLM-CC on long-horizon or sparse-reward tasks, where Monte Carlo methods require substantially more samples and CoLLM-DC struggles to converge.

1 INTRODUCTION

Advanced LLMs have demonstrated remarkable capabilities in natural language understanding and generation Achiam et al. (2023); Anil et al. (2023); Bai et al. (2023). This progress has driven growing efforts to transform them into autonomous agents Yao et al. (2022); Yang et al. (2024).

In this context, it is becoming popular to explore coordinating multiple LLMs to improve performance where agents are specified by roles, e.g., generators, planners, or verifiers Wu et al. (2023); Du et al. (2023); Skreta et al. (2023); Qian et al. (2024); Zhang et al. (2025). Building on the studies of Multi-Agent Systems (MAS) Weiss (1999); Stone & Veloso (2000); Van der Hoek & Wooldridge (2008); Shoham & Leyton-Brown (2009), recent methods employ Multi-Agent Reinforcement Learning (MARL) to optimize their interactions Liu et al. (2025a;b); Subramaniam et al. (2025); Wu et al. (2025b). However, most existing approaches remain confined to predefined execution paradigms, which limits their applicability to broader settings. Moreover, these agents often rely on extensive communication to accomplish tasks, necessitating centralized execution, which limits scalability and introduces potential privacy issues in larger-scale MAS.

Decentralized systems have been studied for decades, where each agent is deployed separately and executes independently based on its own observations Waldo et al. (1996); Ghosh (2006); Oliehoek & Amato (2016); Albrecht et al. (2024). Leveraging decentralized LLM agents to complete tasks is beneficial, as it reduces memory and storage pressure on each node and improves the efficiency Huang et al. (2019); Lepikhin et al. (2020); Douillard et al. (2023); Wu et al. (2025a). However, how to effectively optimize these decentralized LLMs to collaborate remains an open question.

Although Monte Carlo methods are widely adopted in RL fine-tuning due to their simplicity and efficiency Li et al. (2023); Ahmadian et al. (2024); Shao et al. (2024); Guo et al. (2025); Hu et al.

(2025); Liu et al. (2026a), extending them to optimize multi-LLM collaboration faces many difficulties Liao et al. (2025); Zhang et al. (2025); Hong et al. (2025); Liu et al. (2026b). Agents need to wait until the end of an episode to receive return signals with high variance. This leads to poor sample efficiency and limits practicality in long-horizon or episodic tasks Sutton & Barto (1998).

In this paper, we develop Multi-Agent Actor-critic (MAAC) methods for optimizing decentralized LLM collaboration. We analyze when and why MAAC methods are beneficial for MARL fine-tuning and introduce 2 approaches, **CoLLM-CC** that employs a centralized critic to estimate joint history values, and **CoLLM-DC** that uses decentralized critics to estimate individual history values. Our evaluation across writing, coding, and game-playing domains shows that Monte Carlo methods and CoLLM-DC achieve performance comparable to CoLLM-CC in the short-horizon dense-reward task; while both underperform CoLLM-CC in long-horizon sparse-reward tasks, where Monte Carlo methods require substantially more samples for training, and CoLLM-DC fails to converge.

Our contributions are summarized as follows: (i) We develop MAAC methods to optimize decentralized LLM collaboration and analyze their advantages; (ii) We propose CoLLM-DC and CoLLM-CC as 2 MAAC approaches; and (iii) Our results on collaborative writing, coding, and game-playing tasks demonstrate that CoLLM-CC consistently outperforms Monte Carlo methods and CoLLM-DC, particularly in long-horizon tasks with sparse rewards.

2 BACKGROUND

2.1 DECENTRALIZED LLM COLLABORATION

In decentralized LLM collaboration, multiple LLM agents cooperate to solve a class of tasks. Each task is expressed with prompts. Agents then produce responses in parallel according to their own policies conditioned on the prompt. The responses of all agents are aggregated to form a solution. All agents share a joint reward based on the aggregated outcome. After each turn, users, external tools, and other models validate the proposed solution and may provide new requirements, suggestions, or constraints for the next iteration. This feedback, together with the dialog history, is maintained and serves both as a decision-making context and as experience for training. The interaction repeats until the task is solved or a predefined turn limit is reached. The goal of agents is to achieve higher-quality solutions with minimal iterations.

Decentralized LLM collaboration offers great advantages by allowing specialized agents to divide and conquer complex problems and operate in parallel to improve efficiency. Rather than relying on a single gigantic LLM in a centralized system, lightweight agents in a decentralized system can focus on subtasks guided by their own prompts, thereby greatly reducing the overhead of maintaining long contexts and extracting relevant semantics. Deploying smaller models locally is also typically easier, safer, and more scalable, making decentralized LLM collaboration well-suited for applications such as long article generation, software engineering, and embodied coordination Gao et al. (2023); Chen et al. (2024); Liu et al. (2026b); Wu et al. (2025a).

2.2 LLM DEC-POMDP

Decentralized LLM collaboration can be seen as a subclass of the Dec-POMDP Oliehoek & Amato (2016), which is denoted as $\langle \mathcal{I}, \mathcal{V}, C, M, \mathcal{S}, \{\mathcal{O}_i\}, \{\mathcal{A}_i\}, R, T, H \rangle$.

In an LLM Dec-POMDP, $\mathcal{I}$ denotes a set of n LLM agents, $\mathcal{V}$ is the token vocabulary in LLM inputs and outputs, C is the input context window size for each agent, and M is the max number of tokens in each LLM outputs. $\mathcal{S} : \mathcal{S}^{\text{sys}} \times \mathcal{S}^{\text{usr}}$ denotes the full global state space. At turn t, $s_t \in \mathcal{S}$ consists of accessible parts $s^{\text{sys}} \in \mathcal{S}^{\text{sys}}$ from the system and inaccessible $s^{\text{usr}} \in \mathcal{S}^{\text{usr}}$ parts from users. In RL fine-tuning via verifiable rewards (RLVR), only the accessible parts are used by the reward model. $\mathcal{O}_i$ is the observation space for agent i with $\mathcal{O} = \times_i \mathcal{O}_i$ the joint observation space, where a local observation $o_{i,t}$ is a natural language prompt providing a partial and noisy view of s_t, and $|\mathcal{O}_i| = |\mathcal{V}|^C$. $\mathcal{A}_i$ is the action space for agent i with $\mathcal{A} = \times_i \mathcal{A}_i$ the joint action space, where an individual action $a_{i,t}$ is its response to the given prompt, and $|\mathcal{A}_i| = |\mathcal{V}|^M$. The joint reward function $R : \mathcal{S}^{\text{sys}} \times \mathcal{A} \to \mathbb{R}$ is implemented via predefined rules or a pretrained reward model. At turn t, the joint rewards $r_t \leftarrow R(s_t^{\text{sys}}, \mathbf{a}_t)$ are determined by the accessible part of current state s_t^{sys} 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

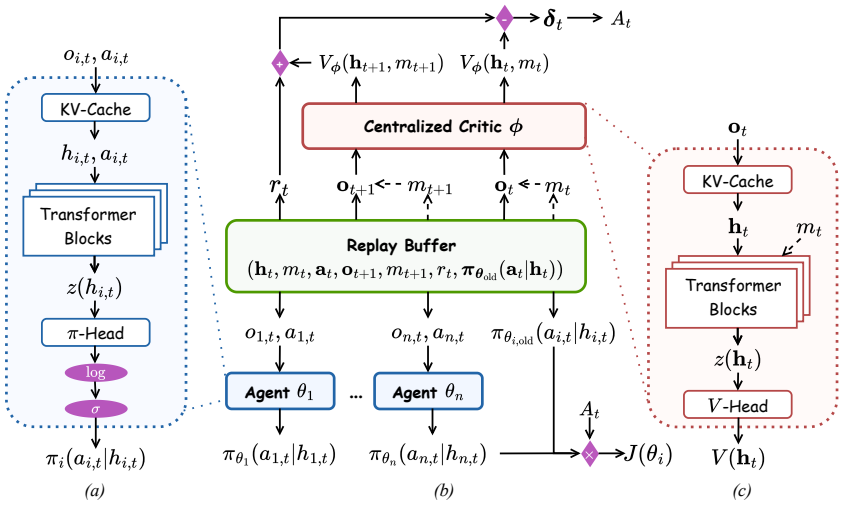


Figure 1: Illustration of CoLLM-CC framework: *(a)* The agent model structure; *(b)* The overall centralized-critic architecture; *(c)* The critic model structure (see CoLLM-DC in Appendix B).

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, the turn limit of a dialog.

Since both the accessible s^{sys} and the inaccessible s^{usr} parts of the states cannot be directly observed by the agents. Each agent maintains its local observation-action history $h_{i,t} = \{o_{i,0}, a_{i,0}, \cdots, o_{i,t}\}$ to infer information about the state $s_t = \{s^{\text{sys}}, s^{\text{usr}}\}$. The history of all agents forms a joint history $\mathbf{h}_t = \{h_{1,t}, \cdots, h_{n,t}\}$, and all agents' policies forms a joint policy $\boldsymbol{\pi} = \{\pi_1, \cdots, \pi_n\}$. A solution to a Dec-POMDP is a joint policy that maximizes the expected cumulative reward over the horizon H, $\boldsymbol{\pi}^* = \{\pi_1^*, \cdots, \pi_n^*\} = \arg\max_{\boldsymbol{\pi}} \mathbb{E}_{\boldsymbol{\pi}} \left[\sum_{t=0}^{H-1} r_t\right]$.

3 Cooperative MARL for LLM Fine-Tuning

In this section, we introduce cooperative MARL methods and provide a theoretical analysis of their advantages and limitations when applied to LLM fine-tuning.

3.1 MA-REINFORCE

Multi-Agent REINFORCE (MA-REINFORCE) is a class of multi-agent policy gradient methods in which each agent i maintains an individual policy π_{θ_i}, and updates it according to Monte-Carlo estimates of joint returns Peshkin et al. (2001). Mathematically,

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{\pi}} \left[\sum_{t=0}^{H-1} \rho_{i,t} \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t} \mid h_{i,t}) \left(G(\mathbf{h}_t) - b(\mathbf{h}_t)\right) \right], \tag{1}$$

where $G(\mathbf{h}_t)$ is the expected return from $\mathbf{h}_t$, $\rho_{i,t} = \frac{\pi_{\theta_i}(a_{i,t}|h_{i,t})}{\pi_{\theta_{i,\text{old}}}(a_{i,t}|h_{i,t})}$ is the importance sampling ratio for correcting the off-policy action sampled, and $b(\mathbf{h}_t)$ denotes an action-independent baseline. In MA-REINFORCE, agents learn from sampled returns without a critic. However, this method can not scale well to long-horizon online learning, because return signals are only available at dialog termination and Monte Carlo estimates suffer from high variance due to accumulated stochasticity.

MA-REINFORCE is initially designed to take one action at each turn during rollout. Recent MARL fine-tuning methods aim to reduce the variance of gradient estimation by generating K i.i.d. joint actions $\{\mathbf{a}_t^1, \cdots, \mathbf{a}_t^K\}$ for each history $\mathbf{h}_t$ following policy $\boldsymbol{\pi}(\cdot|\mathbf{h}_t)$ Liao et al. (2025); Zhang et al. (2025); Hong et al. (2025). In the multi-turn environment, each of these actions yields a successor observation $\{\mathbf{o}_{t+1}^1, \cdots, \mathbf{o}_{t+1}^K\}$, and obtaining low-variance gradients for these successor histories

$\{\mathbf{h}_{t+1}^1, \cdots, \mathbf{h}_{t+1}^K\}$ also requires K samples. This recursively expands into a K-ary rollout tree of depth H Yang et al. (2025); Liu et al. (2026b); Ding & Ye (2025); Ji et al. (2025).

As shown in Theorem 1 in Peshkin et al. (2001), the average gradient estimator by K-sampling MA-REINFORCE is unbiased. Proposition 3.1 measures the variance reduction of K-sampling MA-REINFORCE. Under an independent rollout assumption, its variance scales as $1/K^{H-t}$ in K.

Proposition 3.1. *For each history $\mathbf{h}_t$, suppose agents sample K i.i.d. joint actions $\{\mathbf{a}_t^1, \cdots, \mathbf{a}_t^K\}$ from $\boldsymbol{\pi}(\cdot|\mathbf{h}_t)$. For each $\mathbf{a}_t^k$, an independent K-ary rollout tree is produced to estimate the corresponding expected Monte Carlo return G_t^k. For agent i, define $\bar{g}_{i,t} = \frac{1}{K}\sum_{k=1}^{K} \rho_{i,t}^k \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t}^k \mid h_{i,t})\, G_t^k$ as the averaged gradient estimate over $\{a_{i,t}^1, \cdots, a_{i,t}^K\}$. For any $h_{i,t}$, we assume the gradient estimates for $\{a_{i,t}^1, \cdots, a_{i,t}^K\}$ are independent and episodes are not terminated early. Let σ^2 be the gradient estimate variance when $K = 1$, the variance of $\bar{g}_{i,t}$ satisfies, $\mathrm{Var}_{\boldsymbol{\pi}}(\bar{g}_{i,t} \mid h_{i,t}) = \frac{\sigma^2}{K^{H-t}}$.*

However, this variance reduction comes at the cost of a rapidly increasing LLM inference calls to maintain the rollout tree, as shown in Proposition 3.2.

Proposition 3.2. *Consider an H-horizon episode without early termination $t \in [0, H)$. Suppose MA-REINFORCE expands a full K-ary rollout tree ($K \geq 1$) and, at each history node, draws K i.i.d. joint actions. Each LLM runs one inference at a time to produce a response. Then the total number of inference calls required for this episode is $N_{\mathrm{call}}(n, K, H) = \frac{nK(K^H-1)}{K-1}$.*

3.2 Multi-Agent Actor-Critic

To reduce the variance of gradient estimates and improve sample efficiency, Actor-Critic (AC) methods learn a policy model (actor) π_θ and a value model (critic) V_ϕ (or Q_ψ). In the multi-agent setting, AC methods are common with Decentralized Critics Foerster et al. (2018); De Witt et al. (2020) or with a Centralized Critic (CC) Lowe et al. (2017); Foerster et al. (2018); Yu et al. (2022).

In DC methods, each agent i maintains a local critic $V_{\phi_i}(h_{i,t})$ and updates its policy $\pi_{\theta_i}(\cdot|h_{i,t})$ via

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{H-1} \rho_{i,t}\, \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t} \mid h_{i,t})\; \delta_{i,t}\right], \tag{2}$$

where $\delta_{i,t} = r_t + \gamma V_{\phi_i}(h_{i,t+1}) - V_{\phi_i}(h_{i,t})$, and each V_{ϕ_i} is updated by minimizing its TD loss,

$$\mathcal{L}(\phi_i) = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{H-1} \left(r_t + \gamma V_{\phi_i}(h_{i,t+1}) - V_{\phi_i}(h_{i,t})\right)^2\right]. \tag{3}$$

CC learns a shared value function $V_{\boldsymbol{\phi}}(\mathbf{h}_t)$ that conditions on the joint history $\mathbf{h}_t$ (and even other available information during training) to update each agent's policy $\pi_{\theta_i}(\cdot|h_{i,t})$,

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{H-1} \rho_{i,t}\, \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t} \mid h_{i,t})\; \boldsymbol{\delta}_t\right], \tag{4}$$

where $\boldsymbol{\delta}_t = r_t + \gamma V_{\boldsymbol{\phi}}(\mathbf{h}_{t+1}) - V_{\boldsymbol{\phi}}(\mathbf{h}_t)$, and

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{H-1} \left(r_t + \gamma V_{\boldsymbol{\phi}}(\mathbf{h}_{t+1}) - V_{\boldsymbol{\phi}}(\mathbf{h}_t)\right)^2\right]. \tag{5}$$

Since the critic is just a training construct, in both DC and CC, agents can still execute in a decentralized manner. Assuming convergence of critics, both the gradient estimates of DC in Eq. 2 and CC in Eq. 4 are unbiased Lyu et al. (2021; 2023). Though this assumption can be harder to satisfy for DC methods, because each individual critic V_{ϕ_i} conditions only on its local history $h_{i,t}$, while the policies of the other agents, $\boldsymbol{\pi}_{\boldsymbol{\theta}_{-i}}(\cdot|\mathbf{h}_{-i,t})$, change during training and thus induce non-stationarity.

4 CoLLM-CC

We introduce CoLLM-CC as a representative MAAC approach and discuss key design features, with its DC counterpart, CoLLM-DC, presented in Appendix B.

Algorithm 1: CoLLM-CC

Input: Taskset $\mathcal{D}$, LLM agents $\{\pi_{\theta_i}\}_{i\in\mathcal{I}}$, centralized LLM critic $V_{\boldsymbol{\phi}}$, learning rates α_π, α_V, discount γ, horizon H, replay buffer $\mathcal{B}$, training epochs E
for each episode **do**
  Sample a task from $\mathcal{D}$
  Initialize prompts $o_{i,0}$, and $\mathbf{o}_o \leftarrow \{o_{i,0}\}_{i=1}^n$
  Initialize the dialog history $h_{i,0} \leftarrow \{o_{i,0}\}$, $\mathbf{h}_0 \leftarrow \{h_{i,0}\}_{i=1}^n$, and obtain global information m_0
  Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$
  for turn $t = 0$ to $H-1$ **do**
    Generate response $a_{i,t} \sim \pi_{\theta_i}(\cdot|h_{i,t})$, and calculate its policy $\pi_{\theta_i}(a_{i,t}|h_{i,t})$ with TF, $\forall i \in \mathcal{I}$
    $\mathbf{a}_t \leftarrow \{a_{i,t}\}_{i=1}^n$, $\boldsymbol{\pi}_{\boldsymbol{\theta}} \leftarrow \{\pi_{\theta_i}\}_{i=1}^n$
    Record behavior policy $\boldsymbol{\pi}_{\boldsymbol{\theta}_{\text{old}}} \leftarrow \boldsymbol{\pi}_{\boldsymbol{\theta}}$
    Obtain joint reward r_t, next turn prompts $o_{i,t+1}, \forall i \in \mathcal{I}$, and global information m_{t+1}
    Update $h_{i,t+1} \leftarrow \{h_{i,t}, a_{i,t}, o_{i,t+1}\}, \forall i \in \mathcal{I}$
    $\mathbf{o}_{t+1} \leftarrow \{o_{i,t+1}\}_{i=1}^n$, $\mathbf{h}_{t+1} \leftarrow \{h_{i,t+1}\}_{i=1}^n$
    Store $(\mathbf{h}_t, m_t, \mathbf{a}_t, r_t, \mathbf{o}_{t+1}, m_{t+1}, \boldsymbol{\pi}_{\boldsymbol{\theta}_{\text{old}}})$ into $\mathcal{B}$
  end for
  for training epoch $e = 1, \cdots, E$ **do**
    Sample a minibatch β from $\mathcal{B}$
    for each $\left(\mathbf{h}_t^b, m_t^b, \mathbf{a}_t^b, r_t^b, \mathbf{o}_{t+1}^b, m_{t+1}^b, \boldsymbol{\pi}_{\boldsymbol{\theta}_{\text{old}}}^b\right) \in \beta$ **do**
      Calculate $\mathcal{L}^b(\boldsymbol{\phi}, m^b)$ (Eq. 5)
      Calculate $\pi_{\theta_i}(a_{i,t}^b|h_{i,t}^b)$ with TF, $\forall i \in \mathcal{I}$
      Calculate $\nabla_{\theta_i} J^b(\theta_i)$ (Eq. 4), $\forall i \in \mathcal{I}$
    end for
    Update critic $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha_V \frac{1}{|\beta|} \sum_b \nabla_{\boldsymbol{\phi}} \mathcal{L}^b(\boldsymbol{\phi}, m^b)$
    Update actor $\theta_i \leftarrow \theta_i + \alpha_\pi \frac{1}{|\beta|} \sum_b \nabla_{\theta_i} J^b(\theta_i)$
  end for
end for
Output: $\{\pi_{\theta_i}\}_{i\in\mathcal{I}}$

4.1 History Representation

In cooperative MARL, each agent i typically uses a recurrent neural network to encode its history $h_{i,t}$ by taking $o_{i,t}$ as input, and its hidden state $z_{i,t}$ serving as a compact but lossy representation. However, this paradigm cannot effectively scale to LLMs that take long prompt sequences as input, and language representations are often high-dimensional. Transformers can capture long-range dependencies via attention mechanisms, making them well-suited for modeling dialog histories.

In CoLLM-CC, the dialog history is maintained in a key-value (KV) cache. At each turn, we concatenate the KV pairs from previous turns with the new prompt from the environment (line 5 of Alg. 1), and then maintain only the most recent C pairs in the cache. The agents' KV-caches are independent and private to maintain decentralized inference. In addition to $\mathbf{h}_t$, the global information m_t can also be incorporated into value estimation to facilitate CC learning a richer or more accurate semantic representation during training. Such global information includes model specifications, task completion progress, or external models or tools existing in the system, which can either be appended as prompts or concatenated with $z(\mathbf{h}_t)$ in a numerical representation.

4.2 Sequences as Actions

Since language is highly structured, rewards in language tasks are naturally defined at the level of complete responses. Considering each token an action leads to sparse, uninformative credit assignment, and sequences can be used as macro-actions in RL fine-tuning. However, the response-level action space is combinatorially large $|\mathcal{V}|^M$, making the probability of an entire sequence non-trivial to obtain. CoLLM-CC employs Teacher-Forced (TF) forward passes to obtain the proba-

bility of a response based on the current policy. At dialog turn t, for LLM agent i, the probability of an response $a_{i,t}$ given $h_{i,t}$ under policy π_{θ_i} factorizes autoregressively as $\pi_{\theta_i}(a_{i,t}|h_{i,t}) = \prod_{\mu=1}^{M} \pi_{\theta_i}(a_{i,t_\mu}|h_{i,t}, a_{i,t_{<\mu}})$, where a_{i,t_μ} denotes the μ^{th} token in $a_{i,t}$, and $a_{i,t_{<\mu}}$ is the token prefix up to μ. TF computes the conditional distributions for all target tokens in parallel under a causal mask, and thus the log-probability of the entire sequence can be efficiently obtained in one pass by summing the log-probabilities of each token. TF passes are applied twice in Alg. 1, when agents generate responses to roll out (line 8), and when they update their policies (line 20).

4.3 ALGORITHM

Fig. 1b and Alg. 1 show the training procedure of CoLLM-CC. We first sample a task from $\mathcal{D}$, and initialize prompts for each agent $\mathbf{o}_0 \leftarrow \{o_{1,0}, \cdots, o_{n,0}\}$, with global information m_0 obtained from the environment. At each turn $t \in [0, H)$, each agent generates a response $a_{i,t} \sim \pi_{\theta_i}(\cdot|h_{i,t})$. A TF pass is applied right after for each agent to compute the probability $\pi_{\theta_i}(a_{i,t}|h_{i,t})$ of $a_{i,t}$ under $\pi_{\theta_i}(\cdot|h_{i,t})$. All agents' responses form a solution to the task $\mathbf{a}_t \leftarrow \{a_{1,t}, \ldots, a_{n,t}\}$, and the joint policy can be constructed by $\boldsymbol{\pi_\theta} \leftarrow \{\pi_{\theta_i}, \cdots, \pi_{\theta_i}\}$. This joint policy is recorded as a behavior policy $\boldsymbol{\pi}_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t|\mathbf{h}_t) \leftarrow \boldsymbol{\pi_\theta}(\mathbf{a}_t|\mathbf{h}_t)$. Executing $\mathbf{a}_t$ yields a joint reward r_t from the reward model and triggers the evolution of the environment. Users provide new prompts $o_{i,t+1}$, and global information is updated as m_{t+1}. Each agent updates its history as $h_{i,t+1} \leftarrow \{h_{i,t}, a_{i,t}, o_{i,t+1}\}$. The prompts, global information, responses, reward, new prompts, new global information, and the behavior policy, $(\mathbf{h}_t, m_t, \mathbf{a}_t, r_t, \mathbf{o}_{t+1}, \boldsymbol{\pi}_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t|\mathbf{h}_t))$, are stored into the replay buffer $\mathcal{B}$ for subsequent training.

At each training epoch, a minibatch β of transitions is drawn from $\mathcal{B}$. For each sample indexed by b, $(\mathbf{h}_t^b, m_t^b, \mathbf{a}_t^b, r_t^b, \mathbf{o}_{t+1}^b, m_{t+1}^b, \boldsymbol{\pi}_{\boldsymbol{\theta}_{\text{old}}}^b(\mathbf{a}_t^b|\mathbf{h}_t^b)) \in \beta$, the TD loss $\mathcal{L}^b(\boldsymbol{\phi}, m_t^b)$ is calculated according to Equation 5 by the structure shown in Fig 1c. The probability $\pi_{\theta_i}(a_{i,t}^b|h_{i,t}^b)$ of each agent under its current policy is calculated by a TF pass, which is used to update policy gradient $\nabla_{\theta_i} J^b(\theta_i)$ according to Eq. 4 in Fig 1a. The gradients of all samples are averaged to update the centralized critic $\boldsymbol{\phi}$ and all actors θ_i, $\forall i \in \mathcal{I}$, respectively.

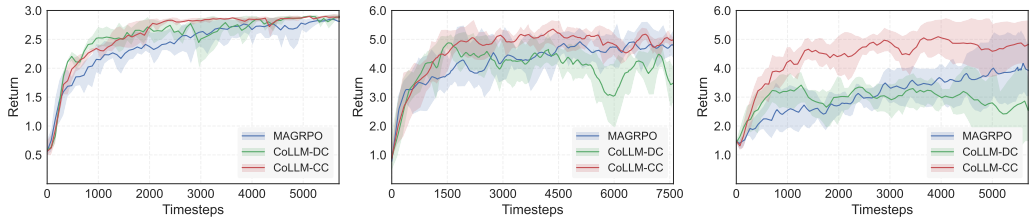
5 EXPERIMENTS

We evaluate MAAC on 3 domains: document processing, programming, and language-based games. Additional results, the prompt and reward design, and anonymized code is provided in Appendices.

5.1 SETUP

Processing long documents is time-consuming, whereas parallel execution among decentralized agents can substantially improve efficiency. We frame the common collaborative writing into a *TLDR* summarization task. 2 *Qwen3-1.7B* agents summarize Reddit posts from the `prompt` field in *TLDR*. One acts as a high-level summarizer producing a concise paragraph, and one serves as a detailed summarizer providing more comprehensive information. The writing quality is evaluated using a weighted sum of 3 metrics with task-specific hyperparameters. The structure assesses the length ratio of generated responses, encouraging agents to contribute fairly to maximize parallel execution. Style consistency is quantified by a Jaccard similarity score to ensure a uniform tone throughout the article. Logical coherence is measured by a logarithmic function of the frequency of transition words, encouraging natural, smooth transitions between paragraphs.

We use a code-generation task to demonstrate collaborative coding. To test the collaboration between heterogeneous agents, an auxiliary *Qwen2.5-Coder-3B* assists a primary *Qwen3-4B-Instruct* generator in solving basic programming tasks. The auxiliary agent intends to implement lightweight utilities that support the primary agent to produce the core logic. As many tasks in *HumanEval* and *MBPP* are atomic and cannot meaningfully benefit from cooperation (e.g., `add(x,y)`), optimizing collaborative behaviors on such instances introduces substantial noise. Therefore, we construct *CoopHumanEval* dataset (*CoopHE*), which focuses on problems that naturally admit cooperative decomposition. In *CoopHE*, the auxiliary function is named `aux`, and the main function signature is provided in the `prompt` field of the problem description. Agents interact in a multi-turn environment, $H = 2$. Agents' outputs are concatenated and they receive feedback from static analysis (abstract syntax trees) or dynamic execution (sandbox tests), then proceed to the next turn. The episode



(a) Article Summarization | *TLDR* (b) Code Generation | *CoopHE* (c) Minecraft | *HouseBuild*

Figure 2: Evaluation results of MAGRPO, CoLLM-DC, and CoLLM-CC across article writing, code generation, and game-playing tasks over 5 runs. The y-axis shows expected return, with limits (min/max) indicating the return scale for each task, and 95% bootstrapped confidence intervals.

terminates if the program passes all tests and the main effectively utilizes the auxiliary function. We use the average pass rate as the evaluation metric. Pass@k results are shown in Appendix D.

We consider a *HouseBuild* task from *Minecraft* to evaluate LLM collaboration. A *Qwen2.5-3B-Instruct* and a *Qwen3-4B-Instruct* agents collaboratively construct a house that conforms to architectural specifications while defending against attacks from hostile mobs (spiders). Since adversaries can actively interfere with agents, effective coordination requires allocating effort between construction and threat mitigation. *HouseBuild* unfolds in 4 turns, $H = 4$. A simulated player gives instructions for incomplete building after each turn. We use players' average health points (HP) as an indicator of whether attacks are successfully repelled, and Intersection over Union (IoU) as an evaluation metric for assessing compliance with the given specifications.

5.2 Baselines

We compare CoLLM-DC and CoLLM-CC with 3 categories of baselines: single-model methods, multi-agent test-time interaction, and other MARL methods.

Single-model baselines contain the results from a larger base model or its fine-tuned variants. We select comparable-size models as the sum of MAS: *Qwen3-4B-Instruct* for writing tasks, and *Qwen2.5-7B Coder* and *Instruct* models for code generation and game-playing, respectively. The reward model is same as that used in MARL methods, except for cooperation metrics. All single-model baselines are single-turn, as the feedback differs significantly from multi-agent methods.

We compare our method against 3 prompt-based multi-agent baselines with the same agents. In the parallel baseline, agents act independently without communication. The sequential pipeline baseline represents a one-way communication setting where agents act in turn and can later observe the inputs and outputs of earlier agents. The discussion baseline frames the multi-agent debate Du et al. (2023), where agents engage in two-way, asynchronous communication. Each agent can access the others' outputs in the next turn and generate its response accordingly. The number of message-passing rounds is the same among multi-agent methods, and we keep the prompt adjustment minimal.

We compare CoLLM-DC and CoLLM-CC with a representative MA-REINFORCE approach (Section 3.1), Multi-Agent Group Relative Policy Optimization (MAGRPO) Liu et al. (2026b). In MAGRPO, the group relative baseline is the average return for K i.i.d. responses $\{\mathbf{a}^1, \cdots, \mathbf{a}^K\}$ on each $\mathbf{h}_t$. We set $K = 4$ for short-horizon writing and coding tasks, and $K = 2$ for Minecraft tasks due to the rapid increase in the number of required samples (Proposition 3.2). The same learning rate is applied to update each sample across all 3 methods to ensure a fair comparison of sample efficiency. We use the turn index and task completion progress as m_t in both CC and DC for consistency, though such m_t is often unavailable to DC in standard MARL.

5.3 Evaluation Results

The performance comparison between MARL and other baselines is presented in Table 1, where the evaluations of MAGRPO (blue), CoLLM-DC (green), and CoLLM-CC (red) are presented in Fig. 2.

In writing tasks, single-model methods achieve strong performance but have low inference speed and high cost due to their larger model size. Since the raw models are not specifically optimized

Method	TLDR			CoopHE			HouseBuild			
	Time	Cost	Score	Time	Cost	Pass	Time	Cost	HP	IoU
Raw Model	5.0	465	30.3	2.5	90	56.3	22.6	1016	99.6	43.2
GRPO	4.1	387	91.7	2.5	**88**	61.8	22.0	890	**100.0**	54.6
AC	4.0	374	94.7	2.5	91	62.5	22.2	904	**100.0**	**55.9**
Parallel	2.3	244	22.7	**2.3**	138	50.0	19.2	502	21.8	46.1
Pipeline	4.3	238	21.7	2.6	177	62.5	20.2	488	30.6	41.3
Discussion	4.6	234	22.3	2.9	191	25.0	21.0	510	27.6	38.1
MAGRPO	**1.8**	**178**	93.5	**2.3**	132	74.3	19.2	446	80.2	50.9
CoLLM-DC	1.9	194	**95.4**	2.5	161	59.1	19.4	470	43.8	46.8
CoLLM-CC	**1.8**	181	95.2	2.6	166	**75.2**	**19.0**	**442**	86.4	52.7

Table 1: Response time (seconds), cost (tokens/agent/turn), and performance (%) of MAAC and other baselines (single-model, multi-agent test-time interaction, and MA-REINFORCE) on article writing, code generation, and Minecraft over 5 runs. Response time is test on a *GeForce RTX 5090*. Underlines and **bolds** indicate the best performance across MARL and all baselines on each dataset.

with coordination-centric objectives, prompt-based multi-agent methods produce solutions with low quality. Both RL and MARL fine-tuning guide agents to generate concise, high-quality content, leading to shorter response times, lower token usage, and improved performance. MAGRPO and CoLLM-CC achieve similar performance after convergence, indicating that $K = 4$ samples are sufficient for accurate value estimation. MAGRPO has slightly higher variance and lower sample efficiency, as shown by a larger shaded region and slower convergence in Fig. 2a.

In coding collaboration, fine-tuning *Qwen2.5-Coder-7B* yields limited improvement, as coding logics vary substantially across training and test problems and no general strategy consistently improves pass rates across diverse tasks. Parallel execution and one-round discussion perform poorly, as agents lack timely information to accurately assess the correctness and functionality of others' functions. Sequential generation achieves performance comparable to the single-model baseline but incurs substantially higher inference latency because agents must wait for others' responses before proceeding. With $K = 4$, MAGRPO achieves comparably high pass rates as CoLLM-CC. The main generator provides fallback solutions for potential vulnerabilities in the auxiliary, and it effectively uses external feedback to correct errors. However, MAGRPO requires significantly more samples to converge in Fig. 2b, reaching stability at approximately 5000 timesteps, compared to 2000 timesteps for CoLLM-CC. CoLLM-DC exhibits oscillations in later stages (4500 timesteps) and thus cannot perform well. This instability is caused by sparse rewards in code generation, where a single incorrect token can invalidate functionality and induce abrupt reward drops.

In Minecraft games, the fine-tuned single model achieves the highest remaining health points in *HouseBuild*, as it does not need to infer other agents' material choices or attack behavior. Prompt-based multi-agent methods perform particularly poorly in this domain, likely because the raw models are not optimized with Minecraft building instructions. CoLLM-CC achieves the best IoU in *HouseBuild* by leveraging external hints and outperforms all other methods except the single model fine-tuned by AC. MAGRPO and CoLLM-DC perform substantially worse than CoLLM-CC. Due to the rapid growth of rollout tree, a smaller sampling $K = 2$ is needed, leading to slower convergence and higher variance. CoLLM-DC fails to converge on these tasks since non-stationarity accumulates across 4 turns, leading to more unstable value estimates.

6 Conclusion

Decentralized LLM collaboration accelerates inference and enables flexible deployment, making it promising. We developed centralized critic (CoLLM-CC) and decentralized critic (CoLLM-DC) methods to optimize cooperation among decentralized LLMs. Evaluations on writing, coding, and game-playing tasks show that MARL-based methods can achieve equal or better performance than a single larger model. Among MARL methods, MA-REINFORCE and CoLLM-DC struggle in long-horizon, sparse-reward settings due to low sample efficiency and convergence issues, while CoLLM-CC achieves the best performance with the lowest variance and highest sample efficiency.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in LLMs. *arXiv preprint arXiv:2402.14740*, 2024.

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`.

Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4311–4317. IEEE, 2024.

Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

Zheng Ding and Weirui Ye. Treegrpo: Tree-advantage grpo for online rl post-training of diffusion models. *arXiv preprint arXiv:2512.08153*, 2025.

Arthur Douillard, Qixuan Feng, Andrei A Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

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*, April 2018.

Yeqi Gao, Zhao Song, and Junze Yin. Gradientcoin: A peer-to-peer decentralized large language models. *arXiv preprint arXiv:2308.10502*, 2023.

Sukumar Ghosh. *Distributed systems: an algorithmic approach*. Chapman and Hall/CRC, 2006.

Daya Guo, Dejian Yang, Haowei Zhang, et al. Deepseek-r1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Haoyang Hong, Jiajun Yin, Yuan Wang, Jingnan Liu, Zhe Chen, Ailing Yu, Ji Li, Zhiling Ye, Hansong Xiao, Yefei Chen, et al. Multi-agent deep research: Training multi-agent systems with m-grpo. *arXiv preprint arXiv:2511.13288*, 2025.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. Reinforce++: Stabilizing critic-free policy optimization with global advantage normalization, 2025.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for llm agent reinforcement learning. *arXiv preprint arXiv:2509.21240*, 2025.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Ziniu Li, Tian Xu, Yushun Zhang, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: A simple, effective, and efficient method for aligning large language models. *arXiv preprint arXiv:2310.10505*, 2023.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. Marft: Multi-agent reinforcement fine-tuning. *arXiv preprint arXiv:2504.16129*, 2025.

Shalev Lifshitz, Sheila A McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. *arXiv preprint arXiv:2502.20379*, 2025.

Bo Liu, Leon Guertler, Simon Yu, Zichen Liu, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, et al. Spiral: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. *arXiv preprint arXiv:2506.24119*, 2025a.

Mickel Liu, Liwei Jiang, Yancheng Liang, Simon Shaolei Du, Yejin Choi, Tim Althoff, and Natasha Jaques. Chasing moving targets with online self-play reinforcement learning for safer language models. *arXiv preprint arXiv:2506.07468*, 2025b.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, et al. Gdpo: Group reward-decoupled normalization policy optimization for multi-reward rl optimization. *arXiv preprint arXiv:2601.05242*, 2026a.

Shuo Liu, Zeyu Liang, Xueguang Lyu, and Christopher Amato. LLM collaboration with multi-agent reinforcement learning. In *Proceedings of the 40th Annual AAAI Conference on Artificial Intelligence*, 2026b.

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

Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning. *arXiv preprint arXiv:2102.04402*, 2021.

Xueguang Lyu, Andrea Baisero, Yuchen Xiao, Brett Daley, and Christopher Amato. On centralized critics in multi-agent reinforcement learning. *Journal of Artificial Intelligence Research*, 77:295–354, 2023.

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

Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie Pack Kaelbling. Learning to cooperate via policy search. *arXiv preprint cs/0105032*, 2001.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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

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. *arXiv preprint arXiv:2303.14100*, 2023.

Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

Vighnesh Subramaniam, Yilun Du, Joshua B Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. Multiagent finetuning: Self improvement with diverse reasoning chains. *arXiv preprint arXiv:2501.05707*, 2025.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

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. *arXiv preprint arXiv:2211.14275*, 2022.

Wiebe Van der Hoek and Michael Wooldridge. Multi-agent systems. *Foundations of Artificial Intelligence*, 3:887–928, 2008.

Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall. A note on distributed computing. In *International Workshop on Mobile Object Systems*, pp. 49–64. Springer, 1996.

Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

Linyu Wu, Xiaoyuan Liu, Tianneng Shi, Zhe Ye, and Dawn Song. Deserve: Towards affordable offline llm inference via decentralization. *arXiv preprint arXiv:2501.14784*, 2025a.

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. *arXiv preprint arXiv:2308.08155*, 2023.

Shirley Wu, Michel Galley, Baolin Peng, Hao Cheng, Gavin Li, Yao Dou, Weixin Cai, James Zou, Jure Leskovec, and Jianfeng Gao. Collabllm: From passive responders to active collaborators. *arXiv preprint arXiv:2502.00640*, 2025b.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Zhicheng Yang, Zhijiang Guo, Yinya Huang, Xiaodan Liang, Yiwei Wang, and Jing Tang. Treerpo: Tree relative policy optimization. *arXiv preprint arXiv:2506.05183*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

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. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24611–24624. Curran Associates, Inc., 2022.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Kaiyan Zhang, Runze Liu, Xuekai Zhu, Kai Tian, Sihang Zeng, Guoli Jia, Yuchen Fan, Xingtai Lv, Yuxin Zuo, Che Jiang, Ziyang Liu, Jianyu Wang, Yuru Wang, Ruotong Zhao, Ermo Hua, Yibo Wang, Shijie Wang, Junqi Gao, Xinwei Long, Youbang Sun, Zhiyuan Ma, Ganqu Cui, Lei Bai, Ning Ding, Biqing Qi, and Bowen Zhou. MARTI: a framework for multi-agent LLM systems reinforced training and inference, 2025.

A PROOFS

Here we provide the full proof of Proposition 3.1.

Proof. We show that, when the episodes are not terminated early and expand into a full K-ary tree, and assuming gradient estimates are independent for $\{a_{i,t}^1, \cdots, a_{i,t}^K\}$, the variance of $g_{i,t}^K$ scales inversely with K.

For agent i at time t, define

$$g_{i,t}^k = \rho_{i,t}^k \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t}^k \mid h_{i,t})\, G_t^k.$$

Then, by variance decomposition, we have

$$\begin{aligned}
&\mathrm{Var}(\bar{g}_{i,t} \mid h_{i,t}) \\
&= \mathrm{Var}\left(\frac{1}{K}\sum_{k=1}^{K} g_{i,t}^k \mid h_{i,t}\right) \\
&= \frac{1}{K^2}\sum_{k=1}^{K} \mathrm{Var}\big(g_{i,t}^k \mid h_{i,t}\big) + \frac{2}{K^2}\sum_{1\le k<l\le K} \mathrm{Cov}\big(g_{i,t}^k, g_{i,t}^l \mid h_{i,t}\big)
\end{aligned},$$

where k and l are different i.i.d. samples at t. Since $\{a_{i,t}^1, \cdots, a_{i,t}^K\}$ are i.i.d., we have $\mathrm{Cov}(g_{i,t}^k, g_{i,t}^l | h_{i,t}) = 0$. Assuming the episode is not terminated early, at $t+1$, we have ℓ new rollouts for each $a_{i,t}^k$, and so on, we have $\mathrm{leaf} = K^{H-1-t}$ leaf nodes and returns at $t = H-1$.

$$\begin{aligned}
&\mathrm{Var}(\bar{g}_{i,t} \mid h_{i,t}) \\
&= \frac{1}{K}\mathrm{Var}\big(g_{i,t}^k \mid h_{i,t}\big) \\
&= \frac{1}{K}\mathrm{Var}\big(\rho_{i,t}^k \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t}^k \mid h_{i,t})\, G_t^k \mid h_{i,t}^k\big) \\
&= \frac{1}{K}\mathrm{Var}\left(\rho_{i,t}^k \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t}^k \mid h_{i,t})\, \frac{1}{K}\sum_{\ell=1}^{K} G_{t+1}^{\ell} \mid h_{i,t}^k\right) \\
&= \frac{1}{K}\mathrm{Var}\left(\rho_{i,t}^k \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t}^k \mid h_{i,t})\, \frac{\sum_{\mathrm{leaf}=1}^{K^{H-1-t}} G_{H-1}^{\mathrm{leaf}}}{K^{H-1-t}} \mid h_{i,t}^k\right) \\
&= \frac{1}{K}\frac{1}{(K^{H-1-t})^2}\sum_{\mathrm{leaf}=1}^{K^{H-1-t}} \mathrm{Var}\big(\rho_{i,t}^k \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t}^k \mid h_{i,t})\, G_t^{\mathrm{leaf}} \mid h_{i,t}^k\big) \\
&\quad + \frac{2}{K}\frac{2}{(K^{H-1-t})^2}\sum_{1\le \mathrm{leaf_1}\le \mathrm{leaf_2}\le K}^{K^{H-1-t}} \mathrm{Cov}\big(g_{i,t}^{\mathrm{leaf_1}}, g_{i,t}^{\mathrm{leaf_2}} \mid h_{i,t}^k\big)
\end{aligned}$$

Assuming the $\{g_{i,t}^1, \cdots, g_{i,t}^K\}$ and are independent,[1] $\mathrm{Cov}(g_{i,t}^{\mathrm{leaf\_1}}, g_{i,t}^{\mathrm{leaf\_2}} | h_{i,t}^k) = 0$.

Thus, we have,

$$\begin{aligned}
&\mathrm{Var}(\bar{g}_{i,t} \mid h_{i,t}) \\
&= \frac{1}{K^{H-t}}\mathrm{Var}\big(\rho_{i,t}^k \nabla_{\theta_i} \log \pi_{\theta_i}(a_{i,t}^k \mid h_{i,t})\, G_t^{\mathrm{leaf}} \mid h_{i,t}^k\big) \\
&= \frac{\sigma^2}{K^{H-t}}
\end{aligned}$$

□

[1] However, this independent gradient assumption is usually hard to satisfy in Dec-POMDP, because dialog histories with the same prefixes usually have positively correlated values.

B CoLLM-DC

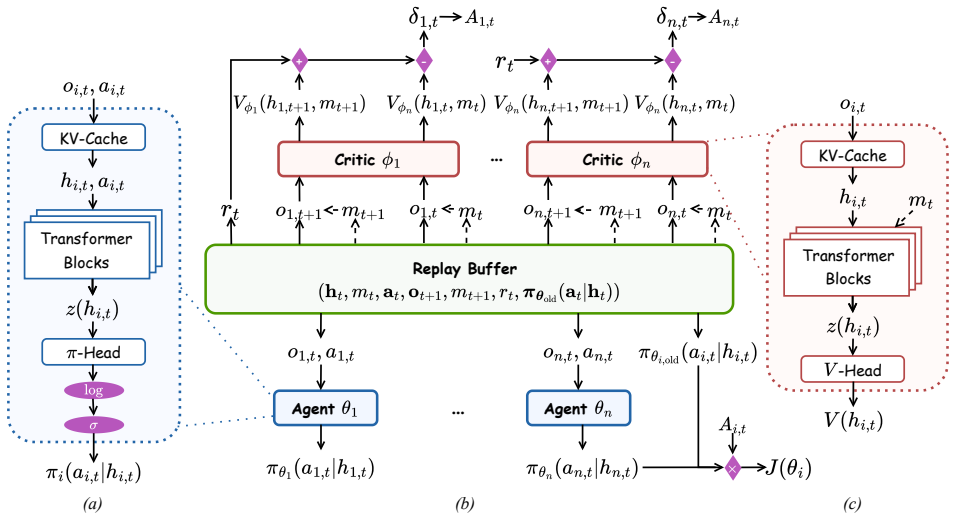


Figure 3: CoLLM-DC framework: *(a)* The agent structure; *(b)* The overall decentralized-critic architecture; *(c)* The critic structure.

Algorithm 2: CoLLM-DC

Input: Taskset $\mathcal{D}$, LLM agents $\{\pi_{\theta_i}\}_{i\in\mathcal{I}}$, decentralized LLM critics $\{V_{\phi_i}\}_{i\in\mathcal{I}}$, learning rates α_π, α_V, discount γ, horizon H, replay buffer $\mathcal{B}$, training epochs E
for each episode **do**
  Roll out same as CoLLM-CC in Alg. 1 (line 3-15)
  for training epoch $e = 1, \cdots, E$ **do**
    Sample a minibatch of joint transitions β from $\mathcal{B}$
    for each agent $i \in \mathcal{I}$ **do**
      for each sample agent i's transition β_i, $(h^b_{i,t}, m^b_t, a^b_{i,t}, r^b_t, o^b_{i,t+1}, m^b_{t+1}\pi^b_{\theta_i,\text{old}}(a^b_{i,t}|h^b_{i,t}))$ $\in \beta_i$ **do**
        Calculate TD loss $\mathcal{L}^b_i(\phi_i)$ (Eq. 3)
        Calculate $\pi_{\theta_i}(a^b_{i,t}|h^b_{i,t})$ with TF
        Calculate $\nabla_{\theta_i} J^b(\theta_i)$ (Eq. 2)
      end for
      Update critic $\phi_i \leftarrow \phi_i - \alpha_V \frac{1}{|\beta|}\sum_b \nabla_{\phi_i}\mathcal{L}^b_i(\phi_i)$
      Update actor $\theta_i \leftarrow \theta_i + \alpha_\pi \frac{1}{|\beta|}\sum_b \nabla_{\theta_i} J^b(\theta_i)$
    end for
  end for
end for
Output: $\{\pi_{\theta_i}\}_{i\in\mathcal{I}}$

Fig. 3 and Alg. 2 show the training procedure of CoLLM-DC. The rollout and actor updates are the same as those in CoLLM-CC (Alg. 1). The only difference is in the training phase. In each training epoch, a minibatch β_i of joint transitions is drawn from $\mathcal{B}$. For each sample, $(h^b_{i,t}, m^b_t, a^b_{i,t}, r^b_t, o^b_{i,t+1}, m^b_{t+1}\pi^b_{\theta_i,\text{old}}(a^b_{i,t}|h^b_{i,t})) \in \beta_i$, each agent i computes its own TD loss $\mathcal{L}^b_i(\phi_i)$ using a decentralized critic conditioned only on its local history, following the structure shown in Fig. 3c. The probability $\pi_{\theta_i}(a^b_{i,t}|h^b_{i,t})$ under the current policy is computed by a TF pass and is used to calculate the policy gradient $\nabla_{\theta_i} J^b(\theta_i)$ according to Eq.2 in Fig.3a. The gradients of all samples in β_i are averaged to update each agent's critic ϕ_i and actor θ_i independently, $\forall i \in \mathcal{I}$.

It is noteworthy that the CoLLM-DC architecture in Fig. 3 has an efficient variant based on parameter sharing. Unlike CoLLM-CC, the agents in Fig. 3*a* and Fig. 3*c* learn the same latent representation $z(h_{i,t})$, and thus can potentially share a single model to accelerate learning. However, because the optimization objectives in Eq. 3 and 2 are different, such parameter sharing may introduce gradient interference during training.

C Experimental Settings

We introduce the experimental settings in Fig. 2 and Table 1.

C.1 Datasets

We list the preprocessed datasets we used for our training.

- ***Writing Collaboration***
  - Training set:
    - *TLDR*[0:1000]
  - Test set:
    - *TLDR*[1000:1100]
- ***Code Collaboration***
  - Training set: *CoopHE*[0:66]
  - Test set: *CoopHE*[66:82]
- ***Minecraft Game-Playing***
  - Training set:
    - *HouseBuild*[0:8]
  - Test set:
    - *HouseBuild*[8:10]

C.2 Architectures

We list the architectures of the agent and critic models used in our training.

- ***Writing Collaboration***
  - Agents
    - *Qwen3-1.7B*
    - *Qwen3-1.7B*
  - Critic (if applicable): *Qwen3-1.7B*
  - Temperature: 0.7
  - Top-p: 0.9
  - Top-k: null
  - Max output tokens: 256
- ***Code Collaboration***
  - Agents
    - *Qwen2.5-Coder-3B*
    - *Qwen3-4B-Instruct-2507*
  - Critic (if applicable): *Qwen2.5-Coder-3B*
  - Temperature: 0.6
  - Top-p: 0.6
  - Top-k: null
  - Max output tokens: 256
- ***Minecraft Game-Playing***

  - Agents
    - *Qwen2.5-3B-Instruct*
    - *Qwen3-4B-Instruct-2507*
  - Critic (if applicable): *Qwen3-4B-Instruct-2507*
  - Temperature: 0.6
  - Top-p: 0.6
  - Top-k: null
  - Max output tokens
    - *HouseBuild*: 512

C.3 HYPERPARAMETERS

We show the key hyperparameters used in MAGRPO, CoLLM-DC, and CoLLM-CC.

- ***Writing Collaboration***
  - Number of turns: 1
  - Number of generations: 4
  - Rollout buffer size: 4
  - Number of train epochs
    - MAGRPO: 2
    - CoLLM-DC/CoLLM-CC: 20
  - Agent learning rate: 5×10^{-6}
  - Critic learning rate (if applicable): 3×10^{-6}
  - Advantage clip: 0.2
  - Number of evaluation samples: 4
- ***Code Collaboration***
  - Number of turns: 2
  - Number of generations: 4
  - Rollout buffer size
    - MAGRPO: 16
    - CoLLM-DC/CoLLM-CC: 4
  - Number of train epochs
    - MAGRPO: 8
    - CoLLM-DC/CoLLM-CC: 80
  - Agent learning rate
    - MAGRPO: 2×10^{-5}
    - CoLLM-DC/CoLLM-CC: 5×10^{-6}
  - Critic learning rate (if applicable): 3×10^{-6}
  - Advantage clip: 0.2
  - Number of evaluation samples: 4
- ***Minecraft Game-Playing***
  - Number of turns: 4
  - Number of generations: 2
  - Rollout buffer size
    - MAGRPO: 2
    - CoLLM-DC/CoLLM-CC: 1
  - Number of train epochs
    - MAGRPO: 16
    - CoLLM-DC/CoLLM-CC: 120
  - Agent learning rate
    - MAGRPO:

  - · *HouseBuild*: 1×10^{-5}
  - ∗ CoLLM-DC/CoLLM-CC:
    - · *HouseBuild*: 5×10^{-6}
- Critic learning rate (if applicable):
  - ∗ CoLLM-DC/CoLLM-CC:
    - · *HouseBuild*: 3×10^{-6}
- Advantage clip: 0.05
- Number of evaluation samples: 2

D Additional Results

We provide additional results in our experiments.

D.1 Pass@k Results on CoopHE

Method	Pass@1	Pass@3	Pass@5	Pass@10
Single-Agent	56.3	58.7	61.9	62.5
GRPO	61.8	62.0	62.2	62.8
AC	62.2	62.6	63.0	63.3
Parallel	50.0	51.1	68.8	68.8
Pipeline	62.5	**76.4**	77.3	85.3
Discussion	25.0	31.5	34.3	74.7
MAGRPO	74.3	76.0	77.5	85.3
CoLLM-DC	59.1	60.5	60.5	62.8
CoLLM-CC	**75.2**	75.9	**77.8**	**86.5**

Table 2: Pass@k results (1, 3, 5, 10) on *CoopHE*. **Bolds** indicate the best performance. Results are averaged over 5 runs.

Table 2 presents the pass@k performance of coding collaboration on *CoopHE*. Fine-tuning with GRPO and AC yields marginal improvements over the raw model. However, this improvement over the given model is not due to acquired algorithmic knowledge or increased capacity. Instead, the training primarily refines the model's policy, increasing the likelihood of producing correct solutions that already lie within its representational scope Yue et al. (2025).

Most prompt-based multi-agent approaches underperform single-model baselines without proper optimization. Agents do not have timely communication to reason about each other's correctness or functionalities. Although this challenge can be overcome through sequential execution, it reduces inference speed because agents take turns responding.

MARL methods can achieve performance comparable to or better than that of a single larger model. CoLLM-CC consistently achieves the best results across most pass@k results, because the main agent is guided to infer the auxiliary agent's functionality and provide fallback solutions when the auxiliary utilities are vulnerable, e.g., the main can provide a boundary condition manipulation. Also, through MARL optimization, the main completes the remaining components of the implementation, thereby accelerating inference through parallel execution. CoLLM-DC performs worse than CoLLM-CC and MAGRPO due to non-convergence (Fig. 2b).

D.2 Training Overhead

We use the setting of *CoopHE* in Appendix C as a representative to compare MARL training overhead.

Metric	MAGRPO	CoLLM-DC	CoLLM-CC
#Epochs	8	80	80
#Rollouts	16	2	2
#Samples	9640	8592	8438
#Updates	603	2148	2110
Duration	4.5	13.4	11.1
VRAM	93.8	126.3	107.4

Table 3: Training overhead of MAGRPO, CoLLM-DC, and CoLLM-CC on *CoopHE*, under the settings of Appendix C. Metrics include the number of epochs, rollouts per episode, total samples used, policy updates, training duration (*H200* hours), and VRAM (GB) usage. Results are averaged over 5 runs.

Since MAGRPO is an instance of K-sampling, the rollout in an H-horizon episode forms a K-ary tree and satisfies Proposition 3.2. Under the hyperparameter setting in Appendix C, with $K = 4$ and $H = 2$, MAGRPO produces $K^H = 16$ rollouts and uses $\sum_{k=1}^{H} K^H = 20$ samples, assuming no early termination is triggered. In contrast, both CoLLM-DC and CoLLM-CC generate a single rollout consisting of $H = 2$ samples. To ensure a comparable number of training samples for MARL, we therefore train CoLLM-DC and CoLLM-CC for 10 times as many epochs as MAGRPO. Also, MAGRPO operates with a larger effective minibatch size. Accordingly, the agent's learning rate is designed to scale proportionally, resulting in fewer updates.

Training CoLLM-DC and CoLLM-CC requires substantially longer time and more GPU memory than MAGRPO, as critic LLMs must be maintained throughout training. However, this overhead does not scale linearly with the number of LLMs in the systems ($126.3 \ll 93.8 \times 2$). This is because the critic provides lower-variance gradient estimates, thereby reducing the number of samples that need to be retained in GPU memory for back-propagation. Furthermore, CoLLM-CC employs a single centralized critic, whose update cost is lower than maintaining n independent critics of the same size in CoLLM-DC, thereby reducing training time. As CoLLM-CC converges faster (Fig. 2b) and a lot of samples trigger early termination during the early stages of training, it ultimately uses the fewest samples.

E Prompt Design Details

E.1 Writing Collaboration

In the *TLDR* summarization, the instructions for each agent are as follows.

```
Summary Agent
Create a concise summary response to this post.
Query: {prompt}
Instructions: Provide a brief and focused summary in a few sentences

Elaboration Agent
Create a detailed summary response to this post.
Query: {prompt}
Instructions: You should use transition words to improve flow
```

E.2 Coding Collaboration

For CoopHE, we extract the `entry_point`, `params` from the `prompt` field and instruct the agents as follows.

```
Auxiliary Agent
Create a helper function for this coding problem.
Problem: {prompt}
Instructions:
```

```
- Output ONLY the function code, no explanations or examples
- Do NOT include markdown code blocks (```python)
- Do NOT include any text before or after the function
- Do NOT include test cases or example usage
- Create a helper function named 'aux' that can assist the main
    function
- The function should return useful data for solving the problem

Your output should follow this format:
def aux(...):
    # your code here
    return result
```

Main Agent

```
Solve this coding problem by implementing the required function.
Problem: {prompt}
You have access to a helper function: aux(...)

Instructions:
- Output ONLY the function code, no explanations or examples
- Do NOT include markdown code blocks (```python)
- Do NOT include any text before or after the function
- Do NOT include test cases or example usage
- Do NOT redefine the aux() function
- Implement ONLY the '{entry_point}' function as specified
- You can call aux() to assign a value to a variable within your
    function if helpful

Your output should follow this format:
def {entry_point}({params}):\n # your function code here\nreturn
    result\n
```

To improve the generated code, agents receive additional feedback in addition to the initial problem description. This feedback comes from the static analyzer and sandbox tests, and is appended to the prompts for subsequent turns.

```
Static and execution diagnostics:
- Main definition: FOUND (Main function prime_fib defined)
- Syntax: OK (Combined code syntax OK)
- Tests: 4/5 passed
assert candidate(1) == 2
AssertionError: expected 2, got 1

Revise your prime_fib accordingly.
```

E.3 MINECRAFT

In *HouseBuild*, since the player can construct blocks flexibly at any adjacent location, the positions of players are assumed to be the center. We employ a simulator to generate `spider_num` spiders from arbitrary locations on the map. It also computes the player's health points `player_hp`, and the damage inflicted on spiders `spider_dmg`. The predefined player attack value is `player_atk`, while each spider is assumed to have an attack value `spider_atk`.

```
You are Player i in an n-person Minecraft building team. You will
    place SOME of the blocks for the final build. Output must be
    Minecraft commands only (no markdown, no code fences, no extra
    text).

Task: Build the 3D structure from the provided y-axis slices.

Available blocks (use ONLY these): {block_agent_lines}

Layers (ascending WORLD y). Each layer is a set of rectangles in WORLD
     (x,z) coords:
```

```
  - Format: {(x1, y1, z1, x2, y2, z2 block_id), (x1, y1, z1,
  x2, y2, z2 block_id)}

WORLD bbox (inclusive):
  - from: {world_bbox_from}
  - to:   {world_bbox_to}

Threat check: There are {spider_num} spiders nearby; you have {
    player_hp} HP. Spider attack list: {spider_atk}, total damage dmg
    ={spider_dmg}. If you want to kill them, output the attack command
    : /damage @e[type=spider,limit=1] {player_atk}
minecraft:player attack. All other commands should be normal building
    commands.

Constraints:
  - Output ONLY Minecraft commands, one per line.
  - Allowed commands: /fill and /kill
  - Fill format: /fill x1 y1 z1 x2 y2 z2 block
  - Use absolute integer coordinates only (no ~).
  - Use ONLY blocks from the legend.
  - Every coordinate must be within the bbox.
```

And we provide DC and CC with the following global information, m_t, during training.

```
You are at turn = {turn} over a {num_turns}-turn episode.
Your last-turn reward is {last_reward} / {reward_max}.
```

F Reward Design Details

Reinforcement Learning with Verifiable Rewards (RLVR) provides objective and reliable training signals. However, most RLVR methods in a multi-turn setting rely on terminal rewards, resulting in extremely sparse feedback that complicates credit assignment across multiple agents. Recent work addresses this by introducing process-based or intermediate rewards, often instantiated as manually designed rubrics Uesato et al. (2022); Lifshitz et al. (2025); Wu et al. (2025b). However, these approaches typically require substantial manual intervention for rubric engineering. It remains unclear whether such designs are still generalizable, consistent, and aligned with the intended objectives.

We provide verifiable reward signals at each turn, enabling fine-grained and objective supervision throughout the interaction. However, the cumulative return becomes dependent on the horizon, resulting in a dynamic return scale that must be carefully handled in policy optimization. We solve this problem by providing the critic with global information m_t, as shown in critic prompts in Appendix E.

F.1 Writing Collaboration

We evaluate *TLDR* summarization along three dimensions: structural quality, style consistency, and logical coherence. Structural wellness measures the relative completion length and lexical diversity (unique-word ratio excluding stopwords). Full rewards are assigned when the length ratio lies in 1.6-3.2×, and the unique-word ratio is $\geq$2.0×; values within broader tolerances (1.1-5.0× for length and 1.3–2.0× for uniqueness) receive proportionally scaled rewards, while out-of-range cases incur zero reward and early termination. Style consistency is quantified by the Jaccard similarity of vocabularies between completions (excluding stopwords), capped at 0.03 to balance lexical consistency with vocabulary expansion. Logical coherence is evaluated through transition-word usage in the combined text across 12 functional categories, with rewards scaled by category diversity as $\min(0.6\log(\#\text{categories}+1), 1)$.

F.2 Coding Collaboration

We evaluate coding collaboration along four dimensions: structural integrity, syntactic correctness, test pass rate, and cooperation quality. Structural integrity verifies that both auxiliary and main functions are properly defined, with valid signatures and return statements; failure to define the main

function results in immediate termination. Syntactic correctness assesses whether the concatenated code parses without errors, verified via the Abstract Syntax Tree (*AST*) library. Any syntax error leads to evaluation termination to prevent runtime failures. The test pass rate measures the proportion of unit tests that successfully pass within an 8-second timeout, with rewards scaled by the number of successful assertions (tests) $\frac{\#\text{passed\_tests}}{\#\text{total\_tests}}$, and termination triggered if no tests pass. Cooperation quality assigns a base bonus when the main function invokes the auxiliary, with additional rewards granted when the main function exhibits substantive logic beyond a simple wrapper. A deduction is applied if the main function calls the auxiliary but discards its return value, penalizing superficial cooperation.

F.3 MINECRAFT

In `HouseBuild`, we similarly employ the coverage rate and redundancy rate to encourage accurate and efficient construction. Also, agents are penalized for damage inflicted by spiders as $\min\left(1, \frac{\text{spider\_total\_dmg}}{\text{player\_hp}}\right) \times 0.2$.

G COMPUTE RESOURCES

The compute resources used in our experiments (Sec. 5) for training and evaluation are listed below.

G.1 TRAINING DEVICES

- ***Writing Collaboration***
  - Type: GPU Cluster
    CPU: NVIDIA Grace ARMv9
    GPU: 1× NVIDIA Hopper H100
- ***Coding Collaboration***
  - Type: GPU Cluster
    CPU: Intel Xeon Platinum 8558
    GPU: 1× NVIDIA Hopper H200
  - Type: GPU Cluster
    CPU: Intel Xeon Gold 5318Y
    GPU: 1× NVIDIA Hopper H200
  - Type: GPU Cloud Instance
    CPU: AMD EPYC 9755
    GPU: 1× NVIDIA Hopper H200
  - Type: GPU Cloud Instance
    CPU: Intel Xeon Platinum 8568Y+
    GPU: 1× NVIDIA Hopper H200
- ***Minecraft Building Collaboration***
  - Type: GPU Cluster
    CPU: Intel Xeon Platinum 8592+
    GPU: 1× NVIDIA Blackwell B200
  - Type: GPU Cluster
    CPU: Intel Xeon Platinum 8558
    GPU: 1× NVIDIA Hopper H200
  - Type: GPU Cloud Instance
    CPU: Intel Xeon Platinum 8568Y+
    GPU: 1× NVIDIA Blackwell B200
  - Type: GPU Cloud Instance
    CPU: AMD EPYC 9655
    GPU: 1× NVIDIA Hopper H200
  - Type: GPU Cloud Instance
    CPU: AMD EPYC 9755
    GPU: 1× NVIDIA Hopper H200

  - Type: GPU Cloud Instance
    CPU: Intel Xeon Platinum 8592+
    GPU: 1× NVIDIA Hopper H200

G.2 Inference Device

- ***All Tasks***
  - Type: Standalone Workstation
    CPU: AMD Ryzen 9 9950X
    GPU: 1× NVIDIA GeForce RTX 5090

H Code and Dataset

Our code and datasets are available in the following repositories (Version 1.3.6):

- `CoMLRL`: **C**ooperative **M**ulti-**LLM** **R**einforcement **L**earning (CoMLRL) is an open-source library for training multiple LLMs to collaborate using Multi-Agent Reinforcement Learning (MARL). It provides implementations of various MARL trainers for decentralized LLM collaboration. `https://github.com/OpenMLRL/CoMLRL/releases/tag/v1.3.6`.
- `LLM_Collab_Writing`: This repository contains the writing collaboration experiments. `https://github.com/OpenMLRL/LLM_Collab_Writing/releases/tag/v1.3.6`.
- `LLM_Collab_Code_Generation`: This repository contains the coding collaboration experiments. `https://github.com/OpenMLRL/LLM_Collab_Code_Generation/releases/tag/v1.3.6`.
- `LLM_Collab_Minecraft`: This repository contains the collaborative game-playing experiments in Minecraft. `https://github.com/OpenMLRL/LLM_Collab_Minecraft/releases/tag/v1.3.6`.

I Limitations and Future Work

Our work has several limitations. Although our MAAC methods (i.e., CoLLM-DC and CoLLM-CC) can, in principle, be applied to online learning, due to MAGRPO's low sample efficiency (Proposition 3.2), we are unable to extend our experiments to longer-horizon tasks for comparison. Also, due to constraints on computing resources, our experiments are limited to proof-of-concept settings. How LLM-based collaboration can scale to larger multi-agent systems with more diverse and heterogeneous agents remains an open question. In addition, as discussed in Appendix B, there exists a more efficient variant of CoLLM-DC. Although it has certain drawbacks, it remains worth exploring whether these limitations can be mitigated or whether this trick can be applied to CC methods. Finally, our methods assume a strictly decentralized setting with concurrent execution and no communication. How to design methods with relaxed constraints is still an open question.