Heterogeneous Group-Based Reinforcement Learning for LLM-based Multi-Agent Systems

Guanzhong Chen[1] **Shaoxiong Yang** [2] **Chao Li** [2] **Wei Liu** [2] **Jian Luan** [2] **Zenglin Xu**[3]

[1]Harbin Institute of Technology, Shenzhen [2]MiLM Plus, Xiaomi Inc. [3]Fudan University
muxichenz@outlook.com {yangshaoxiong, lichao75, liuwei40, luanjian}@xiaomi.com zenglinxu@fudan.edu.cn

arXiv:2506.02718v1 [cs.LG] 3 Jun 2025

Abstract

Large Language Models (LLMs) have achieved remarkable success across diverse natural language processing tasks, yet their deployment in real-world applications is hindered by fixed knowledge cutoffs and difficulties in generating controllable, accurate outputs in a single inference. Multi-agent systems (MAS) built from specialized LLM agents offer a promising solution, enabling dynamic collaboration and iterative reasoning. However, optimizing these systems remains a challenge, as conventional methods such as prompt engineering and supervised fine-tuning entail high engineering overhead and limited adaptability. Reinforcement learning (RL), particularly multi-agent reinforcement learning (MARL), provides a scalable framework by refining agent policies based on system-level feedback. Nevertheless, existing MARL algorithms—such as Multi-Agent Proximal Policy Optimization (MAPPO)—rely on Critic networks, which can cause training instability and increase computational burden. To address these limitations and target the prototypical Multi-Agent Search System (MASS), we propose Multi-Agent Heterogeneous Group Policy Optimization (MHGPO), a novel Critic-free algorithm that guides policy updates by estimating relative reward advantages across heterogeneous groups of rollouts. MHGPO eliminates the need for Critic networks, enhancing stability and reducing computational overhead. Additionally, we introduce three group rollout sampling strategies that trade off between efficiency and effectiveness. Experiments on a multi-agent LLM-based search system demonstrate that MHGPO consistently outperforms MAPPO in both task performance and computational efficiency, without requiring warm-up, underscoring its potential for stable and scalable optimization of complex LLM-based MAS.

Introduction

Large language models (LLMs), such as the GPT series (Radford et al. 2018, 2019; Brown et al. 2020; OpenAI 2023) and LLaMA (Touvron et al. 2023a,b), have exhibited outstanding performance across a wide range of domains. Despite their remarkable capabilities in diverse natural language processing (NLP) tasks, the deployment of LLMs in real-world industrial settings remains nontrivial. On the one hand, the fixed knowledge cutoff of LLMs (Gao et al. 2024) constrains their ability to reason over information that emerges after their training period. On the other hand, although advanced reasoning models such as DeepSeek-R1 (Shao et al. 2024) have made substantial strides in addressing complex tasks, LLMs still face limitations in achieving controllable, trustworthy, and high-accuracy results within a single inference step. Consequently, the practical adoption of LLMs in specialized scenarios increasingly relies on the integration of *search tools* and *agent-based* frameworks.

An LLM Agent refers to a large language model encapsulated within an agent framework, equipped with task-specific functionalities and capable of leveraging its intrinsic commonsense knowledge and reasoning abilities to interact with external environments (e.g., tools and information sources) under a predefined communication protocol to accomplish designated objectives. Tackling complex tasks often necessitates the collaboration of multiple LLM Agents with specialized roles, collectively forming a multi-agent system (MAS). Such systems enable the decomposition of challenging problems into modular subtasks, which are solved through dynamic inter-agent interactions and LLM-driven reasoning. By assigning fine-grained *subtasks* to individual agents, a MAS reduces the cognitive load on each LLM while enhancing overall system interpretability, scalability, and controllability. To balance simplicity and generality, this paper focuses on a prototypical instantiation of MASs, namely the multi-agent search system (MASS), in which multiple LLM Agents collaboratively solve complex tasks by invoking external *search tools* as needed.

Numerous studies have explored the use of MASs to accomplish complex functionalities—for example, in search systems (Chang et al. 2025; Chen et al. 2025; Kulkarni et al. 2024; Wu et al. 2025; Zhao et al. 2025)—highlighting the impressive performance and adaptability of such architectures. Nonetheless, the deployment of sophisticated MASs in real-world settings remains a significant challenge. LLMs often face issues such as instruction non-compliance and suboptimal tool utilization. Taking MASS as a representative case, LLMs frequently fail to generate search queries that align well with the retrieval engine's expectations, resulting in the selection of irrelevant or low-quality content and ultimately hindering task completion. A common workaround involves manually crafting prompts or curating high-quality datasets for the Supervised Fine-tuning (SFT) of each agent. While this method can yield improvements,

it introduces substantial engineering overhead and lacks robustness to dynamic or evolving task demands. These limitations underscore the promise of Reinforcement Learning (RL)-based approaches for *end-to-end*, system-level optimization.

RL methods tailored for LLMs enhance model capabilities by rewarding desirable behaviors, thereby incentivizing exploration of strategies that maximize expected returns. In MASs where LLMs constitute the backbones of individual Agents, RL can similarly be leveraged to optimize Agents' competencies and, in turn, improve the overall system performance through end-to-end training—an approach known as Multi-Agent Reinforcement Learning (MARL). By assigning rewards based on the system-level output and fine-tuning the underlying LLM accordingly, MARL facilitates adaptive and flexible refinement of LLMs to better satisfy the unique requirements of tasks within the MAS.

Existing studies have applied Proximal Policy Optimization (PPO) (Schulman et al. 2017; Yu et al. 2022) to optimize MAS systems (Liao et al. 2025a; Chen et al. 2025). In this framework, the LLM backbone operates as the *actor*, while a comparably large-scale *critic* network is employed to estimate the expected return of actions, enabling model updates via policy gradient methods. However, within the MAS context, the critic must accommodate the diverse outputs generated by heterogeneous agent types, rendering it prone to instability due to task diversity. Moreover, the critic incurs substantial memory and computational overhead, which can significantly hinder system efficiency and scalability.

To address these limitations, this paper explores the concept of recent Group-based Optimization Algorithms (GOAs), such as Group Relative Policy Optimization (GRPO) (Schulman et al. 2016) and DAPO (Yu et al. 2025). These algorithms have demonstrated strong performance while reducing resource requirements by eliminating the need for a critic model and employing a simplified advantage estimation based on relative rewards computed within group rollouts. This approach holds significant potential for tackling complex scenarios such as MASS, yet remains largely unexplored. Consequently, we present the first systematic investigation of GOAs applied to MASs, making the following key contributions:

- We propose Multi-Agent Heterogeneous Group Policy Optimization (MHGPO), a novel method that optimizes MASs via a two-phase procedure involving multi-agent rollout path sampling and backward reward propagation, where advantage estimation leverages relative rewards computed over *heterogeneous groups*.
- We introduce three rollout sampling strategies for MASs, namely Independent Sampling (IS), Fork-on-first (FoF), and Round-robin (RR), designed to explore the trade-off between efficiency and effectiveness in this context.
- We empirically validate the effectiveness of MHGPO on a three-agent MASS using mainstream question answering datasets. Our results demonstrate its superiority over PPO both in terms of performance gains and computational efficiency, notably without requiring warm-up.

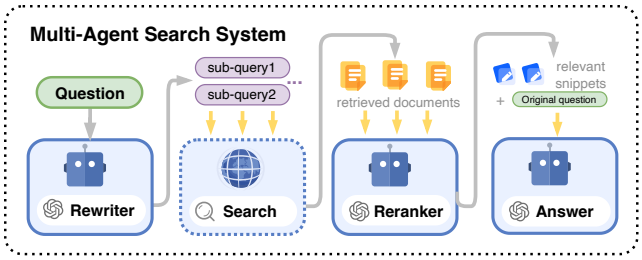

Figure 1: Illustration of the considered three-agent LLM search system: the *Rewriter* decomposes the question into sub-queries, the *Reranker* selects relevant snippets from retrieved documents, and the *Answerer* generates the final response.

Preliminary

Proximal Policy Optimization (PPO)

The PPO algorithm, when applied to LLM optimization, follows the standard Actor-Critic framework comprising three core components: the Actor, Critic, and Reference models, parameterized by θ, ϕ, and θ_{ref}, respectively. The Actor, representing the policy to be optimized, generates a response o given an input question q. The Critic estimates the value function V, providing a scalar assessment of the Actor's output based on the current state (i.e., the question and response). The Reference model constrains the Actor's updates, ensuring stability during training. The objective of PPO is to maximize:

$$\begin{aligned}\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D},o_{\le t}\sim\pi_{\theta_{\text{ref}}}(\cdot|q)} \\ [\min(r_t\hat{A}_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)\hat{A}_t)],\end{aligned} \tag{1}$$

where $r_t = \frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{\theta_{\text{ref}}}(o_t|q,o_{<t})}$ is the ratio of importance sampling, which is used to constrain the magnitude of the policy update. And $\hat{A}_t$ is the advantage estimate at the t-th time step. In PPO, generally, Generalized Advantage Estimation (GAE) is used for its calculation: For the l-th step (token) of the response o generated by the actor, the Critic gives a value estimate V_l, and the reward model (or scoring function) assigns a reward R_l to it, where $l = 0, 1, 2, \ldots, t$. $\hat{A}_t$ is estimated as:

$$\hat{A}_t = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l},\ \delta_l = R_l + \gamma V_{l+1} - V_l, \tag{2}$$

where $0 \le \gamma, \lambda \le 1$ are the hyperparameters of PPO and GAE respectively.

Group Relative Policy Optimization (GRPO)

Compared with PPO, instead of relying on a Critic model to conduct advantage evaluation, GRPO directly uses the within-group relative rewards as the basis for advantage evaluation. Specifically, for an input question q, GRPO will sample a group of G responses $\{o_i\}_{i=1}^G$. Thus, the advantage of the i-th response (shared across all time steps or only applied to the last time step) can be calculated as:

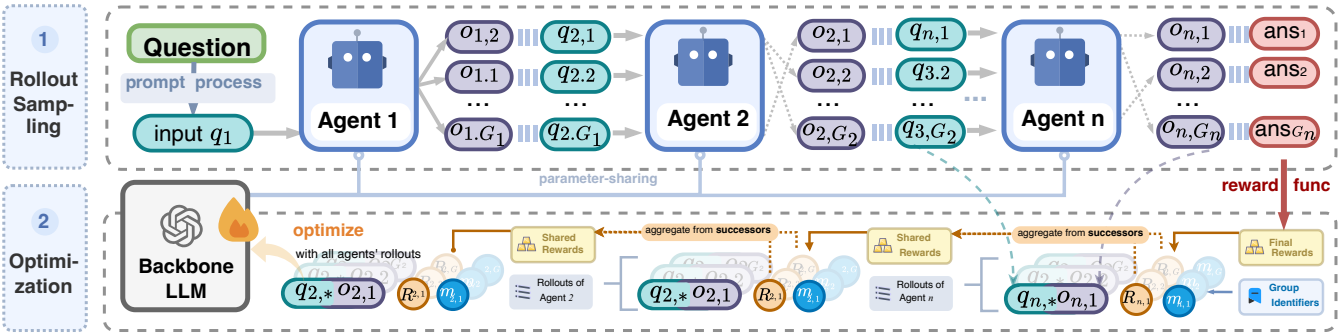

Figure 2: The overall optimization framework of the proposed MHGPO. For each MAS input (question), multi-agent rollout sampling is first performed (upper part). After computing the rewards, backward reward propagation is conducted. Once the intermediate rollout rewards are obtained, the backbone network parameters are updated (lower part).

$$\hat{A}_i = \frac{R_i - \text{mean}(\{R_i\}_{i=1}^G)}{\text{std}(\{R_i\}_{i=1}^G)} \tag{3}$$

Multi-Agent LLM Search System

In this paper, we utilize a simple yet effective three-agent MASS as an example to investigate MARL algorithms, as illustrated in Figure 1. Following the design of previous work (Chen et al. 2025; Ma et al. 2023; Chang et al. 2025), this system comprises a *Rewriter*, responsible for generating retrieval queries tailored for search engines based on the original questions; a *Reranker*, which selects relevant items from a large pool of retrieval results to aid in answering the original questions; and an *Answerer*, which produces the final answers by integrating the original questions with the filtered and reranked retrieval results. This system can invoke external retrieval tools to address complex queries by leveraging supplementary information.

Method: MHGPO

In this section, we introduce Multi-Agent Heterogeneous Group Policy Optimization (MHGPO), the first GOA specifically designed for MASs, utilizing relative advantage estimation over *heterogeneous* agent groups. Our method comprises a general policy optimization framework alongside three group-based rollout sampling strategies. Although our focus is on MASSs, since the proposed approach *applies to all MASs*, we use the terms MAS and MASS interchangeably throughout this paper.

Multi-Agent Policy Optimization Framework

Following the design of Multi-Agent Proximal Policy Optimization (MAPPO) (Yu et al. 2022), MHGPO adopts a *parameter-sharing* mechanism, leveraging the strong multitask capabilities of LLMs to instantiate all agents using a single LLM backend. This strategy significantly simplifies the training pipeline and reduces resource overhead. For a MAS system comprising n agents, we denote the k-th agent as A_k. Under the parameter-sharing scheme, a single LLM is optimized to perform all agent roles. The overall RL framework is illustrated in Figure 2 and described in detail below:

1. Multi-Agent Group Rollout Sampling. Given a single question $q \sim D$ sampled from the RL dataset, the MAS performs *group rollout sampling* by invoking its internal agents to collaboratively generate G final responses $\{o_i\}_{i=1}^G$. The sequence of intermediate steps from the input q to each final response constitutes a rollout *trajectory*. During sampling, the k-th agent produces a total of G_k input-output pairs, each corresponding to a distinct trajectory, denoted as $\{(q_{k,i}, o_{k,i}, m_{k,i})\}_{i=1}^{G_k}$. Here, $m_{k,i}$ denotes the *group identifier* associated with each rollout, as determined by a specific group rollout sampling algorithm. This identifier is subsequently used to aggregate rollouts into coherent groups. The resulting input-output pairs are collected as *agent rollouts* and serve as training data for model optimization.

2. Backward Propagation of Rewards. After the MAS generates the final responses $\{o_i\}_{i=1}^G$ for a query q, a reward model or predefined reward rule assigns reward signals to these outputs, resulting in a shared reward set $\{R_i^{\text{shared}}\}_{i=1}^G$. Starting from the endpoints of the sampling trajectories, these shared rewards are then *propagated backward* through each trajectory to reach the preceding agents A_k, where they are *aggregated* for each agent's output. Specifically, for the i-th output $o_{k,i}$ produced by agent A_k, the corresponding shared reward is computed as:

$$R_{k,i}^{\text{shared}} = \text{Aggr}(\{R_{j,r}^{\text{shared}}\}_{j>k}) \tag{4}$$

where A_j denotes a *direct successor* of A_k along the trajectory—that is, an agent whose input directly depends on the output of A_k, without any intermediate agents. Each $o_{j,r}$ is a response generated by A_j based on inputs that include $o_{k,i}$. The function $\text{Aggr}(\cdot)$ denotes an aggregation operator, which is instantiated as simple averaging in our implementation.

After the shared rewards have been propagated, the final reward for each output $\{o_{k,i}\}_{i=1}^{G_k}$ is calculated by incorporating the agent-specific reward function $R_k^{\text{spe}}(\cdot)$, which typically imposes a penalty based on the output format of the agent. The resulting reward is given by

$$R_{k,i} = R_{k,i}^{\text{shared}} + R_k^{\text{spe}}(q_{k,i}, o_{k,i}).$$

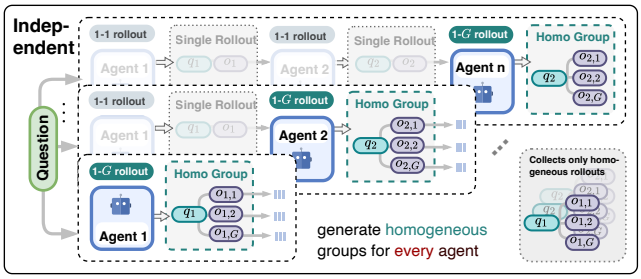

Figure 3: The proposed homogeneous-only group rollout sampling stratgy, namely *Independent Sampling (IS)*.

3. Heterogeneous Group Advantage Estimation Following the idea of GRPO, we adopt group-wise relative rewards for advantage estimation, instead of relying on a critic model. For a single input to the entire MAS, the advantage of the i-th output from agent A_k is computed as:

$$\hat{A}_{k,i} = \frac{R_{k,i} - \text{mean}(\{R_{l,j} \mid m_{l,j} = m_{k,i}\})}{\text{std}(\{R_{l,j} \mid m_{l,j} = m_{k,i}\})} \tag{5}$$

That is, the relative advantage is calculated over the group composed of all samples that share the same *group identifier* with this rollout sample. It is crucial to note that, unlike the GRPO in the single-agent setting, the rollouts within a group can be generated from different input prompts, thus forming *heterogeneous groups*. In the implementation, rewards are only assigned to the last token of each generated sequence. Specifically,

$$R^t_{k,i} = \begin{cases} 0, & \text{if } t < T, \\ R_{k,i}, & \text{if } t = T, \end{cases}$$

where T denotes the length of the output sequence. Based on this reward assignment, the token-wise relative advantage $\hat{A}^t_{k,i}$ is computed accordingly.

4. Multi-Agent Optimization Under the parameter-sharing mechanism, the RL optimization objective for the MAS retains the same form as GRPO, but the losses from all agents must be aggregated to facilitate collaborative optimization.

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{ref}}}(\cdot|q)}[\frac{1}{n}\sum_{k=1}^{n}\frac{1}{G_k}\sum_{i=1}^{G_k}\frac{1}{|o_{k,i}|}$$
$$\sum_{t=1}^{|o_{k,i}|}\min(r^t_{k,i}\hat{A}^t_{k,i}, \text{clip}(r^t_{k,i}, 1-\epsilon, 1+\epsilon)\hat{A}^t_{k,i})$$
$$- \beta D_{KL}(\pi_\theta||\pi_{\text{ref}})], \tag{6}$$

$$r^t_{k,i} = \frac{\pi_\theta(o^t_{k,i}|q, o^{<t}_{k,i})}{\pi_{\theta_{\text{ref}}}(o^t_{k,i}|q, o^{<t}_{k,i})}. \tag{7}$$

With the above objective, coordinated optimization of the multi-agent system can be performed based on all internal input-output pairs generated during the MAS rollout phase.

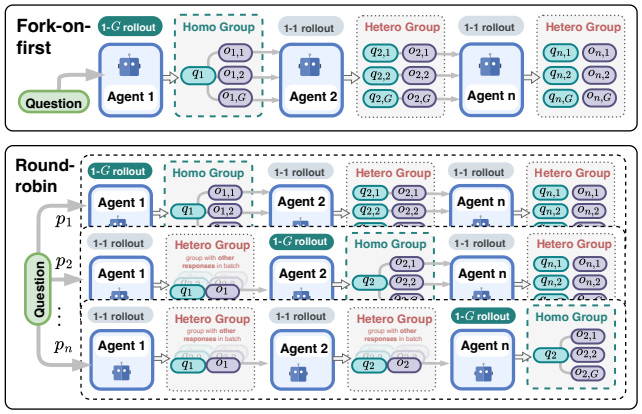

Figure 4: The proposed two *heterogeneous group* rollout sampling strategies, namely *Fork-on-first (FoF)* and *Round-robin (RR)*.

Multi-Agent Group Rollout Sampling

Within the proposed multi-agent optimization framework, a key challenge lies in constructing the candidate group G'_k for each agent, specifically in executing the *group rollout* procedure to obtain G_k and subsequently *forming completion groups* from the sampled trajectories. In this section, we introduce three distinct strategies implemented in MHGPO, each utilizing different mechanisms for group rollout sampling and group formation. These are illustrated in Figures 3 and 4, with detailed algorithms provided in the Appendix.

Independent Sampling (IS) We begin by presenting a baseline strategy for group rollout sampling: IS, as illustrated in Figure 3. As established in the relative advantage computation of GRPO (Equation 3), all samples within a group originate from the same input, forming what we define as a *homogeneous group*. Evaluating relative advantages within such homogeneous groups enables reliable assessment of the LLM's response quality for specific tasks.

Adhering to this principle, IS proceeds as follows: for each question $q \sim D$ in the training dataset, we sequentially activate the MAS until a specific Agent is selected. At this point, a 1-G group rollout is executed. The rewards associated with these rollouts are then computed using the reward backpropagation mechanism introduced in MHGPO. This procedure is repeated for each Agent, and only the resulting 1-G homogeneous rollouts are preserved.

In total, $n \times G$ rollout samples are collected, corresponding to n *independent* homogeneous groups of size G. These groups are subsequently employed to fine-tune the backbone LLM, promoting balanced performance across all Agent-specific tasks.

Fork-on-first (FoF) It is evident that the IS strategy suffers from several limitations. On one hand, it introduces significant rollout redundancy and computational waste; on the other hand, it treats each agent in isolation, neglecting inter-agent dependencies and collaborative dynamics. To overcome these shortcomings, we propose the FoF strategy—a simple yet effective alternative that eliminates redun-

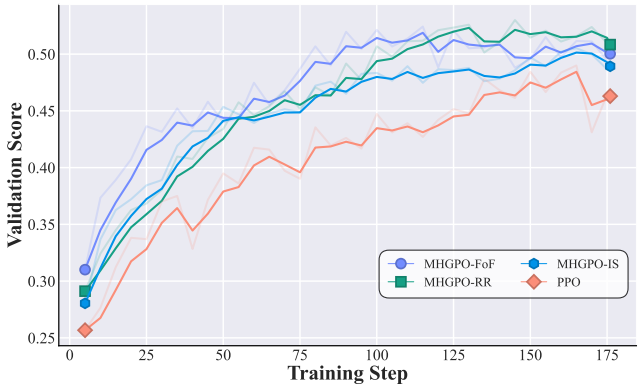

Figure 5: Validation scores (F1-score) during one epoch of training on the HotpotQA training set using different RL algorithms. Validation is performed every 5 steps (with a batch size of 512 for each step) on a validation set consisting of the first 500 samples from the test set.

dant rollouts while explicitly modeling interactions among agents, as shown in Figure 4(upper). Most importantly, FoF enables advantage estimation over *heterogeneous groups* for downstream agents.

Concretely, in FoF, for each input question, G rollouts are performed *only at the entry agent* of the MAS, while all subsequent agents perform one-to-one rollouts. As a result, the entry agent produces G rollouts with the same input but different outputs (forming a homogeneous group), whereas each subsequent agent processes G distinct trajectories with varying inputs and outputs (forming heterogeneous groups), and computes relative advantages accordingly. In this manner, the MAS produces G reward signals at its output layer, with the total number of rollout operations across the system amounting to nG.

Round-robin (RR) Preliminary experiments with FoF validate the effectiveness of computing relative advantages within heterogeneous groups. This inspires us to explore the possibility of computing advantages using both heterogeneous and homogeneous groups at each agent. In FoF, only the entry agent forms homogeneous groups, as the "fork point" is fixed at the first agent. What if we set the fork point at each agent, allowing every agent to compute advantages within homogeneous groups?

Based on this idea, we introduce the RR strategy, where for each input sample, the i-th agent is selected as the fork point with probability p_i. Suppose the fork point for a particular sample is at A_i; then the advantage computation for A_j, where $j \geq i$, follows the same approach as in FoF, while A_j for $j < i$ only has a single rollout. In this case, the rollout of A_j must be grouped with $\lfloor p_i \times \text{batch_size} \rfloor - 1$ other rollouts at A_j in the same batch whose fork points are also at i to compute relative advantages. In this way, each agent has a probability p_i of generating a homogeneous rollout group, and only the $p_1 \times$ batch_size samples incur the same rollout cost as FoF, while the rollout cost for the remaining samples is reduced.

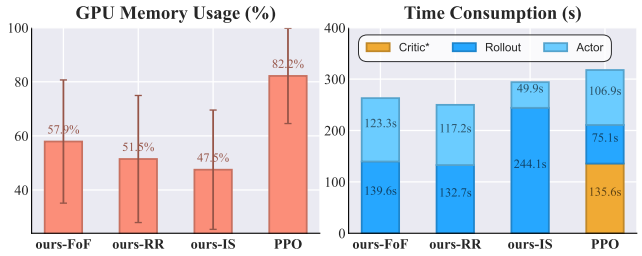

Figure 6: GPU memory usage (%) and average time per training step (in seconds) during training with different MARL algorithms.

Experiments & Discussion

Experiment Setup

Implementation Details We design the prompts for each agent following the approach of Chen et al. (2025). In accordance with Su et al. (2024), we use the Wikipedia dump as the retrieval corpus and deploy a `contriever` (Izacard et al. 2022) service to serve as the underlying search engine for the *retriever* module. Consistent with standard reinforcement learning practices on multi-hop datasets, we perform online training for only one epoch (i.e., both `epoch` and `ppo_epoch` are set to 1). For MAPPO, the critic network adopts a backbone architecture identical in size to that of the actor. In MHGPO, we set the group size to $G = 4$ for all group rollouts by default. For MHGPO-RR, the round-robin probabilities for each agent are set to $(0.7, 0.1, 0.2)$. Experiments were conducted using 8 NVIDIA H100 80GB GPUs. Key parameters used in the experiments are shown in Table 2.

Datasets and Models We adopt commonly used evaluation datasets in search systems, including the multi-hop datasets HotpotQA (Yang et al. 2018), 2WikiMultiHopQA (Ho et al. 2020), and MuSiQue (Trivedi et al. 2022), for training and evaluation. Llama3.1-Instruct-8B was selected as the backbone implementation for the MAS.

Evaluation Metrics Since the selected datasets all contain gold-standard answers, we use the Exact Match (EM) score, which evaluates the exact match between the system's final answer and the reference answer, as well as the F1-score and Accuracy (which measures the overlap between the normalized prediction and the gold answer) as the three primary evaluation metrics.

Main Results

To evaluate the effectiveness of the proposed MHGPO and various group rollout sampling strategies, we use a three-agent search system as an example and conduct training without any SFT warm-up. A batch size of 512 is used to train the model for one epoch (a total of 176 steps) on the training set of HotpotQA.

Question 1. *What are the advantages of the training process of MHGPO compared to that of PPO?*

Table 1: Test performance of different MARL algorithms and unoptimized baselines on mainstream QA datasets, measured by Accuracy (%), Exact Match (%), and F1-Score (%). All MARL models are trained via one epoch of RL on the HotpotQA training set using the same LLM checkpoint.

Methods/Models		HotpotQA			2WikiMultihooQA			MuSiQue		
		Acc	EM	F1	Acc	EM	F1	Acc	EM	F1
Single LLMs	Llama3.1-8B	21.269	14.018	22.782	16.963	7.069	20.817	5.492	1.365	2.813
	Qwen2.5-72B	30.304	25.564	36.183	29.373	27.234	32.038	7.737	5.337	15.411
Single LLM +RL	Llama3.1-8B	27.643	0.500	27.421	14.019	0.676	11.025	4.096	2.400	9.287
LLM + MAS	Llama3.1-8B	20.800	14.000	21.452	22.281	15.872	22.427	4.220	2.524	8.044
	Qwen2.5-72B	32.519	24.254	34.631	36.379	18.511	25.630	7.075	4.717	11.959
LLM +MAS +MARL (Llama3.1-8B-Inst)	MAPPO	38.217	34.450	46.400	35.790	31.767	38.722	11.626	9.350	19.738
	MHGPO-IS	37.677	33.707	45.582	34.796	31.115	38.050	9.888	7.944	18.421
	MHGPO-FoF	40.459	36.570	49.429	36.299	31.750	39.089	**12.702**	**10.012**	**21.633**
	MHGPO-RR	**40.864**	**37.043**	**49.724**	**36.840**	**31.838**	**39.388**	11.378	9.350	21.144

Table 2: Key hyperparameters for the implementation of MHGPO and MAPPO.

Hyperparameter	**Explanation**	**Value(s)**
`lr`	learning rate	(5e-7, 1e-6)
`rollout.n`	group size for GOAs	4
`batch_size`	batch size	512
ϵ	clip range	0.2
`top_n`	param for top-n sampling	0.9
β	param for KL penalty	0.001

We first evaluate the performance of MAS on the validation set (comprising the first 500 samples from the HotpotQA test set) during MARL training. Validation is conducted every five steps, and the F1-score curves of different algorithms are shown in Figure 5. It can be observed that the MHGPO variants converge more quickly than MAPPO, with minor differences among the variants. Specifically, the variant employing IS achieves the fastest convergence rate but exhibits the lowest performance upper bound. This is likely because the independent rollout optimization for each agent within a homogeneous group facilitates rapid individual learning but fails to capture inter-agent dependencies, thus limiting the ability to discover a *globally* optimal policy. In contrast, the FoF strategy introduces a heterogeneous group structure, which, although slower to converge, enables end-to-end optimization. The RR variant strikes a balance between the two: it allows for fine-grained optimization across agents during early training while still achieving a high performance ceiling.

In terms of training overhead and efficiency, Figure 6 presents the experimental results conducted within the verl framework (Sheng et al. 2025), utilizing a machine equipped with eight NVIDIA H100 GPUs (80GB HBM3) and an Intel Xeon Platinum 8468 processor. We report the average GPU memory usage and training time over the first 50 steps of MARL training across different algorithms. As illustrated, MHGPO (e.g., FoF) achieves substantially lower memory consumption than MAPPO, primarily due to the removal of Critic models. Furthermore, because it avoids both value estimation and Critic updates, the overall training time—despite marginally longer rollout durations caused by GOAs—remains lower than that of PPO, particularly when the rollout group size G is relatively small (set to 4 in our experiments). Notably, RR, which incorporates relative advantages across batches, further reduces memory usage compared to FoF.

Question 2. *How does MHGPO perform on the test set compared to the baselines? Can MHGPO generalize to other datasets?*

After MARL training, the resulting model is used as the backbone of the MAS and evaluated on various QA datasets that require retrieval to support answer generation. The performance of the MAS optimized by different RL algorithms, along with baselines including standalone LLMs and unoptimized MASs, is summarized in Table 1. For comparison, we also present the performance of standalone LLMs directly applied to the question answering task, LLM models optimized with RL on HotpotQA, and MAS without reinforcement learning.

As shown in the table, standalone LLMs generally exhibit poor performance, and their integration into the MAS framework without optimization often leads to further degradation—primarily due to instruction-following failures, which are especially severe for smaller models such as LLaMA3.1-8B. However, the incorporation of MARL substantially boosts system performance, achieving more than a twofold improvement over the unoptimized system.

Among the various MARL algorithms, the FoF and RR variants of MHGPO demonstrate significant performance gains over MAPPO on the in-domain HotpotQA test set. Notably, RR achieves marginally superior results compared

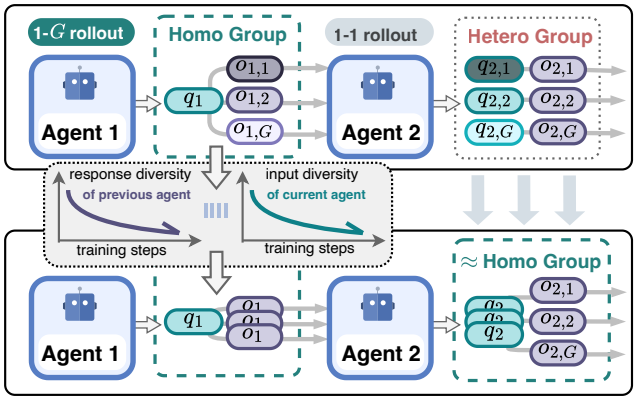

Figure 7: Illustration of how the heterogeneous groups produced by rollout sampling in MAS gradually transition into homogeneous groups during the optimization process of MHGPO.

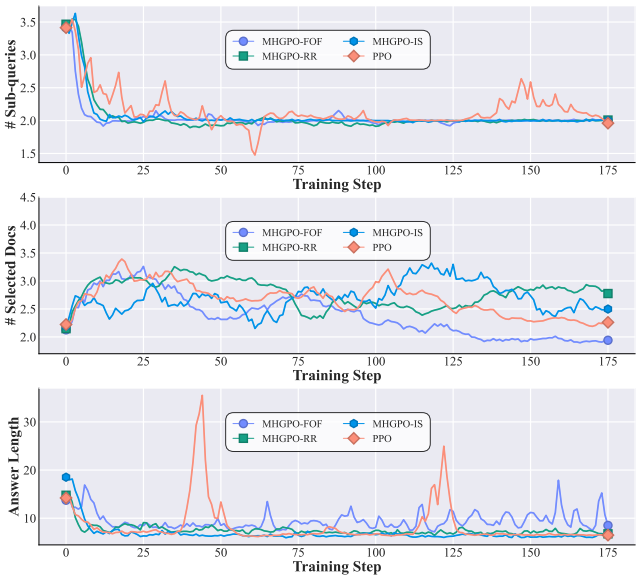

Figure 8: Output pattern changes of each agent during training with different MARL algorithms. From top to bottom: outputs of the rewriter, reranker, and answerer agents.

to FoF, while incurring the lowest computational overhead. Moreover, on out-of-domain datasets such as 2WikiMultihopQA and MuSiQue, all MARL algorithms consistently outperform the unoptimized baseline, underscoring the strong generalizability of end-to-end multi-agent optimization. Across the majority of datasets, MHGPO sustains a clear and consistent advantage over MAPPO.

Observation & Discussion

Question 3. *How do MARL algorithms optimize each agent to find the most effective policies?*

To understand how end-to-end multi-agent optimization influences the behavior of individual agents, we analyze the evolution of output patterns for each agent throughout training with different algorithms, as shown in Figure 8. We observe that during MARL, the rewriter learns to generate two rewritten queries as the optimal strategy—consistent with the two-hop structure of HotpotQA. The reranker stabilizes to selecting 2–3 documents, while the final answer length converges to fewer than 10 tokens.

When comparing different MARL algorithms, we first observe that MAPPO exhibits instability across agents, likely due to its critic needing to simultaneously model heterogeneous tasks for all three agents. In contrast, the variants of MHGPO demonstrate significantly improved stability. Specifically, FoF converges rapidly on the first agent but displays higher variance on the *Answerer*. Meanwhile, both RR and IS show smoother and more consistent convergence behaviors across all agents.

Question 4. *Why does heterogeneous-group-based policy optimization in MHGPO work?*

We argue that the optimization strategy employed in MHGPO—computing relative advantages over *heterogeneous* groups—is effective in practice due to two key factors: (1) parameter sharing combined with reward backpropagation, and (2) a natural reduction in intra-group diversity over the course of training.

First, since all agents in MHGPO are instantiated by a shared LLM, a sampled trajectory from the MAS can be approximately interpreted as a *single long-chain reasoning process* executed by the LLM in response to an input question. Each step in this reasoning chain receives a process-level reward signal via reward backpropagation, even though policy optimization is still performed locally at each agent without access to full historical context.

Second, taking FoF as an example, we observe that *heterogeneous groups naturally evolve into homogeneous ones* as training progresses, as illustrated in Figure 7. This effect arises from the nature of rejection sampling in RL: over time, the diversity of outputs generated through top-n sampling diminishes. We empirically validate this phenomenon in our MAS setting. As shown in Figure 9, the similarity among group rollout samples—measured by F1 score—for each agent steadily increases throughout training.

For the *Rewriter*, the output similarity reaches approximately 0.7 by steps 25–30, suggesting that the *Reranker* is likely receiving nearly identical sets of retrieved documents. At this point, the *Reranker* effectively begins optimizing over a de facto homogeneous group, which plausibly accounts for the marked performance improvements observed beginning at step 25.

Related Works

LLM-based Multi-agent Systems

Compared to a standalone LLM, an MAS comprising multiple agents—each potentially instantiated with distinct LLMs—demonstrates superior competence in addressing complex tasks. Recent multi-agent development frameworks such as Microsoft AutoGen (Wu et al. 2023) and OWL (Hu

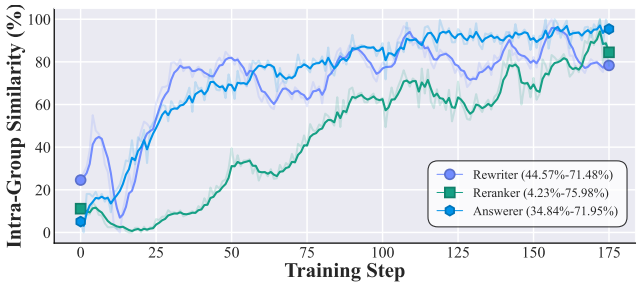

Figure 9: The intra-group similarity for each agent's rollout, measured by the average pairwise F1 score between generated responses. For consistency and ease of comparison across agents, similarity scores are normalized to the range [0, 1], with the corresponding minimum and maximum values indicated in the legend.

et al. 2025) further underscore the practical promise of MASs in real-world applications. In the representative task of search-based question answering, numerous studies have highlighted the benefits of multi-agent collaboration. For example, Chang et al. (2025) model retrieval-augmented generation (RAG) as a sequential MAS, leveraging coordinated LLM interactions to improve retrieval-augmented generation. Wu et al. (2025) propose a routing mechanism that dynamically delegates sub-tasks to specialized expert agents, enhancing overall reasoning capabilities. Additionally, Zhao et al. (2025) integrate a backtracking mechanism into the MAS, enabling reversible multi-hop reasoning and thereby strengthening the performance of RAG-based QA systems.

End-to-end Reinforcement Learning for MASs

Despite the effectiveness of MASs in tackling complex tasks, their construction and deployment remain nontrivial. Cemri et al. (2025) highlight that multi-agent systems are vulnerable to various sources of failure, including brittle system designs and inconsistencies in LLM formatting behaviors. These challenges underscore the importance of applying end-to-end reinforcement learning to MASs, i.e., MARL, to achieve robust and adaptive coordination. The dominant paradigm in MARL is centralized training with decentralized execution (CTDE), exemplified by methods such as MAPPO (Yu et al. 2022) and HAPPO (Kuba et al. 2022), which utilize *parameter sharing* to enable efficient multi-agent training. Recent studies have begun to explore the integration of reinforcement learning within MASs. For instance, Kulkarni et al. (2024) and Ma et al. (2023) investigate component-level optimization using RL techniques. Notably, Chen et al. (2025) apply MAPPO to LLM-based multi-agent RAG, implementing all components with a shared LLM backbone. More recently, Liao et al. (2025b) propose a reinforcement learning framework specifically designed for MASs, enabling end-to-end optimization across interacting agents.

Conclusions

In this paper, we propose MHGPO, a novel reinforcement learning framework for end-to-end optimization of multi-agent LLM systems. By eliminating the critic network and leveraging relative group advantages through a two-phase sampling-propagation strategy, MHGPO addresses the instability and computational overhead commonly associated with traditional MAPPO in MASs. Within MHGPO, we explore three group rollout sampling strategies and demonstrate that both FoF and RR—which compute relative advantages based on heterogeneous groups—achieve competitive performance, with RR offering additional benefits in computational efficiency. Experiments on multi-hop QA tasks show that MHGPO consistently outperforms baselines in terms of both accuracy and training efficiency. Although our detailed analysis and verification were solely carried out on MASS, the proposed method is capable of being generalized to general MAS. Experiments on MAS within more scenarios will be reserved for future work. It is our hope that our findings will stimulate further research on reinforcement learning in multi-agent LLM systems.

References

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D. M.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Cemri, M.; Pan, M. Z.; Yang, S.; Agrawal, L. A.; Chopra, B.; Tiwari, R.; Keutzer, K.; Parameswaran, A. G.; Klein, D.; Ramchandran, K.; Zaharia, M.; Gonzalez, J. E.; and Stoica, I. 2025. Why Do Multi-Agent LLM Systems Fail? *CoRR*, abs/2503.13657.

Chang, C.; Jiang, Z.; Rakesh, V.; Pan, M.; Yeh, C. M.; Wang, G.; Hu, M.; Xu, Z.; Zheng, Y.; Das, M.; and Zou, N. 2025. MAIN-RAG: Multi-Agent Filtering Retrieval-Augmented Generation. *CoRR*, abs/2501.00332.

Chen, Y.; Yan, L.; Sun, W.; Ma, X.; Zhang, Y.; Wang, S.; Yin, D.; Yang, Y.; and Mao, J. 2025. Improving Retrieval-Augmented Generation through Multi-Agent Reinforcement Learning. *CoRR*, abs/2501.15228.

Gao, Y.; Xiong, Y.; Gao, X.; Jia, K.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, M.; and Wang, H. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997.

Ho, X.; Nguyen, A. D.; Sugawara, S.; and Aizawa, A. 2020. Constructing A Multi-hop QA Dataset for Comprehensive Evaluation of Reasoning Steps. In Scott, D.; Bel, N.; and Zong, C., eds., *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020,*

Barcelona, Spain (Online), December 8-13, 2020, 6609–6625. International Committee on Computational Linguistics.

Hu, M.; Zhou, Y.; Wendong Fan, Y. N.; Xia, B.; Sun, T.; Ye, Z.; Jin, Z.; Li, Y.; Zhang, Z.; Wang, Y.; Ye, Q.; Luo, P.; and Li, G. 2025. OWL: Optimized Workforce Learning for General Multi-Agent Assistance in Real-World Task Automation.

Izacard, G.; Caron, M.; Hosseini, L.; Riedel, S.; Bojanowski, P.; Joulin, A.; and Grave, E. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Trans. Mach. Learn. Res.*, 2022.

Kuba, J. G.; Chen, R.; Wen, M.; Wen, Y.; Sun, F.; Wang, J.; and Yang, Y. 2022. Trust Region Policy Optimisation in Multi-Agent Reinforcement Learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Kulkarni, M.; Tangarajan, P.; Kim, K.; and Trivedi, A. 2024. Reinforcement Learning for Optimizing RAG for Domain Chatbots. *CoRR*, abs/2401.06800.

Liao, J.; Wen, M.; Wang, J.; and Zhang, W. 2025a. MARFT: Multi-Agent Reinforcement Fine-Tuning. arXiv:2504.16129.

Liao, J.; Wen, M.; Wang, J.; and Zhang, W. 2025b. MARFT: Multi-Agent Reinforcement Fine-Tuning. *CoRR*, abs/2504.16129.

Ma, X.; Gong, Y.; He, P.; Zhao, H.; and Duan, N. 2023. Query Rewriting for Retrieval-Augmented Large Language Models. *CoRR*, abs/2305.14283.

OpenAI. 2023. GPT-4 Technical Report. *CoRR*, abs/2303.08774.

Radford, A.; Narasimhan, K.; Salimans, T.; and Sutskever, I. 2018. Improving language understanding by generative pre-training. *OpenAI Blog*.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M. I.; and Abbeel, P. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In Bengio, Y.; and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347.

Shao, Z.; Wang, P.; Zhu, Q.; Xu, R.; Song, J.; Zhang, M.; Li, Y. K.; Wu, Y.; and Guo, D. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *CoRR*, abs/2402.03300.

Sheng, G.; Zhang, C.; Ye, Z.; Wu, X.; Zhang, W.; Zhang, R.; Peng, Y.; Lin, H.; and Wu, C. 2025. HybridFlow: A Flexible and Efficient RLHF Framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, 1279–1297. ACM.

Su, W.; Tang, Y.; Ai, Q.; Wu, Z.; and Liu, Y. 2024. DRAGIN: Dynamic Retrieval Augmented Generation based on the Real-time Information Needs of Large Language Models. In Ku, L.; Martins, A.; and Srikumar, V., eds., *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, 12991–13013. Association for Computational Linguistics.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023a. LLaMA: Open and Efficient Foundation Language Models. *CoRR*, abs/2302.13971.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Trivedi, H.; Balasubramanian, N.; Khot, T.; and Sabharwal, A. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. *Trans. Assoc. Comput. Linguistics*, 10: 539–554.

Wu, F.; Li, Z.; Wei, F.; Li, Y.; Ding, B.; and Gao, J. 2025. Talk to Right Specialists: Routing and Planning in Multi-agent System for Question Answering. *CoRR*, abs/2501.07813.

Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Zhang, S.; Zhu, E.; Li, B.; Jiang, L.; Zhang, X.; and Wang, C. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. *CoRR*, abs/2308.08155.

Yang, Z.; Qi, P.; Zhang, S.; Bengio, Y.; Cohen, W. W.; Salakhutdinov, R.; and Manning, C. D. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In Riloff, E.; Chiang, D.; Hockenmaier, J.; and Tsujii, J., eds., *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 2369–2380. Association for Computational Linguistics.

Yu, C.; Velu, A.; Vinitsky, E.; Gao, J.; Wang, Y.; Bayen, A. M.; and Wu, Y. 2022. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Yu, Q.; Zhang, Z.; Zhu, R.; Yuan, Y.; Zuo, X.; Yue, Y.; Fan, T.; Liu, G.; Liu, L.; Liu, X.; Lin, H.; Lin, Z.; Ma, B.; Sheng, G.; Tong, Y.; Zhang, C.; Zhang, M.; Zhang, W.; Zhu, H.; Zhu, J.; Chen, J.; Chen, J.; Wang, C.; Yu, H.; Dai, W.; Song, Y.; Wei, X.; Zhou, H.; Liu, J.; Ma, W.; Zhang, Y.; Yan, L.; Qiao, M.; Wu, Y.; and Wang, M. 2025. DAPO: An Open-Source LLM Reinforcement Learning System at Scale. *CoRR*, abs/2503.14476.

Zhao, X.; Gao, F.; Yang, R.; Chen, Y.; Wang, Y.; Zhu, Y.; Tang, J.; and Li, I. 2025. ReAgent: Reversible Multi-Agent Reasoning for Knowledge-Enhanced Multi-Hop QA. *CoRR*, abs/2503.06951.

Detailed Algorithm

Multi-Agent Policy Optimization

Algorithm 1: Multi-Agent Optimization in MHGPO

Input: Training dataset $\mathcal{D}$ with question-answer pairs (q, a_{golden}), reward function $\mathcal{R}$, `batch_size`, `total_epochs`, `ppo_epochs`, rollout sampling function $\mathcal{H}$
Output: Optimized actor parameters θ
Initialize actor parameters θ (shared among all agents), reference parameters θ_{ref}
for *epoch* $= 1$ **to** `total_epochs` **do**
  for *each batch* $\mathcal{B} \subset \mathcal{D}$ **do**
    for *each question* q *in batch* $\mathcal{B}$ **do**
      `// Multi-Agent rollout trajectory sampling`
      Generate G rollout trajectories $\{\mathcal{T}_i\}_{i=1}^{G} \leftarrow \mathcal{H}(\theta, q)$
      Collect rollout pairs $\{(q_{k,j}, o_{k,j}, m_{k,j})\}$ for all agents across all trajectories
      `// Final reward calculation`
      for *each trajectory* $\mathcal{T}_i$ **do**
        Let o_i be the final output of $\mathcal{T}_i$
        Compute final reward: $R_i^{\text{shared}} = \mathcal{R}(o_i, a_{\text{golden}})$
      `// Backward propagation of rewards`
      for *each rollout pair* $(q_{k,j}, o_{k,j})$ **do**
        Calculate aggregated reward $R_{k,j}^{\text{shared}}$ according to Equation 4
        Apply agent-specific reward $R_{k,j}^{\text{spe}}$ to obtain $R_{k,j}$
    `// Actor update`
    for *ppo_epoch* $= 1$ **to** `ppo_epochs` **do**
      for *each rollout pair* $(q_{k,j}, o_{k,j}, m_{k,j}, R_{k,j})$ **do**
        Form rollout groups based on $m_{k,j}$ and estimate the advantage using Equation 5
      Update policy parameters θ using the policy gradient method with Equation 6 as the target

The complete optimization procedure of the proposed MHGPO framework is outlined in Algorithm 1. Central to this process is the sampling function $\mathcal{H}(\theta)$, which can be instantiated using one of the IS, FoF, or RR strategies. This function is pivotal for trajectory sampling within the MAS, as it determines both the sampling behavior and the group assignment for each rollout. In the illustrative MAS scenario considered in this work, the reward function $\mathcal{R}$ is instantiated as the F1 score.

Rollout Sampling Strategies

To introduce the proposed rollout sampling algorithms, we begin by presenting a core utility function, *fork_on()*, defined in Algorithm 2. This function facilitates group rollout by conducting one-to-one trajectory sampling before and after the *first interaction with Agent* i, and performing a one-to-many branching exclusively at Agent i, designated as the fork point. Concretely, the function generates G parallel trajectories starting from the fork point and assigns group identifiers based on the index of the forking agent. The output is a complete set of group-specific forked trajectories.

Algorithm 2: Base Rollout Sampling Function *fork_on()*

Input: Question q, rollout group size G, fork agent A_i, shared actor parameters θ
Output: Set of rollout trajectories $\{\mathcal{T}_i\}_{i=1}^{G}$
Initialize $o \leftarrow q$, empty prefix trajectory $\mathcal{T}_{\text{prefix}} \leftarrow \emptyset$
`// One-to-one rollout until fork agent` A_i `is encountered`
while $MAS.next_agent(o) \neq A_i$ **do**
  $A_{c_j} \leftarrow MAS.next_agent(o)$
  $q_{c_j} \leftarrow MAS.process_prompt(o, A_{c_j})$
  $o_{c_j} \leftarrow A_{c_j}(q_{c_j}; \theta)$
  Append (q_{c_j}, o_{c_j}) to $\mathcal{T}_{\text{prefix}}$
  $o \leftarrow o_{c_j}$
`// Fork-on-`A_i`: one-to-many rollout from agent` A_i
$q_{c_i} \leftarrow MAS.process_prompt(o, A_i)$
for $i = 1$ **to** G **do**
  $o_{c_i,i} \leftarrow A_i(q_{c_i}; \theta)$
  Initialize $\mathcal{T}_i \leftarrow \mathcal{T}_{\text{prefix}} \cup \{(q_{c_i}, o_{c_i,i})\}$
  $o \leftarrow o_{c_i,i}$
  `// Continue one-to-one rollout for each fork`
  $j \leftarrow 1$
  while $MAS.has_next_agent(o)$ **do**
    $A_{c_{i+j}} \leftarrow MAS.next_agent(o)$
    $q_{c_{i+j}} \leftarrow MAS.process_prompt(o, A_{c_{i+j}})$
    $o_{c_{i+j}} \leftarrow A_{c_{i+j}}(q_{c_{i+j}}; \theta)$
    Append $(q_{c_{i+j}}, o_{c_{i+j}})$ to $\mathcal{T}_i$
    $o \leftarrow o_{c_{i+j}}$
    $j \leftarrow j + 1$
`// Group identifier assignment`
for $i = 1$ **to** G **do**
  for *each rollout pair* $(q_{i,*}, o_{i,*})$ *in trajectory* $\mathcal{T}_i$ **do**
    Assign group identifier $m_{i,*} \leftarrow i$
return $\{\mathcal{T}_i\}_{i=1}^{G}$

Fork-on-first (FoF) The FoF strategy leverages this utility by directly invoking *fork_on()* with the first agent in the trajectory designated as the fork point, as detailed in Algorithm 3.

Independent Sampling (IS) In contrast, IS executes an exhaustive search across all agents for each input q, invoking *fork_on()* at every potential fork point. After sampling, it filters out all *heterogeneous* rollout groups and retains only the homogeneous ones for downstream training.

Algorithm 3: Fork-on-First Rollout Sampling

Input: Question q, rollout group size G, shared actor parameters θ
Output: Set of rollout trajectories $\{\mathcal{T}_i\}_{i=1}^{G}$

```
// Determine the first agent to handle the question
```
$A_{c_0} \leftarrow MAS.next_agent(q)$
```
// Call fork_on to perform Fork-on-First rollout
```
$\{\mathcal{T}_i\}_{i=1}^{G} \leftarrow$ `fork_on`(q, G, A_{c_0}, θ)
return $\{\mathcal{T}_i\}_{i=1}^{G}$

Algorithm 4: Independent Sampling

Input: Single question q, rollout group size G, shared actor parameters θ
Output: Merged set of rollout pairs $\mathcal{P}$ collected from each individual agent
Initialize $\mathcal{P} \leftarrow \emptyset$
for *each agent* A_i *where* $i = 1$ *to* n **do**
  `// Fork on agent` A_i
  Extract rollout pairs $\{(q_{k,j}, o_{k,j}, m_{k,j})\}$ from `fork_on`(q, G, A_i, θ)
  `// Filter only rollout pairs where` $k = i$
  for *each rollout pair* $(q_{k,j}, o_{k,j}, m_{k,j})$ **do**
    if $k = i$ **then**
      Append $(q_{k,j}, o_{k,j}, m_{k,j})$ to $\mathcal{P}$
return final rollout pairs $\mathcal{P}$

Round-robin (RR) The RR method operates at the batch level. For each sample in the batch, it randomly selects an Agent i as the fork point according to a predefined probability distribution $\{p_i\}$. Upon processing the entire batch, rollout samples that belong to singleton groups (i.e., groups with no shared samples) are re-shuffled and regrouped prior to training, as specified in Algorithm 5.

Implementation Details

Training Dataset

We use the training set of HotpotQA (Yang et al. 2018), which contains 90,447 samples, as the training data for reinforcement learning. Only the questions from the dataset are used as model inputs, while the corresponding answers serve as golden labels for reward computation based on the F1 score. Representative examples are shown in Figure 3.

Agent Prompts

We follow the three-agent MAS design proposed by Chen et al. (2025), with only minor modifications to the agents' prompts and output formatting to simplify the system implementation.

Algorithm 5: Round-Robin Rollout Sampling

Input: Batch of questions $\mathcal{B} = \{q_1, q_2, \ldots, q_B\}$, rollout group size G, shared actor parameters θ, agent sampling probabilities $\{p_1, p_2, \ldots, p_n\}$
Output: Set of rollout trajectories $\{\mathcal{T}_i\}_{i=1}^{G'}$ with consistent group identifiers
Initialize rollout pair set $\mathcal{P}_{\text{all}} \leftarrow \emptyset$
for *each question* $q \in \mathcal{B}$ **do**
  `// Sample fork agent from categorical distribution`
  $A_i \sim \text{Categorical}(p_1, p_2, \ldots, p_n)$
  `// Fork-on-`A_i `to obtain` G `trajectories`
  $\{\mathcal{T}_j^{(q)}\}_{j=1}^{G} \leftarrow$ `fork_on`(q, G, A_i, θ)
  for *each trajectory* $\mathcal{T}_j^{(q)}$ **do**
    Extract all rollout pairs $(q_{k,j}, o_{k,j}, m_{k,j})$ from $\mathcal{T}_j^{(q)}$ and append to $\mathcal{P}_{\text{all}}$
`// Group reassignment for rollout pairs with group size` $= 1$
Count the number of occurrences for each group identifier m in $\mathcal{P}_{\text{all}}$
Let $\mathcal{P}_{\text{single}} \leftarrow$ rollout pairs whose original group m appears only once
Let $\mathcal{P}_{\text{valid}} \leftarrow$ rollout pairs whose group m appears more than once
Shuffle $\mathcal{P}_{\text{single}}$ randomly
Partition $\mathcal{P}_{\text{single}}$ into $N = \lfloor |\mathcal{P}_{\text{single}}|/G \rfloor$ disjoint groups $\{g_i\}_{i=1}^{N}$, where each $g_i = \{(q_{i,j}, o_{i,j})\}_{j=1}^{G}$
for *each group* g_i *where* $i = 1, \ldots, N$ **do**
  for *each rollout pair* $(q_{i,j}, o_{i,j})$ *where* $j = 1, \ldots, G$ **do**
    Assign new group identifier: $m_{i,j} \leftarrow i + |\mathcal{P}_{\text{valid}}|$
Merge $\mathcal{P}_{\text{valid}}$ with all reassigned g_i groups to form final $\mathcal{P}_{\text{all}}$
return final rollout pairs $\mathcal{P}_{\text{all}}$

Prompt for Rewriter

System: You are a professional assistant proficient in transforming complex or unclear questions into simpler, more searchable sub-questions.
Please assist me in rewriting or breaking down the provided questions into sub-questions to make it easier to search for answers using a search engine. The rewritten sub-questions must have logical connections and dependencies, and should not be overly repetitive in meaning.
Moreover, refrain from using vague demonstrative pronouns and similar expressions, and avoid duplicate sub-questions.
Output the sub-questions in the form of a string list, with the format:

```
### <query1>; <query2>; <query3>; ... ###
```

User: Original question is {`raw_question...`}

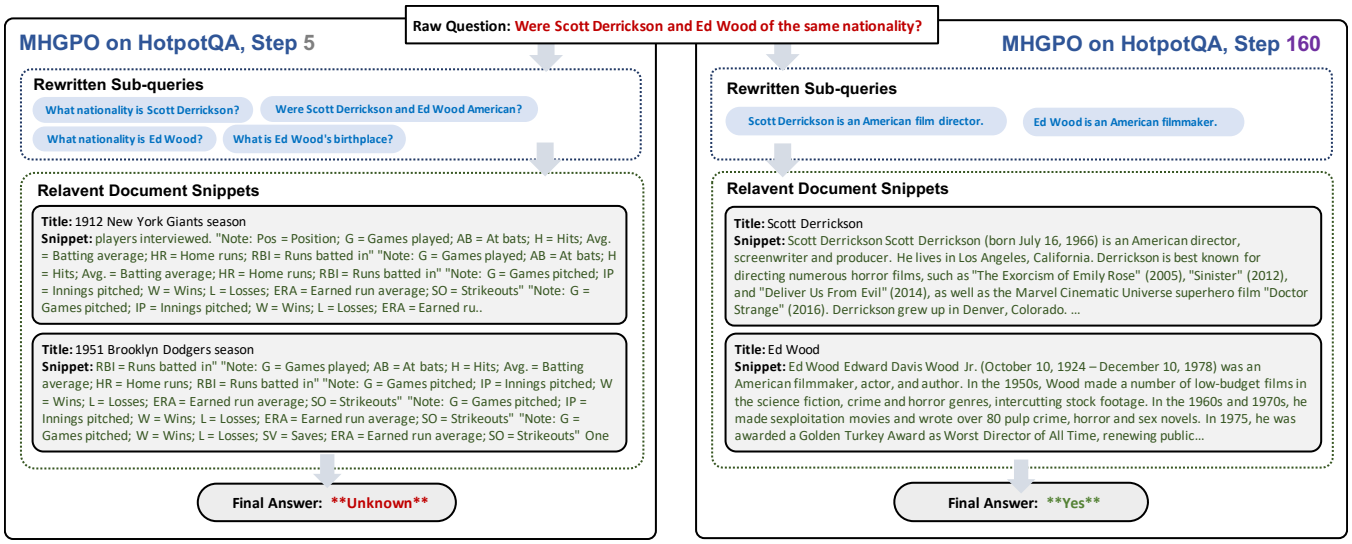

Figure 10: Case study: An example question is processed by the example MAS, with outputs from each stage shown for comparison between MHGPO-FoF after 5 steps and after 160 steps of training on HotpotQA.

Prompt for Reranker

System: You are a helpful, respectful, and honest assistant. Your task is to identify and output the IDs (such as 0, 1, 2, etc.) of the candidate Documents that are useful for answering the given Question.
Now, simply output the IDs of those candidate Documents that can assist in answering the Question: {`raw_question`}. Arrange them in the order where the more relevant documents are listed first.
User: Question is: {`raw_question`}
Document0: {`title:..., snippet:...`}
Document1: {`title:..., snippet:...`}
...
Do not output anything else.

Prompt for Answerer

System: You are a helpful, respectful and honest assistant. Your task is to predict the answer to the question based on the given documents. If you don't know the answer to a question, please don't share false information. Answer the question as accurately as possible.
User: Question is: {`raw_question`}
Document 0: {`title: ..., snippet: ...`}
Document 1: {`title: ..., snippet: ...`}
...
Now, answer the Question: {`raw_question`}, based on the above Documents.

Agent-specific Rewards

Regarding agent-specific reward design, we follow Chen et al. (2025): the *Rewriter* is penalized proportionally to the number of rewritten queries, receiving a penalty of -0.5 if more than four rewritten queries are generated. The *Reranker* receives penalties based on the format and redundancy of the selected documents, with a -0.5 penalty applied for duplicated entries or index out-of-bound errors. The *Answerer* is penalized according to the length of the generated answer, incurring a -1.0 penalty if the answer is excessively long. This reward structure enables fine-grained control over the behavior of individual agents during optimization.

Figure 11: Reward scores of different MARL algorithms on HotpotQA during one training epoch, recorded at every step with a batch size of 512.

Detailed Experiment Results

Training Process

In order to meticulously observe and compare the characteristics of different MARL algorithms during the training process, we have recorded the detailed metrics during the

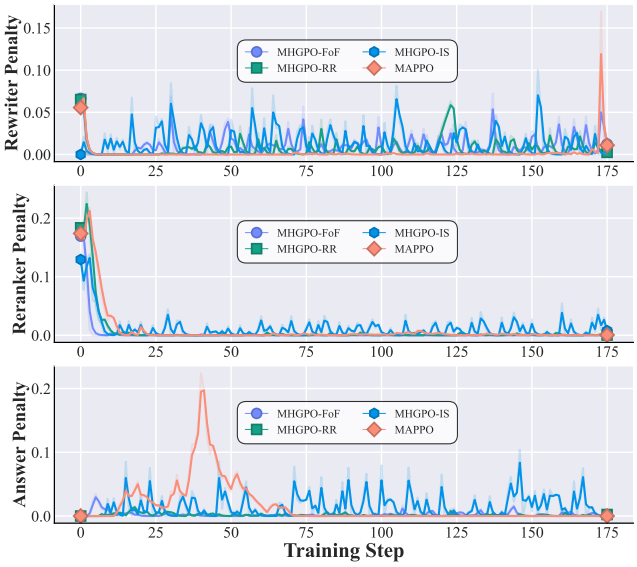

Figure 12: Agent-specific rewards (format penalty) for each agent in the three-agent MAS during one training epoch on HotpotQA using different MARL algorithms.

MARL training process on HotpotQA for the example MAS under consideration. Figure 11 records the reward score (with the penalties of each agent added) on the training set. It can be seen that compared to the rather unstable MAPPO, the other MHGPO variants all possess a more stable and rapid convergence process as well as an upper bound on the score. Compared to the validation score, the scores on the training set are generally 5% to 10% higher. Figure 12 separately records the changes in the format penalties obtained by different agents during the training process within the considered example 3-agent MAS. It can be observed that all methods can quickly find the correct output format and thus avoid penalties (the decrease within 10 steps reaches within 0.1). During this process, MAPPO once again demonstrates instability, with the possibility of the output format of agents collapsing midway. In addition, FoF and RR are hardly troubled by the format and consistently obtain relatively low penalties.

Case Study

To investigate the specific improvements brought by MARL to multi-hop question answering, we examine the performance of the proposed MAS system on a representative QA example, comparing its behavior in the early stage (step 5) and the later stage (step 160) of reinforcement learning, as illustrated in Figure 10.

At step 5, the system produces an incorrect answer (*unknown*) because the selected relevant snippets do not contain useful information, despite the seemingly reasonable sub-queries generated by the Rewriter. In contrast, by step 160, the MAS successfully answers the question. This improvement stems from the Reranker correctly identifying two relevant documents—biographical entries on Scott Derrickson and Ed Wood. Interestingly, the Rewriter now produces only two sub-queries, which, while less intuitive to humans, align better with the behavior of the `contriever` retriever engine, enabling the retrieval of the desired documents.

This example highlights how reinforcement learning can help agents like the Rewriter adapt to their environment (i.e., the retrieval engine), thereby overcoming performance bottlenecks.

Table 3: Examples of multi-hop questions and answers in HotpotQA.

Question	Answer
What is the relationship of Yeshahework Yilma's mother to the man who was Ethiopia's emperor from 1930 to 1974?	niece
What role on "Switched at Birth" was played by the actor who is being replaced by Daniel Hall on "The Young and the Restless"?	Tyler "Ty" Mendoza
Are the Sleepers located north or south of the Kancamagus Highway?	south