Multi-Agent Deep Research: Training Multi-Agent Systems with M-GRPO

Haoyang Hong[2*] Jiajun Yin[1] Yuan Wang[1] Jingnan Liu[1] Zhe Chen[1] Ailing Yu[1]
Ji Li[1] Zhiling Ye[1] Hansong Xiao[1] Yefei Chen[1] Hualei Zhou[1] Yun Yue[1]
Minghui Yang[1] Chunxiao Guo[1] Junwei Liu[1] Peng Wei[1] Jinjie Gu[1]

[1]Ant Group [2]Imperial College London

Code and Dataset: `https://github.com/AQ-MedAI/MrlX`

Abstract

Multi-agent systems perform well on general reasoning tasks. However, the lack of training in specialized areas hinders their accuracy. Current training methods train a unified large language model (LLM) for all agents in the system. This may limit the performances due to different distributions underlying for different agents. Therefore, training multi-agent systems with distinct LLMs should be the next step to solve. However, this approach introduces optimization challenges. For example, agents operate at different frequencies, rollouts involve varying sub-agent invocations, and agents are often deployed across separate servers, disrupting end-to-end gradient flow. To address these issues, we propose M-GRPO, a hierarchical extension of Group Relative Policy Optimization designed for vertical Multi-agent systems with a main agent (planner) and multiple sub-agents (multi-turn tool executors). M-GRPO computes group-relative advantages for both main and sub-agents, maintaining hierarchical credit assignment. It also introduces a trajectory-alignment scheme that generates fixed-size batches despite variable sub-agent invocations. We deploy a decoupled training pipeline in which agents run on separate servers and exchange minimal statistics via a shared store. This enables scalable training without cross-server backpropagation. In experiments on real-world benchmarks (e.g., GAIA, XBench-DeepSearch, and WebWalkerQA), M-GRPO consistently outperforms both single-agent GRPO and multi-agent GRPO with frozen sub-agents, demonstrating improved stability and sample efficiency. These results show that aligning heterogeneous trajectories and decoupling optimization across specialized agents enhances tool-augmented reasoning tasks.

1 Introduction

AI agent systems have demonstrated significant potential in solving complex tasks by integrating large language models (LLMs). Early single-agent approaches, which combine reasoning and actions, have improved robustness and ability in open-domain tasks [Yao et al., 2022, Schick et al., 2023a, Qin et al., 2023, Patil et al., 2024, Xu et al., 2023]. However, as evaluations move from static NLP benchmarks to real-world scenarios, especially in specialized domains like medicine, performance gains are inconsistent. While single agent systems perform well on short tasks, they struggle with long-horizon coordination, compounding errors, and dynamic environments [Zhou et al., 2024, Liu et al., 2025].

Real-world tasks often require decomposition, critique, and iterative processes, which a single agent cannot handle effectively. Benchmarks based on functional websites and multi-environment suites highlight common failures: brittle long-term plans, poor self-checking, and challenges in task delegation [Zhou et al., 2024, Liu et al., 2025]. These limitations drive the need for multi-agent architectures, where agents assume distinct roles (e.g., planner, researcher, solver) and verify each other's actions [Wu et al., 2023, Li et al., 2023, Du et al., 2023]. Vertical architectures, where one agent leads and others report to it, outperform horizontal systems with equal agent roles. Studies show that multi-agent systems with a leader complete tasks nearly 10% faster than those without [Masterman et al., 2024]. Therefore, we adopt a vertical architecture for improved performance. The work flow is illustrated in Figure 1.

While multiple agents improve coverage and reliability, it still requires sophisticated training for complex tasks or specific domains. However, training multi-agent systems with different LLMs intergrated is challenging. The main issue in vertical multi-agent systems is the imbalance in rollout numbers between the leader and task agents, leading

[*]Work done during an internship at Ant Group.

to unstable and asynchronous training. Additionally, deploying different LLMs for each agent across separate training servers complicates gradient flow, making standard backpropagation methods unworkable.

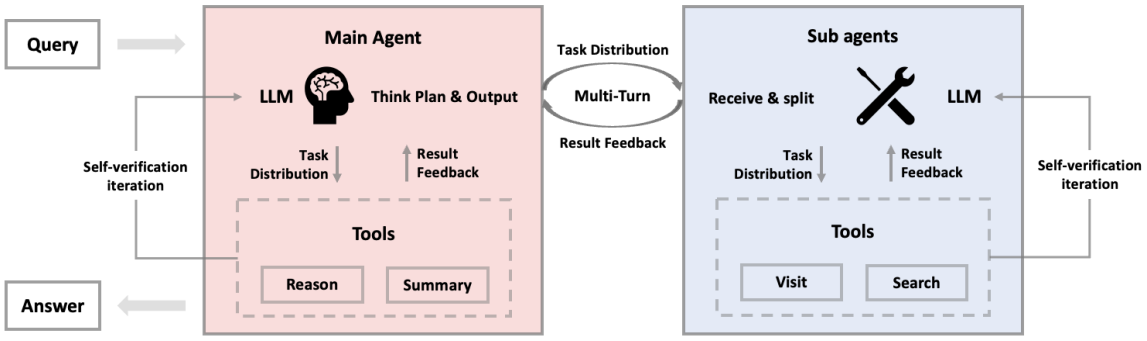

Figure 1: **System workflow with coordinated main and sub-agents.** A user query is fed to the main agent $\mathcal{M}$, which plans, reasons, and delegates subtasks to specialized sub-agents $\{\mathcal{S}_i\}$ if needed (e.g., visit/browsing and search tools). Sub-agents return structured feedback to $\mathcal{M}$, which integrates evidence, performs verification, and produces the final answer. Both $\mathcal{M}$ and $\mathcal{S}_i$ may iterate via self-verification loops before feedback outputs to $\mathcal{M}$.

To address these challenges, we propose **M-GRPO**, a reinforcement-learning framework for training separate LLMs as distinct agents in vertical multi-agent systems. Our contributions include:

- **Vertical multi-agent RL formalization**. We formalize training for a vertical multi-agent architecture where a main agent ($\mathcal{M}$) plans and sub-agents ($\mathcal{S}$) execute tool-use subtasks. We extend Group Relative Policy Optimization to this nested setting by computing group-relative advantages that respect hierarchical credit assignment.
- **Trajectory-alignment for variable sub-invocations**. We introduce a simple but effective alignment scheme to handle the variable number of $\mathcal{S}$ invocations per rollout. By choosing a target $D_{\max}$ and masking (or duplicating/dropping) sub-trajectories, we obtain fixed-shape batches without destabilizing the group baseline, enabling efficient, batched policy-gradient updates.
- **Empirical gains on real-world agent benchmarks.** On GAIA, XBench-DeepSearch and WebWalkerQA benchmarks, our method yields consistent improvements over a strong single-agent GRPO and a multi-agent system with frozen sub-agent baselines across training checkpoints.

2 Related work

2.1 Reinforcement Learning for LLM Post-Training

Reinforcement learning from human feedback (RLHF) [Ouyang et al., 2022] established the contemporary post-training pipeline for aligning large language models (LLMs). It often instantiates proximal policy optimization (PPO) [Schulman et al., 2017] with a Kullback–Leibler (KL) divergence penalty. However, value functions and costly rollouts motivated simpler preference fitting. Accordingly, Direct Preference Optimization (DPO) [Rafailov et al., 2023] and RRHF [Yuan et al., 2023] remove explicit RL while keeping preference alignment. More recently, group-relative objectives such as Group Relative Policy Optimization (GRPO) [Shao et al., 2024a] have shown strong gains in mathematical and program reasoning. DeepSeek-R1 demonstrated that pure-RL can incentivize longer, higher-quality reasoning traces without explicit human-labeled trajectories [Guo et al., 2025]. In parallel, work on language feedback and reflection mechanisms complements preference learning by providing process-level signals rather than outcome-only labels [Shinn et al., 2023]. Taken together, these advances deepen planning and improve reliability of LLMs. As capability rises, AI agent, as an important application mode of LLM, starts to show its extraordinary performances in dealing with multi-step tasks. The evolution of agentic systems has progressed from single-agent architectures to more sophisticated multi-agent frameworks, enabling the collaborative resolution of increasingly complex and high-level goals.

2.2 Single-Agent system training

Early single-agent systems emphasized planning and thinking without changing model parameters. ReAct [Yao et al., 2022] and Self-Ask [Press et al., 2023] structure reasoning–acting–decomposition for complex tasks. Tree of thought (ToT) and Graph of thought (GoT) explore and select among branched thoughts [Yao et al., 2023, Besta et al.,

2024]. To improve reliability, self-monitoring approaches introduce process-level checks and feedback, including Reflexion-style reflection and LLM-as-a-judge techniques [Shinn et al., 2023, Gu et al., 2025].

To further improve the performance of single-agent system, researches also focus on how to fine-tune single-agent system. Single-agent training has progressed from supervised fine-tuning and RLHF to lighter direct preference objectives (DPO/RRHF), process-level rewards (PRMs [Lightman et al., 2023]), and reasoning-targeted RL (GRPO). Aside from aligning LLMs to expected answer formats, other works focus on training single-agent to perform proper and better tool calling [Schick et al., 2023b, Patil et al., 2023]. Despite these advances, a single model often struggles with long-horizon tasks and heterogeneous skills. These limitations motivate a shift toward multi-agent systems that decompose roles to better match the growing complexity of real-world problems.

2.3 Multi-agent system training

Multi-agent systems (MAS) have shown inspiring collective intelligence even in inference-time collaboration [Guo et al., 2024]. Lot of works propose sophisticated frameworks for MAS and shows incredible performances [Chen et al., 2023, Wu et al., 2023, Hong et al., 2024, Qian et al., 2024]. However, when it comes to complex tasks or tasks in specific domains such as medicine, MASs still require training for better performance. Recent work trains collaborative agents end-to-end. For instance, Debate/self-play have been used to produce supervisory signals without ground-truth labels, improving judge accuracy and reasoning quality [Arnesen et al., 2024, Chen et al., 2024, Kenton et al., 2024, Xue et al., 2025]. Surveys synthesize these directions and open issues around coordination, evaluation, and reward shaping in LLM-based MASs [Chen et al., 2025c, Luo et al., 2025, Sun et al., 2024].

Recent works introduce RL training frameworks for multi-agent system. Bo et al. [2024] introduces agent-level rewards and counterfactual credit assignment tailored to LLM agents. MALT [Motwani et al., 2025] organizes generator–verifier–refiner roles and propagates credit across a pipeline to improve math and commonsense reasoning. MAPoRL [Park et al., 2025] co-trains agents with multi-agent reinforcement learning to elicit cooperative behaviors via discussion rewards, reporting gains over single-agent post-training. The method adjusts PPO for MAS training. Hence, it can be costly with value function required. Further, Mo et al. [2025] and Chen et al. [2025a] propose GRPO-based methods for MAS training. However, both of them share a single underlying LLM across agents, which ties parameters and inductive biases. This hinders role specialization when agents perform heterogeneous functions (e.g., plan vs. split task) with distinct state–action/data distributions, motivating role-specialized LLMs. Team [2025a] co-trains dual agents with distinct LLMs under a synchronous Q&A loop where both take the same number of actions per rollout. In vertical systems, however, agents act at different frequencies, creating reward misalignment and unstable cross-agent credit assignment. We address this with *Multi-agent Group Relative Policy Optimization* (M-GRPO), which aligns heterogeneous trajectories via group-relative advantages.

3 Problem setup

3.1 Architecture setup

We employ a vertical two-agent architecture as the base case for hierarchical systems. A **main agent** $\mathcal{M}$ plans, decomposes a user query q into subtasks, decides when to delegate, and synthesizes a final answer. It is also equipped with a reason tool for analyzing complex question. A **sub-agent** $\mathcal{S}$ executes delegated subtasks that typically involve tool use (e.g., search, visit) and reports findings back to $\mathcal{M}$. Communication follows a call–return protocol: $\mathcal{M}$ issues a subtask specification and optional context to $\mathcal{S}$, $\mathcal{S}$ returns structured results (evidence, summaries, and metadata). This two-agent setting extends naturally to $1 + n$ agents by instantiating a set $\{\mathcal{S}_i\}_{i=1}^{n}$ with the same architecture but different functions. Without loss of generality, we focus on $n = 1$ to present the core method cleanly in this paper (Figure. 2 illustrates the workflow in one rollout).

Action space. Each agent's policy comprises (i) *token-level* textual emissions and (ii) *discrete* control actions (tool invocation, delegation, stop). Text actions are trained at the token level; control actions are treated as single-step categorical choices interleaved in the trajectory.

Notations. For one rollout in Figure. 2, the trajectory (including actions, states and rewards) and outputs are

$$\tau = \left\{\tau_{\mathcal{M}},\ \{\tau_{\mathcal{S}_i}\}_{i=1}^{d}\right\} \tag{1}$$

$$\tau_{\mathcal{M}} = \left\{a_{\mathcal{M},t},\ s_{\mathcal{M},t},\ r_{\mathcal{M},t}\right\}_{t=1}^{T_{\mathcal{M}}}, \quad \tau_{\mathcal{S}_i} = \left\{a_{\mathcal{S}_i,t},\ s_{\mathcal{S}_i,t},\ r_{\mathcal{S}_i,t}\right\}_{t=1}^{T_{\mathcal{S}_i}} \tag{2}$$

$$o = \left\{o_{\mathcal{M}},\ \{o_{\mathcal{S}_i}\}_{i=1}^{d}\right\} \tag{3}$$

where d is the number of sub-agent invocations made by $\mathcal{M}$ in this rollout. The value of d varies across rollouts. At time step t out of total step $T_{\mathcal{M}}$, the main agent trajectory ($\tau_{\mathcal{M}}$) consists of action ($a_{\mathcal{M},t}$), state ($s_{\mathcal{M},t}$) and reward ($r_{\mathcal{M},t}$). In practice, the reward $r_{\mathcal{M},t}$ is broadcasted from the reward of final output of the main agent. Notations are

similar for the i-th invocation of subagent. The final answer to $\mathcal{M}$ is $o_{\mathcal{M}}$; each sub-invocation returns $o_{\mathcal{S},i}$. Agent policies are denoted by $\pi_{\theta_{\mathcal{M}}}$ and $\pi_{\theta_{\mathcal{S}}}$, mapping states to actions.

3.2 Reward setup

The main agent is evaluated based on the quality of its final output $o_{\mathcal{M}}$:

$$\mathcal{R}_{\mathcal{M}}(o_{\mathcal{M}}) = \begin{cases} \alpha_1 \cdot r_{\text{format}} + \alpha_2 \cdot r_{\text{correct}} & \text{valid format} \\ 0 & \text{invalid format} \end{cases} \tag{4}$$

where:

- r_{format}: Binary reward for valid output structure (proper JSON, required fields).
- r_{correct}: Accuracy reward comparing final answer to ground truth.
- α_1 and α_2 are hyperparameters that balance these two features.

If the output is in invalid format, the trajectory gets zero reward, regardless of correctness. This strict policy ensures:

- Invalid formats cannot break downstream processing in real deployments.
- Partial correctness with invalid format is penalized early, driving correct structural learning.

The **Sub-Agent** receives a composite reward that balances local execution quality and global outcome alignment:

$$\mathcal{R}_{\mathcal{S}}(o_{\mathcal{S}_i}) = \begin{cases} \beta_1 \cdot r_{\text{format}} + \beta_2 \cdot r_{\text{correct}}^{\text{main}} + \beta_3 \cdot r_{\text{expert}} & \text{valid format} \\ 0 & \text{invalid format} \end{cases} \tag{5}$$

where:

- r_{format}: Format validation (same zero-reward policy as Main Agent).
- $r_{\text{correct}}^{\text{main}}$: Main Agent's correctness reward, replicated to Sub-Agent.
- r_{expert}: LLM-based evaluation of subtask execution quality.
- β_1, β_2, β_3 are hyperparameters that balance the three reward terms.

The coefficients for each component should be settled considering following aspects.

- Format weight: Ensures the Sub-Agent outputs valid structures that the Main Agent can parse.
- Correctness weight: Ensures the Sub-Agent's output contributes directly to the final result.
- Expert weight: Assesses the quality of subtask execution, guide the Sub-Agent in tool usage and response.

3.3 Targeted problem

As shown in Figure 1, multiple sub-agent invocations may occur during one single answering rollout. Therefore, in a batch with multiple training samples, the ratio between the number of main agent and sub-agent trajectories is not fixed and can vary across batches and training steps. To address this issue, we propose M-GRPO, which is detailed with implementation in the following section.

4 Methodology

4.1 M-GRPO

Training objective. For training the multi-agent system, we adopt Group Relative Policy Optimization (GRPO) [Shao et al., 2024b] and extend it to the multi-agent setting. We refer to this extension as ***M**ulti-Agent **GRPO** (M-GRPO)*. The method preserves the group-relative baseline while respecting the hierarchy between $\mathcal{M}$ and $\mathcal{S}$.

Group-relative advantages. For a query q, we collect K rollouts, the trajectories and outputs can be denoted:

$$\{\tau_k = \{\tau_{\mathcal{M}}^{(k)}, \{\tau_{\mathcal{S}_i}^{(k)}\}_{i=1}^{d_k}\}\}_{k=1}^{K} \tag{6}$$

$$\tau_{\mathcal{M}}^{(k)} = \{a_{\mathcal{M},t}^{(k)},\ s_{\mathcal{M},t}^{(k)},\ r_{\mathcal{M},t}^{(k)}\}_{t=1}^{T_{\mathcal{M}}^{(k)}}, \quad \tau_{\mathcal{S}_i}^{(k)} = \{a_{\mathcal{S}_i,t}^{(k)},\ s_{\mathcal{S}_i,t}^{(k)},\ r_{\mathcal{S}_i,t}^{(k)}\}_{t=1}^{T_{\mathcal{S}_i}^{(k)}} \tag{7}$$

$$\{o_k = \{o_{\mathcal{M}}^{(k)}, \{o_{\mathcal{S}_i}^{(k)}\}_{i=1}^{d_k}\}\}_{k=1}^{K} \tag{8}$$

For these rollouts, the numbers of main agent $\mathcal{M}$ invocating sub agent $\mathcal{S}$ are d_1, d_2, ..., d_K respectively, i.e. $i \in \{d_1, d_2, ..., d_K\}$. Now we compute centered, normalized returns and advantages.

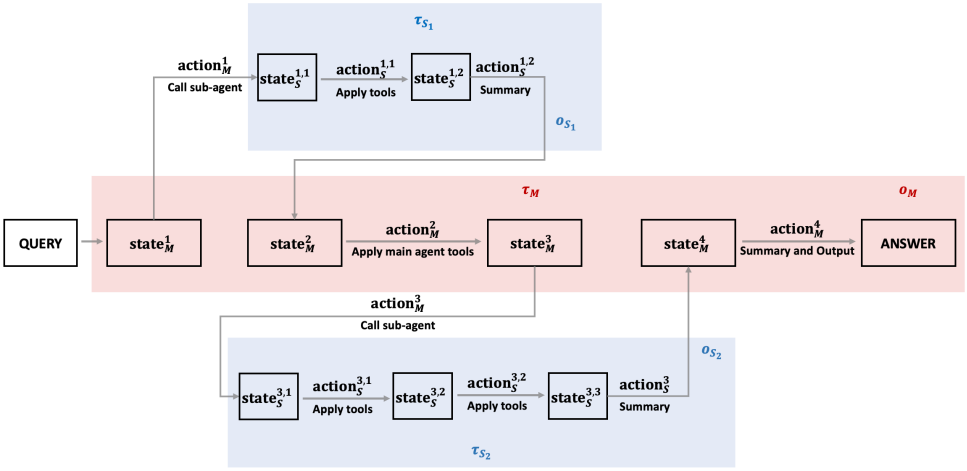

Figure 2: **One rollout with nested $\mathcal{M} \to \mathcal{S}$ interactions.** The main agent $\mathcal{M}$ follows trajectory $\tau_{\mathcal{M}}$ (red) and may distribute subtasks by invoking the sub-agent $\mathcal{S}$ multiple times (e.g., $a^1_{\mathcal{M}}$ and $a^3_{\mathcal{M}}$). Each invocation generates a sub-trajectory $\tau_{\mathcal{S}_i}$ (blue) that performs tool-use steps and returns a summarized message $o_{\mathcal{S}_i}$ to $\mathcal{M}$. The main trajectory integrates these intermediate results and finally outputs the answer $o_{\mathcal{M}}$.

For **main agent** $\mathcal{M}$:

$$\mu_{q,\mathcal{M}} = \frac{1}{K}\sum_{k=1}^{K}\mathcal{R}_{\mathcal{M}}(o_{\mathcal{M}}^{(k)}), \quad \sigma_{q,\mathcal{M}} = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\left(\mathcal{R}_{\mathcal{M}}(o_{\mathcal{M}}^{(k)}) - \mu_{q,\mathcal{M}}\right)^2} \tag{9}$$

$$\hat{A}_{q,\mathcal{M}}^{(k)} = \hat{A}_{q,\mathcal{M},t}^{(k)} = \frac{\mathcal{R}_{\mathcal{M}}(o_{\mathcal{M}}^{(k)}) - \mu_{q,\mathcal{M}}}{\sigma_{q,\mathcal{M}}} \quad \forall\, t \in \{1, 2, \cdots, T_{\mathcal{M}}^{(k)}\} \tag{10}$$

For **sub-agent** $\mathcal{S}$, to align the varying invocation counts d_k across K rollouts and enable efficient batch RL training, we propose a batching scheme. Let d denote an approximate upper bound on the number of sub-agent invocations in one rollout, such that

$$P(d_k \le d) \approx 0.99 \quad d \text{ can be selected by a preliminary sanity check} \tag{11}$$

For each invocation in one rollout, align the number of subagent trajectories according to the rules:

- If $d_k < d$, randomly duplicate $d - d_k$ trajectories from $\{\tau_{\mathcal{S}_i}^{(k)}\}_{i=1}^{d_k}$ and fill them as a whole batch. For example, $\tau_{\mathcal{S}_{d_k+1}}^{(k)} = \tau_{\mathcal{S}_j}^{(k)}$ where j is randomly chosen from $\{1, 2, ..., d_k\}$.
- If $d_k > d$, randomly drop $d_k - d$ trajectories from $\{\tau_{\mathcal{S}_i}^{(k)}\}_{i=1}^{d_k}$.

After this adjustment, for K rollouts corresponding to any query q, the number of sub-agent invocations will be $d \times K$, and the sub-agent trajectories will become $\{\tau_{\mathcal{S}_i}^{(k)}\}_{i=1}^{d}$. We then follow the analogous computation logic for the main agent $\mathcal{M}$.

$$\mu_{q,\mathcal{S}} = \frac{1}{d\times K}\sum_{k=1}^{K}\sum_{i=1}^{d}\mathcal{R}_{\mathcal{S}}(o_{\mathcal{S}_i}^{(k)}), \quad \sigma_{q,\mathcal{S}} = \sqrt{\frac{1}{d\times K}\sum_{k=1}^{K}\sum_{i=1}^{d}\left(\mathcal{R}_{\mathcal{S}}(o_{\mathcal{S}_i}^{(k)}) - \mu_{q,\mathcal{S}}\right)^2} \tag{12}$$

$$\hat{A}_{q,\mathcal{S}}^{(k),i} = \hat{A}_{q,\mathcal{S},t}^{(k),i} = \frac{\mathcal{R}_{\mathcal{S}}(o_{\mathcal{S}_i}^{(k)}) - \mu_{q,\mathcal{S}}}{\sigma_{q,\mathcal{S}}} \quad \forall\, t \in \{1, 2, \cdots, T_{\mathcal{S}_i}^{(k)}\} \tag{13}$$

M-GRPO objective. With defined advantages, the objective functions for main and sub agents are defined as follows.

$$\mathcal{J}_{\text{M-GRPO}}(\theta_{\mathcal{M}}) = \mathbb{E}_{\{q\sim P(Q),\ \{o_{\mathcal{M}}^{(k)}\}_{k=1}^{K}\sim\pi_{\theta_{\mathcal{M}}}(o|q)\}} \left[\frac{1}{K}\sum_{k=1}^{K}\min\left(\frac{\pi_{\theta_{\mathcal{M}}}(o_{\mathcal{M}}^{(k)}|q)}{\pi_{\theta_{\mathcal{M}_{old}}}(o_{\mathcal{M}}^{(k)}|q)}\hat{A}_{q,\mathcal{M}}^{(k)},\ \text{clip}\left(\frac{\pi_{\theta_{\mathcal{M}}}(o_{\mathcal{M}}^{(k)}|q)}{\pi_{\theta_{\mathcal{M}_{old}}}(o_{\mathcal{M}}^{(k)}|q)},\ 1-\varepsilon,\ 1+\varepsilon\right)\hat{A}_{q,\mathcal{M}}^{(k)}\right)\right] \tag{14}$$

where $p(Q)$ is the distribution of the query and $\pi_{\theta_\mathcal{M}}(o_\mathcal{M}^{(k)}|q)$ is the likelihood of the final answer given the query q.

$$\pi_{\theta_\mathcal{M}}(o_\mathcal{M}^{(k)}|q) = \prod_{t=1}^{T_\mathcal{M}^{(k)}} \pi_{\theta_\mathcal{M}}(a_{\mathcal{M},t}^{(k)}|s_{\mathcal{M},t}^{(k)}) \tag{15}$$

The sub-agent objective sums over all its invocations:

$$\mathcal{J}_{\text{M-GRPO}}(\theta_\mathcal{S}) = \mathbb{E}_{\{q\sim P(Q),\ \{o_{\mathcal{S}_1}^{(k)},\ldots,o_{\mathcal{S}_d}^{(k)}\}_{k=1}^{K}\sim\pi_{\theta_\mathcal{M}}(o|q)\}} \left[\frac{1}{dK}\sum_{k=1}^{K}\sum_{i=1}^{d}\min\left(\frac{\pi_{\theta_\mathcal{S}}(o_{\mathcal{S}_i}^{(k)}|q)}{\pi_{\theta_{\mathcal{S}_{old}}}(o_{\mathcal{S}_i}^{(k)}|q)}\hat{A}_{q,\mathcal{S}}^{(k),i},\ \text{clip}\left(\frac{\pi_{\theta_\mathcal{S}}(o_{\mathcal{S}_i}^{(k)}|q)}{\pi_{\theta_{\mathcal{S}_{old}}}(o_{\mathcal{S}_i}^{(k)}|q)},\ 1-\varepsilon,\ 1+\varepsilon\right)\hat{A}_{q,\mathcal{S}}^{(k),i}\right)\right] \tag{16}$$

Similarly, $\pi_{\theta_\mathcal{S}}(o_{\mathcal{S}_i}^{(k)}|q)$ is the likelihood of the answer to the i-th sub-agent invocation given the query q.

$$\pi_{\theta_\mathcal{S}}(o_{\mathcal{S}_i}^{(k)}|q) = \prod_{t=1}^{T_{\mathcal{S}_i}^{(k)}} \pi_{\theta_{\mathcal{S}_i}}(a_{\mathcal{S}_i,t}^{(k)}|s_{\mathcal{S}_i,t}^{(k)}) \tag{17}$$

We maximize $\mathcal{J}_{\text{M-GRPO}}(\theta_\mathcal{M})$ and $\mathcal{J}_{\text{M-GRPO}}(\theta_\mathcal{S})$ separately for distinct LLMs integrated in main and sub agents respectively. Specifically:

$$\hat{\theta}_\mathcal{M} = \arg\max_{\theta_\mathcal{M}} \mathcal{J}_{\text{M-GRPO}}(\theta_\mathcal{M}) \tag{18}$$

$$\hat{\theta}_\mathcal{S} = \arg\max_{\theta_\mathcal{S}} \mathcal{J}_{\text{M-GRPO}}(\theta_\mathcal{S}) \tag{19}$$

4.2 Implementation

We deploy the main agent and the sub-agent on separate rollout servers, server M and server S. For each query in a batch, we run 8 rollouts and select d to be 8. Following prior work [Team, 2025a], we update the main agent on server M with its rewards and write its rewards to a shared database. We compute the log likelihood on batches containing $\tau_\mathcal{M}$ and $\tau_\mathcal{S}$ separately using $\pi_{\theta_\mathcal{M}}$, $\pi_{\theta_{\mathcal{M}_{old}}}$ and $\pi_{\theta_\mathcal{M}}$, $\pi_{\theta_{\mathcal{S}_{old}}}$. Server S then reads the required rewards from the database, computes the rewards for sub-agents and updates the sub-agent accordingly. The illustration of the subagent trajectory alignment process and training on different servers is shown in Figure 3, 4. As for tools, main agent has the access to a reasoning tool which is able to solve logical math problems. Sub agents can invoke visit and search tools for desired knowledge.

5 Experiments

We evaluate our multi-agent training architecture on domain-specific and general-purpose benchmarks. In this paper, we focus on the scenario that the multi-agent system tries to find the answer for a given query by a multi-turn deep search loop. Performance is measured by (i) answer correctness and (ii) adherence to the required output format.

5.1 Experimental Setup

Curriculum learning. We adopt a two-stage reinforcement learning curriculum [Narvekar et al., 2020]. In stage 1, we train the model on a simple dataset to enable rapid learning of basic answering formats. In stage 2, we introduce more complex problems that encourage collaborative problem-solving within the multi-agent framework.

Training data. The training datasets for both stages are constructed separately. For stage 1, we use the data synthesis approach from Yu et al. [2025], but with simpler graphs containing fewer nodes and edges to facilitate quick format learning. For stage 2, we adopt the same method but by building graphs with more nodes and edges. This helps to create more challenging datasets. We also select queries that achieve low success rates in the same multi-agent system with untrained LLMs, thereby focusing on difficult collaborative problem-solving scenarios.

Benchmarks. We evaluate our trained MAS on three benchmarks.

- **GAIA** [Mialon et al., 2023]. Tests real-world assistant capabilities on multi-modal tasks involving tools, web search, and multi-step reasoning. It emphasizes human-simple yet AI-hard skills, such as reading comprehension, logical reasoning, and effective tool use in realistic settings.

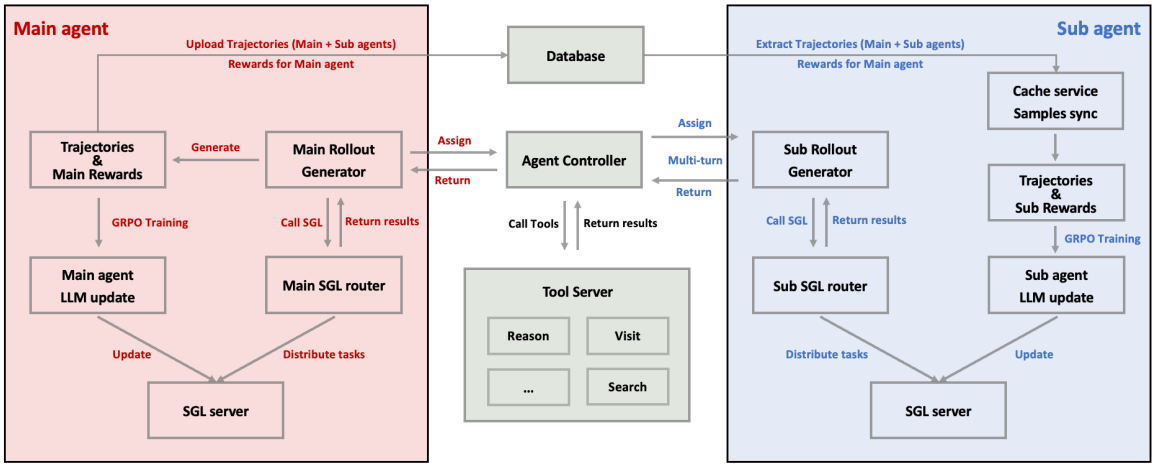

Figure 3: **Workflow of the decoupled two-agent architecture with M-GRPO.** The Main agent (left) and Sub agent (right) each generate rollouts via their SGL router/server. Main agent logs trajectories and rewards to a shared Database. Sub agent extracts the required rewards from the database and calculates its own rewards for training. A central Agent Controller (middle) coordinates multi-turn interactions, assigns subtasks to the sub agent, and aggregates returned results. Tool calls (reason/search/visit) are executed through a Tool Server. The sub-agent side maintains a cache for sample synchronization. Arrows indicate data and control flow.

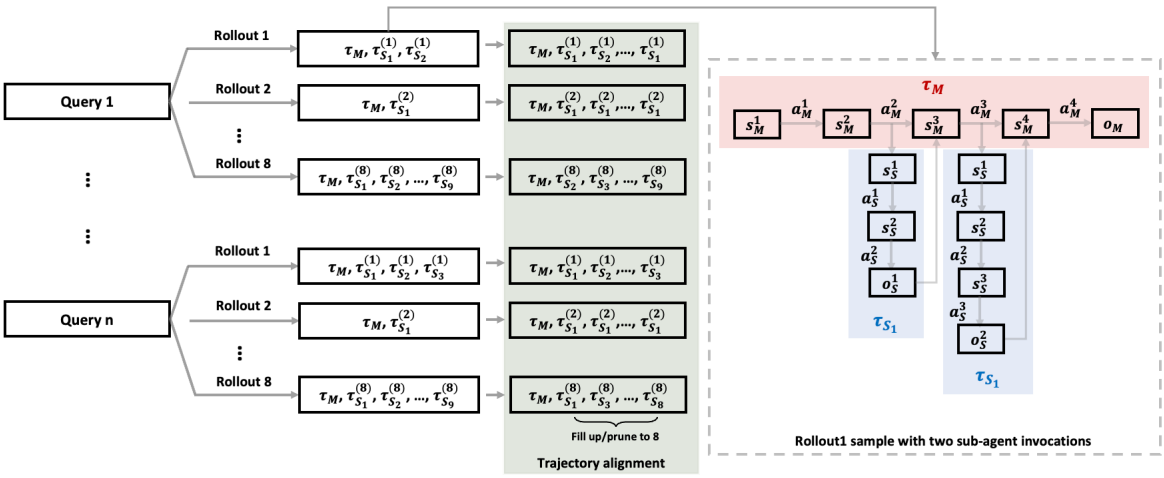

Figure 4: **Trajectory alignment for batch training with variable sub-agent invocations.** For each query, we sample K rollouts. Every rollout yields one main-trajectory $\tau_{\mathcal{M}}$ and a variable number of sub-agent trajectories $\{\tau_{\mathcal{S}_i}\}$. Because the number of sub-invocations d_k differs across rollouts, we fix a target d (e.g., 8) and randomly duplicate or drop $\tau_{\mathcal{S}}$ samples so that each batch contains a consistent count of $\tau_{\mathcal{S}}$ (while keeping a fixed number of $\tau_{\mathcal{M}}$, e.g., 8). This alignment produces uniform tensor shapes for policy-gradient updates.

- **XBench-DeepSearch** [Chen et al., 2025b]. Assesses tool use across diverse open-domain scenarios, including fact-checking, comparative analysis, browsing-based reasoning, and complex information synthesis. It provides a broad measure of real-world problem solving.
- **WebWalkerQA** [Wu et al., 2025]. Evaluates web navigation and information extraction capabilities through multi-step browsing tasks.

Baselines. To validate the performance improvement of updating both main and sub agents together with M-GRPO, we compare our methods with two baselines. To better illustrate the role of M-GRPO in training, we initialize all baselines from the same stage 1 training checkpoint.

- **Multi-agent system with fixed sub-agent**. A multi-agent system with the same architecture, where the main agent is updated using common GRPO, while the parameters of the sub-agent are frozen. To compare performance on answering rather than formatting, we freeze the sub-agent parameters exclusively in Stage 2.

Model and infrastructure. We initialize both single-agent and multi-agent systems with Qwen3-30B-A3B [Team, 2025b]. All training is conducted using the SLIME framework [Zhu et al., 2025]. The single-agent baseline is trained on 32 H800 GPUs, while the multi-agent system uses 2×32 H800 GPUs, with one cluster for the main agent ($\mathcal{M}$) and one for sub-agent ($\mathcal{S}$).

Training algorithms. For the single-agent baseline, we provide access to all tools available to the sub-agent $\mathcal{S}$ in the multi-agent system, while keeping other configurations (model architecture, hyperparameters, etc.) identical to the main agent $\mathcal{M}$. We apply GRPO with the format reward weight $\alpha_1 = 0.1$ and $\alpha_2 = 0.9$ in Eq. (4). In the multi-agent system, the main agent $\mathcal{M}$ uses M-GRPO with the reward structure specified in Eq. (4) and the same α values. The sub-agent $\mathcal{S}$ receives a composite reward as defined in Eq. (5) with $\beta_1 = 0.1$, $\beta_2 = 0.4$ and $\beta_3 = 0.5$. The 0.4/0.5 balance is crucial. Too much weight on global outcome creates noisy gradients (Sub-Agent performed well but Main Agent failed elsewhere), while too much weight on local quality allows Sub-Agents to optimize for metrics that do not contribute to final success.

5.2 Stage 1: Format learning with simple data

Figure 5 shows the reward curve during stage 1 RL training. The reward trajectory during stage 1 demonstrates a stable learning with rewards rising steadily from zero to a high plateau. This validates that our curriculum design enables rapid acquisition of proper output formatting with a simple QA dataset in the first stage. It also establishes a foundation for subsequent complex collaborative learning.

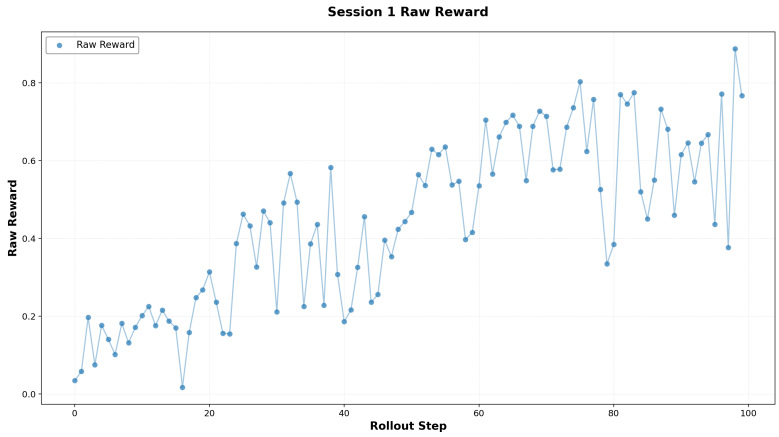

Figure 5: **Reward curve during Stage 1 RL training on simple data**. The system shows stable improvement from zero to high rewards, demonstrating effective format acquisition.

5.3 Stage 2: Learning multi-agent collaboration on complex tasks

After establishing format competence in stage 1, we proceed to stage 2 RL training using challenging problems and designed M-GRPO. The goal is to elicit and strengthen collaborative problem-solving behaviors between $\mathcal{M}$ and $\mathcal{S}$ under complex task demands.

5.3.1 Main results: benchmark performance

To assess whether the collaborative capabilities learned during stage 2 RL training generalize to real-world tasks, we evaluate checkpoints on three benchmarks throughout the stage 2 training process. Figure 6 presents performance trajectories on GAIA, XBench-DeepSearch, and WebWalkerQA, evaluated every 25 training steps. We compare co-training (both agents updated) versus main-only training (sub-agent fixed) and single-agent training. All reported results are averaged over three independent inference runs to ensure reliability.

The results show that co-training yields consistent improvements over main-only training across all three benchmarks throughout stage 2. This demonstrates that the collaborative problem-solving capabilities elicited by M-GRPO on challenging training data effectively transfer to diverse real-world evaluation scenarios.

Figure 6: **Benchmark performance during Stage 2 training (averaged over three runs)**. Co-training both agents (blue) consistently outperforms main-only training (purple) across GAIA, XBench-DeepSearch, and WebWalkerQA benchmarks, demonstrating that learned collaborative behaviors transfer to diverse real-world tasks.

5.3.2 Ablation study: training configurations

To understand the source of the performance gains observed in Figure 6 and validate the contribution of different components in our framework, we conduct ablation studies comparing three training configurations during stage 2:

1. **Multi-agent co-training**: Both $\mathcal{M}$ and $\mathcal{S}$ are trained jointly using M-GRPO (our full method).
2. **Multi-agent main-only**: Only $\mathcal{M}$ is trained while $\mathcal{S}$ remains fixed (the orange curves in Figure 6).
3. **Single-agent baseline**: A single agent handles both planning and tool execution with access to all tools available to $\mathcal{S}$, maintaining identical model architecture and hyperparameters as $\mathcal{M}$. This baseline is trained with a separate configuration including its own format warm-up phase.

For fair comparison, all multi-agent configurations initialize from the stage 1 checkpoint. Figure 7 shows the stage 2 learning curves. Since the raw reward curves exhibit high variance, we overlay exponential moving average (EMA) smoothed curves to better visualize the training trends.

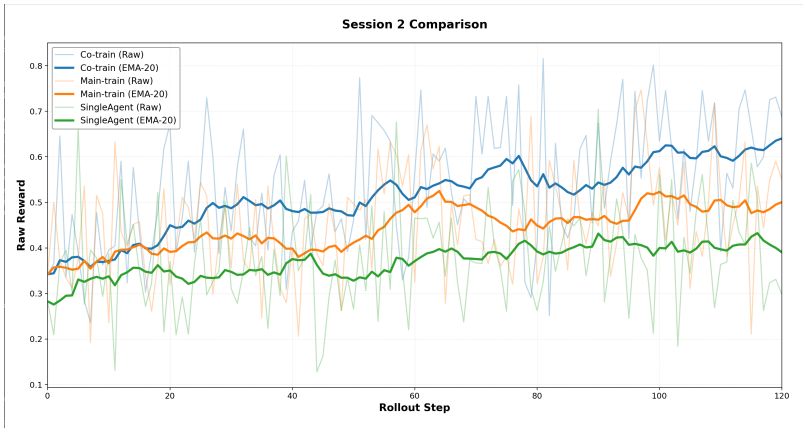

Figure 7: **Stage 2 RL learning curves on challenging data comparing three training configurations**. Raw rewards and EMA-smoothed trends are shown. Co-training both agents achieves the highest rewards, followed by main-only training, with single-agent baseline showing the lowest performance.

The results demonstrate a clear performance hierarchy: **co-training** $>$ **main-only** $>$ **single-agent**. This validates that (1) the multi-agent architecture itself provides benefits over single-agent systems when facing complex tasks, and

(2) joint optimization of both agents through M-GRPO further improves collaborative problem-solving compared to training only the main agent.

5.3.3 Ablation study: trajectory synchronization

To maintain more on-policy training dynamics, we introduce a sample-level synchronization strategy as described in Section 4.1. This mechanism aligns trajectories across agents by fixing the number of sub-agent invocations to a target value d through duplication or dropping of sub-trajectories. To validate the effectiveness of this design choice, we compare two implementations during stage 2:

1. **Without synchronization**: An earlier implementation that uses off-policy data without considering sample-level alignment across agents.
2. **With synchronization**: Our current implementation that enforces trajectory alignment to maintain more consistent policy-data correspondence.

Figure 8 shows the stage 2 learning curves for both implementations, with EMA smoothing applied to highlight trends. The synchronized version achieves better performance, demonstrating that sample-level synchronization between agents improves training stability and collaborative learning effectiveness by keeping the training process closer to on-policy.

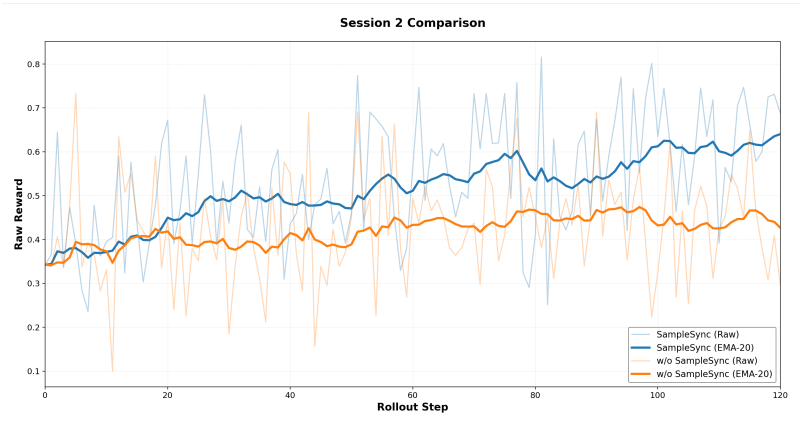

Figure 8: **Stage 2 RL learning curves comparing implementations with and without trajectory synchronization**. Raw rewards (lighter lines) and EMA-smoothed trends (darker lines) are shown. The synchronized version (blue) outperforms the unsynchronized version (orange), validating the benefit of maintaining more on-policy training dynamics.

5.4 Summary

Our experiments demonstrate several key findings:

- The two-stage RL curriculum (stage 1 on simple data for format learning, stage 2 on challenging data for collaborative capability development) enables stable training progression (Figure 5).
- Co-training both agents consistently outperforms main-only training across multiple real-world benchmarks throughout stage 2, demonstrating effective transfer of learned collaborative behaviors (Figure 6).
- During stage 2 RL training, the performance hierarchy co-training $>$ main-only $>$ single-agent validates both the multi-agent architecture and the M-GRPO joint optimization approach (Figure 7).
- Trajectory synchronization improves training stability and collaborative learning effectiveness during stage 2 by maintaining more on-policy training dynamics (Figure 8).

These results validate M-GRPO as an effective framework for training collaborative LLM agents on complex, tool-augmented tasks.

6 Conclusion

In this paper, we introduce M-GRPO, an effective reinforcement learning framework designed for training multi-agent systems that utilize LLMs. Our approach leverages hierarchical credit assignment and trajectory alignment to address key challenges in vertical multi-agent architectures, including imbalanced rollout numbers and complex cross-server gradient flow. Through empirical evaluation, we demonstrate that M-GRPO outperforms traditional single-agent systems and fixed sub-agent configurations on real-world benchmarks, such as GAIA, XBench and WebWalker QA.

The findings highlight the importance of multi-agent collaboration in complex tasks, where M-GRPO effectively fosters task-specific expertise among sub-agents while maintaining global alignment with the main agent's goals. Our experimental results validate that joint optimization of both main and sub-agents leads to superior performance in long-horizon planning and tool-augmented reasoning.

References

Samuel Arnesen, David Rein, and Julian Michael. Training language models to win debates with self-play improves judge accuracy, 2024. URL https://arxiv.org/abs/2409.16636.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16): 17682–17690, March 2024. ISSN 2159-5399. doi:10.1609/aaai.v38i16.29720. URL http://dx.doi.org/10.1609/aaai.v38i16.29720.

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. Reflective multi-agent collaboration based on large language models. *Advances in Neural Information Processing Systems*, 37:138595–138631, 2024.

Guoxin Chen, Zile Qiao, Wenqing Wang, Donglei Yu, Xuanzhong Chen, Hao Sun, Minpeng Liao, Kai Fan, Yong Jiang, Penguin Xie, Wayne Xin Zhao, Ruihua Song, and Fei Huang. Mars: Optimizing dual-system deep research via multi-agent reinforcement learning, 2025a. URL https://arxiv.org/abs/2510.04935.

Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Haotong Tian, Tianbao Xie, Fangfu Liu, Haoye Zhang, Hongzhang Liu, Yuan Gong, Chen Sun, Han Hou, Hui Yang, James Pan, Jianan Lou, Jiayi Mao, Jizheng Liu, Jinpeng Li, Kangyi Liu, Kenkun Liu, Rui Wang, Run Li, Tong Niu, Wenlong Zhang, Wenqi Yan, Xuanzheng Wang, Yuchen Zhang, Yi-Hsin Hung, Yuan Jiang, Zexuan Liu, Zihan Yin, Zijian Ma, and Zhiwen Mo. xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations, 2025b. URL https://arxiv.org/abs/2506.13651.

Shuaihang Chen, Yuanxing Liu, Wei Han, Weinan Zhang, and Ting Liu. A survey on llm-based multi-agent system: Recent advances and new frontiers in application, 2025c. URL https://arxiv.org/abs/2412.17481.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors, 2023. URL https://arxiv.org/abs/2308.10848.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models, 2024. URL https://arxiv.org/abs/2401.01335.

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

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. A survey on llm-as-a-judge, 2025. URL https://arxiv.org/abs/2411.15594.

Dong Guo et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges, 2024. URL https://arxiv.org/abs/2402.01680.

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

Zachary Kenton, Noah Y. Siegel, János Kramár, Jonah Brown-Cohen, Samuel Albanie, Jannis Bulian, Rishabh Agarwal, David Lindner, Yunhao Tang, Noah D. Goodman, and Rohin Shah. On scalable oversight with weak llms judging strong llms, 2024. URL https://arxiv.org/abs/2407.04622.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society, 2023. URL https://arxiv.org/abs/2303.17760.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2025. URL https://arxiv.org/abs/2308.03688.

Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, Yiqiao Jin, Fan Zhang, Xian Wu, Hanqing Zhao, Dacheng Tao, Philip S. Yu, and Ming Zhang. Large language model agent: A survey on methodology, applications and challenges, 2025. URL https://arxiv.org/abs/2503.21460.

Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey, 2024. URL https://arxiv.org/abs/2404.11584.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023. URL https://arxiv.org/abs/2311.12983.

Zhanfeng Mo, Xingxuan Li, Yuntao Chen, and Lidong Bing. Multi-agent tool-integrated policy optimization, 2025. URL https://arxiv.org/abs/2510.04678.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip H. S. Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improving reasoning with multi-agent llm training, 2025. URL https://arxiv.org/abs/2412.01928.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey, 2020. URL https://arxiv.org/abs/2003.04960.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

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

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023. URL https://arxiv.org/abs/2305.15334.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models, 2023. URL https://arxiv.org/abs/2210.03350.

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

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789.

Rafael Rafailov et al. Direct preference optimization: Your language model is secretly a reward model. *arXiv:2305.18290*, 2023. URL https://arxiv.org/abs/2305.18290.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023a.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023b. URL https://arxiv.org/abs/2302.04761.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhen Shao et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv:2402.03300*, 2024a. URL https://arxiv.org/abs/2402.03300. Introduces GRPO.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024b. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

Chuanneng Sun, Songjun Huang, and Dario Pompili. Llm-based multi-agent reinforcement learning: Current and future directions, 2024. URL https://arxiv.org/abs/2405.11106.

AQ Team. Mrlx: A multi-agent reinforcement learning framework. https://github.com/AQ-MedAI/MrlX, 2025a.

Qwen Team. Qwen3 technical report, 2025b. URL https://arxiv.org/abs/2505.09388.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. Webwalker: Benchmarking llms in web traversal, 2025. URL https://arxiv.org/abs/2501.07572.

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, 2023. URL https://arxiv.org/abs/2308.08155.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models, 2023. URL https://arxiv.org/abs/2305.18323.

Xiangyuan Xue, Yifan Zhou, Guibin Zhang, Zaibin Zhang, Yijiang Li, Chen Zhang, Zhenfei Yin, Philip Torr, Wanli Ouyang, and Lei Bai. Comas: Co-evolving multi-agent systems via interaction rewards, 2025. URL https://arxiv.org/abs/2510.08529.

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.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Ailing Yu, Lan Yao, Jingnan Liu, Zhe Chen, Jiajun Yin, Yuan Wang, Xinhao Liao, Zhiling Ye, Ji Li, Yun Yue, Hansong Xiao, Hualei Zhou, Chunxiao Guo, Peng Wei, Junwei Liu, and Jinjie Gu. Medresearcher-r1: Expert-level medical deep researcher via a knowledge-informed trajectory synthesis framework, 2025. URL https://arxiv.org/abs/2508.14880.

Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, Songfang Huang, and Fei Huang. Rrhf: Rank responses to align language models with human feedback without tears, 2023. URL https://arxiv.org/abs/2304.05302.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An llm post-training framework for rl scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

A Appendix

In appendix, we present several case studies to show that the multi-agent system can learn targeted functions.

A.1 Tool call

After training with M-GRPO, muti-agent system (MAS) learns how to select the optimal tools for question solution.

Question

A standard Rubik's cube has been broken into cubes making up its sides. The cubes are jumbled, and one is removed. There are 6 cubes with one colored face, 12 edge cubes with two colored faces, and 8 corner cubes with three colored faces. All blue cubes have been found. All cubes directly left, right, above, and below the orange center cube have been found, along with the center cube. The green corners have all been found, along with all green that borders yellow. For all orange cubes found, the opposite face's cubes have been found. The removed cube has two colors on its faces. What are they? Answer using a comma separated list, with the colors ordered alphabetically.

The correct answer should be "green, white." Before training, MAS with the same vertical architecture produces the following outputs. The answer given before training is incorrect, whereas after training, the MAS can correctly answer the question. We explore the reasons for this discrepancy in the MAS's thinking process.

Answer before training

"prediction": "Green,Orange"

Answer after training

"prediction": "green, white"

The thinking processes differ before and after training in a MAS, as the system learns how to select the appropriate tools for a given question. The following outlines the initial responses upon receiving a question. Before training, the MAS invokes the "browser" tool, but after training, it uses the "reasoner" tool. In this particular case, where logical derivation is required, the "reasoner" tool is the more appropriate choice.

For the whole answering process, before training, the main agent in the MAS does not invoke the "reasoner" tool at all, instead calling the sub-agent twice to handle searching and browsing tasks. However, after training, the MAS uses the "reasoner" tool twice and calls the sub-agent only once, primarily to verify basic information about the Rubik's cube. This example demonstrates how the training process helps the system choose the most suitable tool for a given task.

Thinking before training

role: assistant
content: <thinking> To solve this puzzle, I need to determine the two colors on the removed cube. Let's break down the information:

1. A standard Rubik's cube has 27 smaller cubes: 6 center cubes (1 color), 12 edge cubes (2 colors), 8 corner cubes (3 colors), and 1 central cube (no color).
2. The removed cube has 2 colors (it's an edge cube).
3. All blue cubes have been found.
4. All cubes adjacent to the orange center cube (left, right, above, below) have been found, along with the orange center cube.
5. All green corner cubes have been found, and all green pieces that border yellow have been found.

Given these constraints, I need to:

- Identify which edge cubes could be missing.

Thinking process after training

role: assistant
content: <thinking> This is a complex logic puzzle about a Rubik's cube. I need to carefully analyze the information given and use logical reasoning to determine which cube was removed and what colors are on its faces.
Let me break down what I know:

1. A standard Rubik's cube has 27 smaller cubes total (3x3x3).
2. The cube has been broken apart and one cube is missing.
3. There are 6 cubes with one colored face (center pieces).
4. 12 edge cubes with two colored faces.
5. 8 corner cubes with three colored faces.
6. Total: 6 + 12 + 8 = 26 cubes, so one is missing (which matches the puzzle).

Given information:

- All blue cubes have been found.

- Use the constraints to eliminate possibilities.
- Deduce the two colors of the removed cube.

I'll start by searching for the standard color configuration of a Rubik's cube and the position of the center cubes to understand the relationships between colors. </thinking>
<tool_call>{"name": "browser", "arguments": {"task": "standard Rubik's cube color arrangement center faces", "context": ""}}</tool_call>

- All cubes directly left, right, above, and below the orange center cube have been found....

I need to use logical reasoning to determine which cube is missing and what colors it has. Let me start by analyzing the implications of each clue. </thinking>
<tool_call>{"name": "reasoner", "arguments": {"problem": "A standard Rubik's cube has 27 smaller cubes total ... What are the two colors on the missing cube? Answer with colors in alphabetical order separated by commas."}</tool_call>

A.2 Split and distribute tasks

An important ability that the main agent learns from training with M-GRPO is to properly distribute tasks for sub agent. This progress greatly improves the performance of the sub agent and the results from tools. An example is shown below.

Question

I'm researching species that became invasive after people who kept them as pets released them. There's a certain species of fish that was popularized as a pet by being the main character of the movie Finding Nemo. According to the USGS, where was this fish found as a nonnative species, before the year 2020? I need the answer formatted as the five-digit zip codes of the places the species was found, separated by commas if there is more than one place.

The correct answer to the question is "34689". Before training, the MAS produces an incorrect prediction of "33755". After training, however, it returns the correct answer. The comparison of reasoning processes reveals that the incorrect prediction is primarily caused by ambiguous task assignment to the sub-agent.

Before training, the sub-agent's search task fails to include a critical detail from the question: "became invasive after being released by pet owners." Instead, it retrieves general information about the invasive species Ocellaris Clownfish, which is not directly relevant. In contrast, after training, the main agent explicitly instructs the sub-agent to search for species that "became invasive after being released by pet owners" and provides a clear search scope, including references to "USGS data" and "found as nonnative before 2020."

This case demonstrates that the main agent develops stronger reasoning capabilities after training. It can identify missing but essential information in the question and assign more accurate and context-aware tasks to sub-agents.

Thinking before training

role: assistant
content: <thinking> I need to identify the species of fish from the movie "Finding Nemo" that became invasive after being released by pet owners. The fish is the Clownfish, specifically the Ocellaris Clownfish (Amphiprion ocellaris). I will search for information from the USGS (United States Geological Survey) on where this species was found as a nonnative species before 2020, specifically focusing on five-digit zip codes. </thinking>
<tool_call>{"name": "browser", "arguments": {"task": "Find USGS data on nonnative Ocellaris Clownfish (Amphiprion ocellaris) occurrences in the USA before 2020, including zip codes.", "con-

Thinking process after training

role: assistant
content: <thinking>
I need to research information about the fish species from "Finding Nemo" that became invasive after being released by pet owners. The question specifically asks for:

1. The fish species that was popularized by "Finding Nemo".
2. Where this species was found as a nonnative species before 2020 according to USGS.
3. The answer should be formatted as five-digit zip codes.

Let me start by researching the fish species from Finding Nemo and its invasive status. </thinking>
<tool_call>{"name": "browser", "arguments": {"task": "Search for information about the fish

text": "Ocellaris Clownfish, popularized by the movie Finding Nemo, has been reported as an invasive species in the USA."}}</tool_call>

species from Finding Nemo that became invasive after being released by pet owners, specifically looking for USGS data on where this species was found as nonnative before 2020" }}</tool_call>