Bridging the Capability Gap: Joint Alignment Tuning for Harmonizing LLM-based Multi-Agent Systems

Anonymous ACL submission

Abstract

The advancement of large language models (LLMs) has enabled the construction of multi-agent systems to solve complex tasks by dividing responsibilities among specialized agents, such as a planning agent for subgoal generation and a grounding agent for executing tool-use actions. Most existing methods typically fine-tune these agents independently, leading to capability gaps among them with poor coordination. To address this, we propose MOAT, a Multi-Agent Joint Alignment Tuning framework that improves agents collaboration through iterative alignment. MOAT alternates between two key stages: (1) Planning Agent Alignment, which optimizes the planning agent to generate subgoal sequences that better guide the grounding agent; and (2) Grounding Agent Improving, which fine-tunes the grounding agent using diverse subgoal-action pairs generated by the agent itself to enhance its generalization capability. Theoretical analysis proves that MOAT ensures a non-decreasing and progressively convergent training process. Experiments across six benchmarks demonstrate that MOAT outperforms state-of-the-art baselines, achieving average improvements of 3.1% on held-in tasks and 4.4% on held-out tasks. [1]

1 Introduction

The rapid advancement of large language models (LLMs) has significantly transformed the development of intelligent agents capable of reasoning, decision-making, and interacting with complex environments (Sumers et al., 2024; Song et al., 2023a; Chase, 2022; Song et al., 2023b). Previous work typically involves prompting or fine-tuning a single foundation model on a specific dataset, training the LLMs how to use external search engines for information retrieval or call Web APIs for tasks like travel planning (Qin et al., 2024; Xie et al., 2024). Recently, to enable LLM-based agents to handle more real-world and multi-step tasks, more and more research has increasingly focused on multi-agent systems (Yin et al., 2024a; Qiao et al.; Shen et al., 2024; Wang et al., 2024), which aim to synergize functionality-specialized agents. Figure 1 illustrates a commonly-used pipeline, where a multi-agent system typically includes a planning agent that decomposes the task into subgoals, followed by a grounding agent that executes these subgoals by invoking appropriate tools, ultimately producing the final solution.

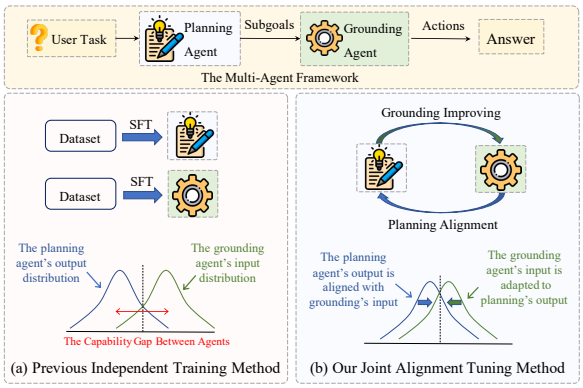

Figure 1: Comparison between (a) previous independent training method and (b) our joint alignment tuning method MOAT. The MOAT performs iterative joint tuning to align agent capabilities and improve coordination.

Despite the progress made by existing multi-agent systems, effectively aligning different agents toward holistic performance remains an active research challenge. Most existing methods construct specific training data for each agent and train each agent independently. While this decomposition can enhance overall performance, it does not guarantee effective collaboration among agents. As illustrated in Figure 1(a), independently trained agents often exhibit varying levels of proficiency, leading to capability mismatches. For example, a planning agent might generate high-level subgoals that are difficult for a weaker grounding agent to under-

[1] Code is available on Anonymous GitHub.

stand and execute. Conversely, a strong grounding agent might struggle with subgoals generated by a weaker planning agent, leading to errors or inefficient task execution. Without explicit mechanisms for adapting to each other's behaviors, these agents struggle to collaborate effectively, resulting in misaligned interactions and coordination failures.

To address the above challenges, we propose MOAT, a **M**ulti-agent **J**o**i**nt **A**lignment **T**uning framework that iteratively alternates between two key stages to achieve alignment in a multi-agent system: (i) **Planning Agent Alignment**, and (ii) **Grounding Agent Improving**. Unlike prior works that train each agent independently, MOAT performs multi-agent joint alignment tuning by iteratively and coordinately optimizing the planning and grounding agents.

Specifically, in the Planning Agent Alignment stage, MOAT optimizes the planning agent to generate subgoals that better guide the grounding agent in producing correct tool-calling actions. Given an input task, we first sample multiple candidate sequences of subgoals from the planning agent. For each subgoal sequence, we then evaluate its effectiveness by measuring the perplexity of the grounding agent in generating correct tool calls conditioned on each sequence. Perplexity reflects how well the grounding agent can follow a subgoal, where lower perplexity indicates higher suitability. Using this as a reward signal, we apply the direct preference optimization (DPO) algorithm (Rafailov et al., 2024) to train the planning agent to align with the grounding agent's preferences. In the Grounding Agent Improving stage, we aim to enhance the grounding agent's ability to interpret and act upon subgoals produced by the planning agent. Specifically, we reuse the subgoal sequences from planning agent in the first stage as training data, exposing the grounding agent to realistic settings. For each input task, we use the ubgoal–action pairs to train the grounding agent via standard language modeling loss. Comparing with relying on ground truth or manually curated subgoal inputs, this allows the grounding agent to adapt to the distribution of subgoals it will receive from the planning agent at practical inference time, thereby improving its robustness and execution accuracy.

Through theoretical analysis, we demonstrate that the holistic performance of the multi-agent system is improved progressively by alternating the above two stages. We apply MOAT to several open-source model families (Llama, Mistral, and Qwen) and evaluate it on three types of tasks, i.e., : Web, Math, and QA, across six benchmarks. The results show that MOAT consistently outperforms existing baselines, on both in-distribution training sets and out-of-distribution test sets. These validate the effectiveness of our joint alignment framework and demonstrate its strong generalization ability.

Our main contributions are as follows: (i) We introduce MOAT, a Multi-Agent Joint Alignment Tuning framework to jointly optimize interconnected agents, bridging the capability gap between them; (ii) We provide formal analysis proving that the alternating optimization of planning and grounding agents guarantees non-decreasing performance and convergence; and (iii) Experiments on both held-in and held-out settings across six benchmarks demonstrate that MOAT achieves the best performance with 4.4% improvement.

2 Related work

LLM-based multi-agent system. Large language models (LLMs) have enabled the development of autonomous agents capable of reasoning, planning, tool use, and memory retention to solve specific goals through self-directed interaction and decision-making (Liu et al., 2024; Madaan et al., 2024). These agents have demonstrated strong capabilities across various complex tasks, such as web navigation (Yao et al., 2022; Zhou et al., 2023), task planning (Zhang et al., 2024b), and tool learning (Shi et al., 2024). While single-agent frameworks like AutoGPT (Yang et al., 2023), XAgent (Team, 2023), and LangChain (Chase, 2022) address such tasks by equipping a single LLM agent with external tools and functions, recent work has explored multi-agent systems that improve problem-solving efficiency through collaborative interaction among multiple agents. For example, CAMEL (Li et al., 2023), AutoGen (Wu et al., 2024), MetaGPT (Hong et al., 2024), and ChatEval (Chan et al., 2024) employ role-playing and structured dialogues to improve task-solving efficiency. However, these systems typically rely on closed-source models, limiting their transparency and practical deployment in privacy scenarios.

Agent tuning. Agent tuning improves a model's ability to perform downstream tasks by fine-tuning open-source LLMs using trajectories distilled from stronger models (Song et al., 2024; Chen et al., 2023) For example, approaches such as AgentTuning (Zeng et al., 2024), and AgentOhana (Zhang

et al., 2024a) fine-tune smaller models on datasets generated by GPT-series LLMs. While these improve instruction following and reasoning, single-agent tuning remains limited for complex tasks requiring long-term planning and execution (Liu et al., 2024). To overcome these limitations, frameworks like Lumos (Yin et al., 2024a) and α-UMi (Shen et al., 2024) propose multi-agent training methods that enable collaboration across functionality-specialized agents. More recent work like AutoACT (Qiao et al.) further advances this direction by introducing a self-training process where each agent is trained on a dataset generated by itself. However, existing methods often train agents independently, lacking joint optimization to ensure effective coordination. In contrast, our work performs iterative joint tuning to align agents' capabilities for improved cooperation.

3 Task Preliminary

A multi-agent system typically consists of three components: (1) a planning agent that breaks down tasks into subgoals, (2) a grounding agent that converts subgoals into executable actions, and (3) an execution module that carries out the actions to get the final answer. Given a complex task x, the planning agent, denoted as π_p is tasked to decompose it to a sequence of subgoals, formulated as $\boldsymbol{s} = \pi_{\mathrm{p}}(x) = \{s_i \mid i \in [|\boldsymbol{s}|]\}$. Each s_i represents a subgoal like "`Calculate the total number of units in the entire building`", contributing to solving the overall task x. Next, the grounding agent, denoted as π_g, takes the task x, the set of available tools I as well as the decomposed subgoals $\boldsymbol{s}$ as input to generates a sequence of tool calls $\boldsymbol{a} = \pi_{\mathrm{g}}(x, I, \boldsymbol{s}) = \{a_i \mid i \in [|\boldsymbol{a}|]\}$. Each $a_i \in I$ represents an individual tool invocation required to complete the subgoal $\boldsymbol{s}_i$, such as "`R1 = Calculator(15 * 8)`". Finally, the execution module is responsible for executing the generated tool-call sequence $\boldsymbol{a}$ to accomplish the user task x.

4 Multi-agent Joint Alignment Tuning

The proposed **multi-agent joint alignment tuning** (MOAT framework aims to iteratively align the planning and grounding agents, enhancing the overall performance and coordination of the multi-agent system. As illustrated in Figure 2, MOAT alternates between two stages: (1) **Planning Agent Alignment**, where the planning agent explores diverse subgoal sequences to guide the grounding agent more effectively, and (2) **Grounding Agent Improving**, we reuse the generated sub-goal sequences, improving the grounding agent to better understand them and generate correct actions. Through this iterative process, both agents progressively adapt to each other, resulting in more coherent subgoal generation, more accurate tool calling, and holistic improvement. Before applying our joint alignment strategy, we perform **initial tuning** to equip both agents with basic task-solving capabilities, following previous work (Yuan et al., 2024; Zhu et al., 2024; Su et al., 2025).

4.1 Initial Tuning

To equip the model with the basic skills to understand user instructions and solve tasks, we first conduct an initial tuning using the SFT dataset collected in previous work (Yin et al., 2024a). Specifically, the planning agent is trained to generate the correct subgoals $\boldsymbol{s}$ for an input task x, which is formulated as:

$$\mathcal{L}_{\mathrm{p}} = -\sum\nolimits_{i=1}^{|\boldsymbol{s}|} \log P_{\pi_{\mathrm{p}}}(s_i \mid s_{<i}, x), \qquad (1)$$

The grounding agent is optimized to ground the subgoals $\boldsymbol{s}$ to the corresponding tool-calling actions $\boldsymbol{a}$, formulated as:

$$\mathcal{L}_{\mathrm{g}} = -\sum\nolimits_{i=1}^{|\boldsymbol{a}|} \log P_{\pi_{\mathrm{g}}}(a_i \mid a_{<i}; x, I, \boldsymbol{s}), \qquad (2)$$

where I is the list of external tools. The final answer is obtained by executing the tool-callings $\boldsymbol{a}$.

4.2 Planning Agent Alignment

LLMs encode strong reasoning abilities in their parameter space, enabling them to generate diverse and meaningful subgoal sequences through sampling (Hou et al., 2025). However, not all sampled subgoal sequences are equally effective—some better guide the grounding agent to generate correct tool-use actions. To exploit this potential, we sample multiple subgoal candidates and optimize the planning agent to prefer those that lead to better grounding outcomes.

Given a task x, we sample K candidate subgoal sequences from the planning agent as $\boldsymbol{s} = \pi_{\mathrm{p}}(x)$ and obtain a set $\mathcal{S} = \{\boldsymbol{s}_j \mid j \in [|K|]\}$. For each $\boldsymbol{s} \in \mathcal{S}$, we calculate its perplexity (PPL) with respect to the grounding agent, where a lower perplexity indicates that the subgoal sequence is more helpful to the grounding agent, facilitating the generation of correct responses. Therefore, the PPL

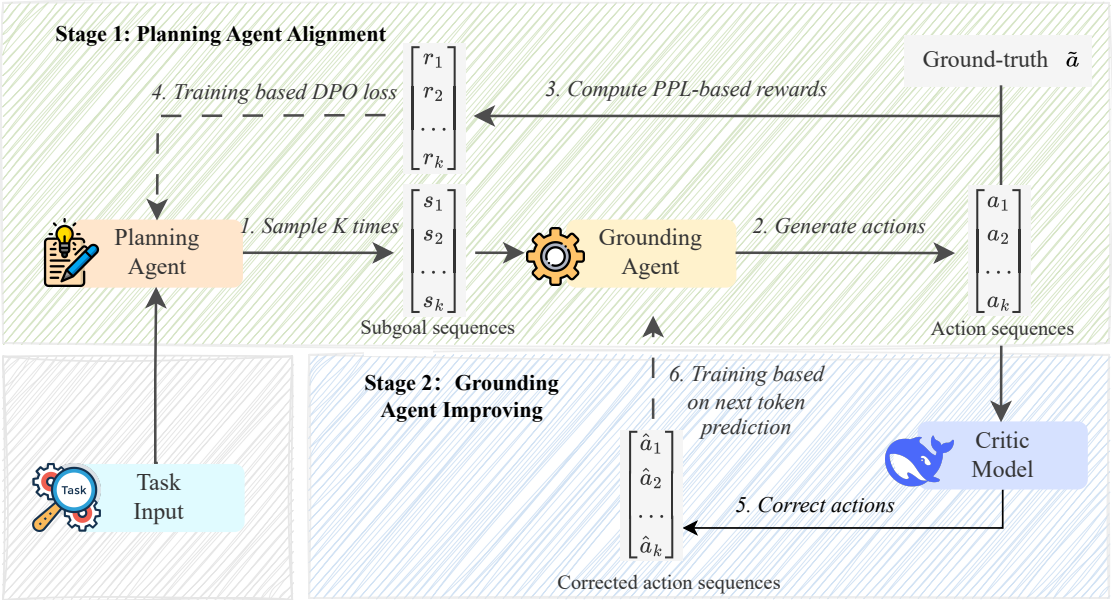

Figure 2: The proposed MOAT framework iteratively alternates between two stages: (1) Planning Agent Alignment: The planning agent samples K candidate subgoal sequences, and the grounding agent generates corresponding tool-calling actions. Subgoal sequences are ranked by PPL, and the planning agent is optimized via DPO. (2) Grounding Agent Improving: The subgoal-action pairs generated are corrected using a critic model, and the grounding agent is fine-tuned on the corrected dataset to enhance generalization.

can directly reflect how $\boldsymbol{s}$ is useful to the end-to-end task performance, which is formulated as follows:

$$\mathrm{PPL}_{\pi_\mathrm{g}}(\boldsymbol{a} \mid x, I, \boldsymbol{s}) := \exp\Big\{-\frac{1}{|\boldsymbol{a}|}\sum_{i=1}^{|\boldsymbol{a}|}\log P_{\pi_\mathrm{g}}(\boldsymbol{s} \mid \boldsymbol{a}_{<i}, x, I, \boldsymbol{s})\Big\}.$$

To align the planning agent with holistic task-solving performance, we train the planning agent to generate subgoal sequences with lower PPL, as desirable behaviors, while penalizing undesirable ones, i.e., subgoal sequences with high PPL. Specifically, we adopt the Direct Preference Optimization (DPO) algorithm (Rafailov et al., 2024), which allows us to align the model by learning from preference pairs. Specifically, we construct preference pairs $(\boldsymbol{s}_w, \boldsymbol{s}_l)$ from the sampled candidates in $\mathcal{S}$, where $\boldsymbol{s}_w$ yields the lowest perplexity (strongest grounding guidance) and $\boldsymbol{s}_l$ the highest. The corresponding loss function is formulated as follows:

$$\mathcal{L}_\mathrm{DPO} = -\mathbb{E}_{(t,(\boldsymbol{s}_w,\boldsymbol{s}_l))\sim D}\left[\log\sigma\left(\beta\log\frac{\pi_\mathrm{p}(\boldsymbol{s}_w|x)}{\pi_\mathrm{ref}(\boldsymbol{s}_w|x)} - \beta\log\frac{\pi_\mathrm{p}(\boldsymbol{s}_l|x)}{\pi_\mathrm{ref}(\boldsymbol{s}_l|x)}\right)\right],$$

where π_ref represents the reference model, which is initialized as the original π_p before optimization. And σ denotes the sigmoid function, β is a hyperparameter.

4.3 Grounding Agent Improving

This stage aims to enhance the generalization capability of the grounding agent and improve its adaptability to the diverse subgoal sequences generated by the planning agent. To achieve this, we reuse the diverse subgoal sequences sampled from the first stage as inputs, and prompt the grounding agent to generate outputs to fine-tune itself. Specifically, given a task x, for each subgoal sequence $\boldsymbol{s}$ from the sampled set $\mathcal{S} = \{\boldsymbol{s}_j \mid j \in [|K|]\}$, the grounding agent π_g generates the corresponding tool-calling actions as $\boldsymbol{a}_j = \pi_\mathrm{g}(x, I, \boldsymbol{s}_j)$.

However, since these action sequences are model-generated, they may contain errors. Directly using such noisy data for fine-tuning can lead to performance degradation or even training collapse, as highlighted in prior work (Dohmatob et al., 2024; Shumailov et al., 2024). To address this issue, we introduce a validation mechanism that filters out incorrect outputs before using them for training. In particular, we employ a more powerful LLM as a critic model to evaluate whether each generated sequence $\boldsymbol{a}$,successfully solves the task , given the input $(x, I, \boldsymbol{s})$. If the critic determines that the sequence fails to complete the task, it provides a corrected version $\hat{\boldsymbol{a}}$ by referencing the ground-truth outcome $\tilde{\boldsymbol{a}}$. This filtering and correction process

Algorithm 1: Dataset Construction

Initialize SFT dataset $\mathcal{D}_{\mathrm{g}} \leftarrow \emptyset$;
for *each task* x *and* $s \in \mathcal{S}$ **do**
  Generate $\boldsymbol{a} \leftarrow \pi_{\mathrm{g}}(x, I, \boldsymbol{s})$;
  if *Critic*$(\langle x, I, \boldsymbol{s}\rangle, \boldsymbol{a}, \tilde{\boldsymbol{a}}) = \mathrm{False}$ **then**
    $\hat{\boldsymbol{a}} = \mathrm{Critic}(\langle x, I, \boldsymbol{s}\rangle, \boldsymbol{a}, \tilde{\boldsymbol{a}})$;
    $\mathcal{D}_{\mathrm{g}} \leftarrow \mathcal{D}_{\mathrm{g}} \cup \{(x, I, \boldsymbol{s}), \hat{\boldsymbol{a}}\}$;
  end
  else
    $\mathcal{D}_{\mathrm{g}} \leftarrow \mathcal{D}_{\mathrm{g}} \cup \{(x, I, \boldsymbol{s}), \boldsymbol{a}\}$;
  end
end
return SFT dataset $\mathcal{D}_{\mathrm{g}}$;

ensures that only reliable supervision signals are used during grounding agent training. The full procedure is summarized in Algorithm 1, and the critic prompting strategy is detailed in Appendix A.3.

The overall MOAT alternates between the first and second stages described above. During this process, the planning agent gradually adapts to the grounding agent by generating subgoal sequences that better align with its inference process; the grounding agent, in turns, improves its generalization capability to understand the subgoals of the planning agent. This formulates a loop for a consistent improvement. We also provide a detailed pseudo algorithm in Algorithm 2 to further clarify our joint training process.

5 Theoretical analysis

In our framework, the planning agent and grounding agent are optimized iteratively. In this section, we provide a theoretical analysis to demonstrate that each optimization step leads to non-decreasing improvements and ultimately ensures the convergence. We start by defining the expected performance of the overall multi-agent system as:

$$\mathbb{E}[R] = \mathbb{E}_{\boldsymbol{s}\sim\pi_{\mathrm{p}}(x)}\left[\mathbb{E}_{\boldsymbol{a}\sim\pi_{\mathrm{g}}(s)}[R(\boldsymbol{s}, \boldsymbol{a})]\right]. \quad (3)$$

Here the reward function $R(\boldsymbol{s}, \boldsymbol{a})$ evaluate the quality of tool-calling action $\boldsymbol{a}$ given sub-goal sequence $\boldsymbol{s}$. And x indicates the input task. Below, we can state the following two lemmas.

Lemma 5.1. *Optimizing the planning agent while keeping the grounding agent fixed leads to a non-decreasing expected reward.*

The planning agent is optimized using DPO, with PPL as the reward signal. The optimization objective can be formalized as:

$$\max_{\pi_{\mathrm{p}}} \mathbb{E}_{\boldsymbol{a}\sim\pi_{\mathrm{p}}(x)}\left[-\mathrm{PPL}(\boldsymbol{a}; \pi_{\mathrm{g}})\right]. \quad (4)$$

Since PPL is negatively correlated with the true reward $R(\boldsymbol{s}, \boldsymbol{a})$, this is equivalent to maximizing the expected reward:

$$\max_{\pi_{\mathrm{p}}} \mathbb{E}_{\boldsymbol{s}\sim\pi_{\mathrm{p}}(x)}\left[R(\boldsymbol{s}, \boldsymbol{a})\right]. \quad (5)$$

The DPO algorithm guarantees that updates to π_{p} lead to non-decreasing expected rewards when the grounding agent is fixed. Thus, we have:

$$\mathbb{E}[R]^{(t+1)} \geq \mathbb{E}[R]^{(t)}. \quad (6)$$

This inequality holds because the optimization process aligns the planning agent with sub-goal sequences that facilitate better performance in the grounding agent.

Lemma 5.2. *Optimizing the grounding agent while keeping the planning agent fixed leads to a non-decreasing expected reward.*

The grounding agent is optimized through supervised fine-tuning using pairs $(\boldsymbol{s}, \boldsymbol{a})$ generated by the planning agent. The corresponding optimization objective is:

$$\min_{\pi_{\mathrm{g}}} \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\mathcal{S}}\left[\mathcal{L}(\pi_{\mathrm{g}}(\boldsymbol{a} \mid \boldsymbol{s}))\right], \quad (7)$$

where $\mathcal{L}$ denotes the loss function (e.g., cross-entropy loss). Minimizing this loss is equivalent to maximizing the log-likelihood of the correct tool invocation sequences:

$$\max_{\pi_{\mathrm{g}}} \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a})\sim\mathcal{S}}\left[\log \pi_{\mathrm{g}}(\boldsymbol{a}|\boldsymbol{s})\right]. \quad (8)$$

Since improved log-likelihood corresponds to reduced PPL and, consequently, higher reward (Singh et al., 2023), it follows that:

$$\mathbb{E}[R]^{(t+1)} \geq \mathbb{E}[R]^{(t)}. \quad (9)$$

Hence, optimizing the grounding agent improves or maintains the expected reward when the planning agent is fixed.

From Lemma 5.1 and Lemma 5.2, we establish that both optimization steps ensure non-decreasing expected rewards, i.e., $\mathbb{E}[R]^{(t+1)} \geq \mathbb{E}[R]^{(t)}$. Additionally, the expected reward $\mathbb{E}[R]$ is upper-bounded due to the following reasons: (i) The reward function $R(\boldsymbol{s}, \boldsymbol{a})$ is bounded in practical scenarios; and (ii) The PPL has a lower bound. Based on the *Monotone Convergence Theorem* (Bibby, 1974), the non-decreasing and upper-bounded nature of $\{\mathbb{E}[R]^{(t)}\}_{t=1}^{\infty}$ ensures this sequence converges to a finite limit. Thus, we derive the convergence of overall training process.

Task	Skill Dim.	#Inst.	Metric
Held-in Tasks			
StrategyQA (Yang et al., 2018)	QA	300	Exact Match
GSM8K (Cobbe et al., 2021)	Math	1300	Accuracy
Mind2Web (Deng et al., 2023)	Web	200	Step Success Rate
Held-out Tasks			
HotpotQA (Geva et al., 2021)	QA	100	Exact Match
SVAMP (Patel et al., 2021)	Math	1000	Accuracy
WebShop (Yao et al., 2022)	Web	500	Avg. Reward

Table 1: The held-in and held-out tasks used to evaluate the agent capabilities of different LLMs.

6 Experimental Setup

6.1 Benchmarks and Evaluation Metrics

Following prior work (Song et al., 2024; Chen et al., 2024), we evaluate MOAT under both held-in and held-out settings to evaluate its performance and generalization across diverse task types. We consider a wide range of tasks, including mathematical reasoning, web interaction, and question answering. As listed in Table 1, the held-in setting includes three tasks that are used during training: GSM8K, StrategyQA, and Mind2Web; the held-out setting evaluates generalization on unseen tasks: SVAMP, WebShop, and HotpotQA. Evaluation metrics for each task are also reported in Table 1. Following the recipe of baselins (Yin et al., 2024a), we define a set of action instructions (i.e., tool set I), covering common actions required for each task. Details are provided in Appendix A.4.

6.2 Baselines

We compare our MOAT with widely-used agent tuning methods, including: (i) **Agent Tuning** (Zeng et al., 2024), a multi-task tuning approach training LLMs on synthetic datasets comprising six tasks; (ii) **Agent-FLAN** (Chen et al., 2024) employs a modular architecture that trains distinct single-agent capabilities through specialized parameter groups; and (iii) **Agent Lumos** (Yin et al., 2024a), a multi-agent training framework that separately fine-tunes models on datasets to obtain specialized agents. Furthermore, we included GPT-3.5-Turbo and GPT-4 (Achiam et al., 2023) as strong single-agent baselines for comparison.

6.3 Implementation Details

To ensure a fair comparison with prior work, we adopt `Llama2-7b-hf` as the backbone LLM for both MOAT and baseline methods, following the official implementation of previous methods (Zeng et al., 2024; Yin et al., 2024a). To comprehensively evaluate our method across different LLMs, we additionally apply MOAT to two different model series with varying parameter scales: `Mistral-7B-Instruct-v0.2` and `Qwen2.5-14B`. During the alignment process, we set the number of sampled subgoal sequences K to 15 and the number of training iterations to 2. The sampling temperature is set to 1.0 to encourage diversity in the generated subgoals. We employ `DeepSeek-R1-Distill-Qwen-32B` as the critic model (denoted as `DS-Qwen-32B`) for verifying and correcting the generated tool-use action sequences. We further analyze the impact of using different critic models in Section 7.4. More implementation details are provided in Appendix A.1.

7 Experiment results

7.1 Overall Performance

Held-in Tasks. Table 2 presents the evaluation results. Compared with single-agent systems and independently trained multi-agent baselines, the MOAT achieves superior performance across three held-in tasks across different base models. The MOAT with Llama-7B demonstrates an average improvement of 15.6% compared to AgentTuning with `Llama-13B`. These improvements validate the effectiveness of our joint training framework, which tightly interconnects specialized agents to enhance overall task-solving performance.

Held-out Tasks. We further investigate the generalizability of our method in solving unseen tasks. As illustrated in Table 2, our method achieves the highest performance compared to open-source baselines. For example, the MOAT with `Mistral-7B` outperforms Lumos with an average performance improvement of 4.4%. An explanation for this improvement is that through iterative alignment in MOAT, the subgoals generated by the planning model align better with the preferences of the grounding models, and the grounding models also achieve a more accurate understanding of the generated subgoals. This mutual understanding enhances the generalizability of the overall system when facing unseen tasks.

Comparison with Closed-source Agents. Although our method is trained on 7B models like `Llama-7B`, it achieves about a 50% performance improvement over GPT-4 on the Mind2Web task. This further validates the superiority of the MOAT in synergizing smaller open-source models to achieve competitive performance.

Method	Base Model	Held-in Tasks				Held-out Tasks			
		GSK8K	Mind2Web	StrategyQA	Avg.	SVAMP	WebShop	HotpotQA	Avg.
API-Based Agents									
GPT-4	-	87.0	22.6	71.0	60.2	90.5	58.6	52.1	67.1
GPT-3.5-Turbo	-	65.0	21.7	58.0	48.2	81.0	62.4	24.0	55.8
Llama Model Agents									
Llama-2-7B-Chat	Llama-2-7B	15.0	11.9	5.0	10.6	20.7	15.8	3.0	13.2
Agent Tuning	Llama-2-7B	14.0	10.6	49.0	24.5	35.3	59.8	10.0	35.0
Agent Tuning	Llama-2-13B	22.3	11.1	**52.0**	28.5	56.9	**65.0**	24.0	48.6
Agent-FLAN	Llama-2-7B	28.5	16.9	48.0	31.1	39.2	55.9	12.0	35.7
Agent Lumos	Llama-2-7B	46.6	29.9	46.7	41.1	65.5	58.3	25.0	49.6
MOAT	Llama-2-7B	**47.4**	**33.0**	**52.0**	**44.1**	**69.2**	60.6	**27.0**	**52.3**
Mistral Model Agents									
Agent Lumos	Mistral-7B-v0.2	46.4	33.8	49.3	43.2	61.9	58.7	27.0	49.2
MOAT	Mistral-7B-v0.2	**48.2**	**34.7**	**56.0**	**46.3**	**73.7**	**59.0**	**28.0**	**53.6**
Qwen Model Agents									
Agent Lumos	Qwen2.5-14B	81.7	31.8	49.3	54.3	85.5	64.7	27.0	59.1
MOAT	Qwen2.5-14B	**82.4**	**32.6**	**55.3**	**56.8**	**87.4**	**65.8**	**28.0**	**60.4**

Table 2: Evaluation results of MOAT and baselines on both held-in and held-out tasks. The best results in each group are highlighted in **bold**.

Method	Mind2Web	WebShop	Avg. Δ
MOAT	35.43	60.63	48.03
-w/o stage 1	$31.94_{\downarrow 3.49}$	$58.76_{\downarrow 1.87}$	$45.35_{\downarrow 2.68}$
-w/o stage 2	$34.72_{\downarrow 0.71}$	$60.29_{\downarrow 0.34}$	$47.51_{\downarrow 0.52}$
-w/o critic	$33.88_{\downarrow 1.55}$	$59.79_{\downarrow 0.84}$	$46.84_{\downarrow 1.19}$

Table 3: Ablation study on two web datasets.

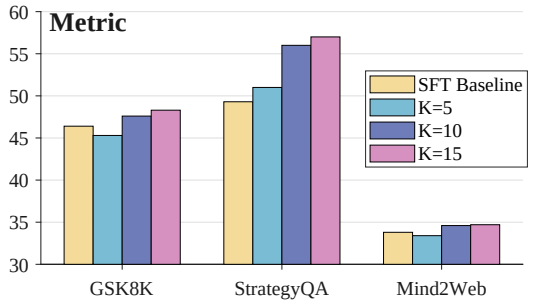

Figure 3: Results of MOAT on three held-in tasks under different numbers of sampled subgoal sequences.

7.2 Ablation Study

To further analyze the contribution of each component in MOAT, we conduct an ablation study by removing individual components, including planning alignment (*w/o stage 1*), grounding improvement (*w/o stage 2*), and the critic model (*w/o critic*), respectively. As shown in Table 3, all variants exhibit substantial performance degradation, confirming the effectiveness of each component in our joint alignment framework. Besides, we highlight **two key points**: (1) the largest performance drop occurs in *w/o stage 1*, highlighting the critical role of aligning the planning agent to generate coherent subgoals; and (2) removing the critic model (*w/o critic*) results in the second-largest performance drop, even lower than that caused by removing the grounding improvement (*w/o stage 2*). This suggests that, without external feedback from the critic model, the system may suffer from significant negative updates, thereby validating the importance and rationality of incorporating a critic model.

7.3 Hyperparameter Analysis

Analysis of Different Sample Numbers. In our main experiments, we set the number of sampled responses K to 15. To explore the impact of the sampling number K on model performance, we further vary K from 5 to 15 during the training of `Mistral-7B` at $2th$ iteration. As shown in Figure 3, we observe a positive correlation between the sampling number and the overall performance. We also identify a performance drop on the GSM8K and Mind2Web benchmarks when K=5. An explanation is that a smaller number of samples may fail to include high-quality subgoal sequences that align well with the grounding agent. In such cases, even the subgoal sequence with the highest reward may still be suboptimal or incorrect, thus negatively affecting training performance.

Analysis of Iteration Count. We further investigate how the iteration count impacts model performance using `Mistral-7B` with set K to 15. As shown in Figure 4, the model's performance improves gradually with the increasing number of iterations. However, by the third iteration, the per-

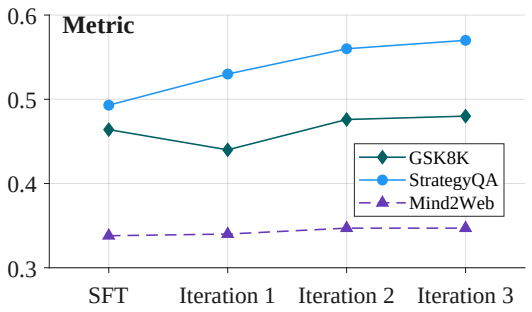

Figure 4: Performance trends of MOAT (K=15) on held-in tasks as iterations increase.

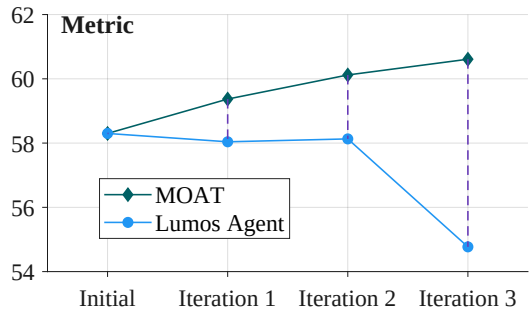

Figure 5: Performance comparison on WebShop between MOAT and Lumos under equal training time.

Model	Mind2Web	WebShop	Avg.
MOAT *w/ DS-Qwen-32B*	34.03	**60.78**	47.41
MOAT *w/ GPT-4o*	**35.28**$_{\uparrow 3.67\%}$	60.57$_{\downarrow 0.03\%}$	**47.93**$_{\uparrow 1.08\%}$
MOAT *w/ DS-Qwen-14B*	33.52$_{\downarrow 1.50\%}$	60.63$_{\downarrow 0.02\%}$	47.08$_{\downarrow .07\%}$

Table 4: Model Performance on Mind2Web and WebShop benchmarks using different critic models.

formance gains become marginal. We suspect this is because, after several iterations, the planning and grounding agents gradually converge and reach a performance equilibrium, as discussed in Section 5.

7.4 Impact of Different Critic Model.

We use `DeepSeek-R1-Distill-Qwen-32B` as the default critic model in our framework to validate and refine the tool-use action sequences generated by the grounding agent. To investigate the effect of the critic model's capability, we conduct a comparative study using a stronger critic (GPT-4o) and a weaker one (`DeepSeek-R1-Distill-Qwen-14B`). As shown in Table 4, the results demonstrate an upward trend in task performance as the ability of the critic model increases. However, we also observe that using a smaller, open-source model like `Qwen-14B` still yields competitive results, surpassing existing baselines by a notable margin. We attribute this to the relatively simple nature of the critic's task, i.e., verifying whether the predicted action sequence achieves the same effect as the ground-truth. Since both the prediction and reference are provided to the context of the critic model, this task requires less complex reasoning with simplified difficulty. Therefore, while stronger critic models can further enhance performance, our framework remains robust and effective even when using smaller, fully open-source critics.

7.5 Training Iteration Control Analysis

A potential concern is that the observed performance gains from our iterative training strategy may stem merely from additional training epochs, rather than from the collaborative optimization of planning and grounding agents. To investigate this, we compare our approach with a baseline trained independently for the same total number of epochs. Starting from a model trained for 2 epochs, we apply our iterative method for 1, 2, and 3 iterations, equivalent to 3, 4, and 5 total epochs. We compare with the Lumos Agent baseline trained independently for the same number of epochs without inter-agent interaction. As shown in Figure 5, the baseline struggles to consistently improve and even suffers from degradation due to overfitting. In contrast, our method shows consistent improvements, indicating that the gains stem from iterative co-training rather than extended training iterations.

7.6 Case Study

We manually analyze the outputs of both the planning and grounding agents after training in MOAT. The results show that our MOAT effectively enhances the specialized expertise of both agents, as well as their adaptability. Concrete examples and detailed analysis are provided in Appendix A.2.

8 Conclusion

In this work, we present MOAT, a novel Joint Alignment Tuning framework designed to harmonize the collaboration between planning and grounding agents in LLM-based multi-agent systems. By iteratively optimizing the planning agent to generate subgoals that align with the grounding agent's capabilities and enhancing the grounding agent's adaptability to diverse subgoal sequences, MOAT effectively bridges the capability gap caused by independent training. Both theoretical analysis and extensive experimental results demonstrate the superiority of MOAT. We suggest that future work can explore the integration of visual models to expand the capability boundary of agents, as well as integrate more specialized agents, such as tool retrieval and reflection modules, to expand the versatility and efficiency of the system.

Limitations

Our framework is currently developed and evaluated exclusively on text-based scenarios, without exploring multimodal learning settings. While modern open-source language models (e.g., LLaVA, Qwen-VL) have demonstrated emerging capabilities in processing multimodal inputs, our current architecture lacks explicit mechanisms for cross-modal alignment during collaborative training. In future work, we plan to incorporate multimodal information into our framework.

Ethics Statement

This research strictly adheres to the ethical principles outlined in the ACM Code of Ethics, with rigorous implementation of transparency and accountability measures. All datasets, tools, and language models (including Llama-2, Mistral and Qwen) are sourced from publicly available platforms under compliant licenses, ensuring ethical alignment and reproducibility. The complete code and evaluation protocols will be open-sourced upon publication.

References

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

John Bibby. 1974. Axiomatisations of the average and a further generalisation of monotonic sequences. *Glasgow Mathematical Journal*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better LLM-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*.

Harrison Chase. 2022. LangChain.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9354–9366.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.

Elvis Dohmatob, Yunzhen Feng, and Julia Kempe. 2024. Model collapse demystified: The case of regression. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies. *Transactions of the Association for Computational Linguistics (TACL)*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. 2025. Advancing language model reasoning through reinforcement learning and inference scaling. *arXiv preprint arXiv:2501.11651*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2024. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang,

et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Huajun Chen, et al. Autoact: Automatic agent learning from scratch for qa via self-planning. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

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. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. 2024. Small LLMs are weak tool learners: A multi-LLM agent. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16658–16680.

Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. 2024. Learning to use tools via cooperative and interactive agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. 2024. Ai models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. 2023. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*.

Chan Hee Song, Brian M. Sadler, Jiaman Wu, Wei-Lun Chao, Clayton Washington, and Yu Su. 2023a. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 2986–2997. IEEE.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Agentbank: Towards generalized llm agents via fine-tuning on 50000+ interaction trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2124–2141.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023b. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*.

Zhaochen Su, Linjie Li, Mingyang Song, Yunzhuo Hao, Zhengyuan Yang, Jun Zhang, Guanjie Chen, Jiawei Gu, Juntao Li, Xiaoye Qu, et al. 2025. Openthinkimg: Learning to think with images via visual tool reinforcement learning. *arXiv preprint arXiv:2505.08617*.

Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. 2024. Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

XAgent Team. 2023. Xagent: An autonomous agent for complex task solving.

Yiying Wang, Xiaojing Li, Binzhu Wang, Yueyang Zhou, Yingru Lin, Han Ji, Hong Chen, Jinshi Zhang, Fei Yu, Zewei Zhao, Song Jin, Renji Gong, and Wanqing Xu. 2024. Peer: Expertizing domain-specific tasks with a multi-agent framework and tuning methods. *Preprint*, arXiv:2407.06985.

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. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*.

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024a. Agent lumos: Unified and modular training for open-source language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12380–12403. Association for Computational Linguistics.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024b. Agent lumos: Unified and modular training for open-source language agents. *Preprint*, arXiv:2311.05657.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077. Association for Computational Linguistics.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Ming Zhu, Juntao Tan, Thai Hoang, Zuxin Liu, Liangwei Yang, et al. 2024a. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*.

Jintian Zhang, Xin Xu, Ningyu Zhang, Ruibo Liu, Bryan Hooi, and Shumin Deng. 2024b. Exploring collaboration mechanisms for LLM agents: A social psychology view. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14544–14607.

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. 2023. Webarena: A realistic web environment for building autonomous agents. In *Second Agent Learning in Open-Endedness Workshop*.

Junda Zhu, Lingyong Yan, Haibo Shi, Dawei Yin, and Lei Sha. 2024. Atm: Adversarial tuning multi-agent system makes a robust retrieval-augmented generator. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10902–10919.

Algorithm 2: The proposed multi-agent joint optimization, which iteratively aligns the planning and grounding agents, improving the holistic task-solving performance.

Input: The number of iterations N_r, the number of samples K, the tasks set $\mathcal{D}$, the set of available tools I, the critic model CRITIC , planning agent π_p , grounding agent π_g.

for *iteration t = 1... N* **do**
  # *Initialize the training dataset*
  $\mathcal{D}_p \leftarrow \phi, \mathcal{D}_g \leftarrow \phi\ \mathcal{R} \leftarrow \phi$
  # *Planning Agent Optimization*
  for *each task* $x_i \in \mathcal{D}$ **do**
    # *sample K response*
    for $j \leftarrow 1$ **to** K **do**
      $\boldsymbol{s}_{i,j} \sim \pi_p^{t-1}(x_i)$
      # $\tilde{\boldsymbol{a}}_{i,j}$ *denotes correct actions*
      $\boldsymbol{r}_{i,j} \leftarrow \mathrm{PPL}_{\pi_g^{t-1}}(\tilde{\boldsymbol{a}}_{i,j}|x_i, I, \boldsymbol{s}_{i,j})$
      $\mathcal{D}_{p,i} \leftarrow \mathcal{D}_{p,i} \cup \boldsymbol{s}_{i,j}$
      $\mathcal{R}_{p,i} \leftarrow \mathcal{R}_{p,i} \cup \boldsymbol{r}_{i,j}$
    end
  end
  $\mathcal{S}_{win} \leftarrow \mathcal{D}_p[\mathrm{argmin}(\mathcal{R}, axis = 1)]$
  $\mathcal{S}_{lose} \leftarrow \mathcal{D}_p[\mathrm{argmax}(\mathcal{R}, axis = 1)]$
  $\pi_p^t \leftarrow \pi_p^{t-1} - \nabla_{\pi_p}\mathcal{L}_{\mathrm{DPO}}(\mathcal{D}, \mathcal{S}_{win}, \mathcal{S}_{loss})$
  # *Grounding Agent Optimization*
  for *each task* $x_i \in D$ **do**
    for *each* $\boldsymbol{s}_{i,j} \in \mathcal{D}_p$ **do**
      $\boldsymbol{a}_{i,j} \sim \pi_g^{r-1}(\boldsymbol{a}_{i,j}|x_i, I, \boldsymbol{s}_{i,j})$
      $\hat{\boldsymbol{a}}_{i,j} \leftarrow \mathrm{Critic}(\boldsymbol{a}_{i,j}, \tilde{\boldsymbol{a}}_{i,j})$
      $\mathcal{D}_g \leftarrow \mathcal{D}_g \cup (x, I, \boldsymbol{s}_{i,j}, \hat{\boldsymbol{a}}_{i,j})$
    end
  end
  $\pi_g^t \leftarrow \pi_g^{t-1} - \nabla_{\pi_g}\mathcal{L}_{\mathrm{SFT}}(\mathcal{D}_g)$
end

Output: π_p^N, π_g^N

A Appendix

A.1 Implementation Details

We show more training details about our experiments. All our experiments are conducted on 6 × NVIDIA A800 (80GB) GPUs.

For the initial fine-tuning stage, we use the public datasets provided by Lumos (Yin et al., 2024a) and train two epochs with a learning rate of 2×10^{-5}. And we set the maximum sequence length to 1024 and the batch size to 128. We also apply linear warmup for 3% of the total training steps to adjust the learning rate.

For DPO training, we fine-tuned the model using the `accelerate` framework with DeepSpeed for optimized distributed training. We set batch size to 4 and gradient accumulation to 8. The learning rate is set to 4×10^{-7} with a cosine learning rate scheduler. And We set the maximum sequence length to 1024. Additionally, we leveraged the `TRL` library [2] to facilitate the training of reinforcement learning-based models. Meanwhile, we filter out data samples where the reward difference between s_w and s_l is less than 0.1 for stability during DPO training.

For grounding agent improving training, we implement training over two epochs with a learning rate of 2×10^{-5} and a batch size 128 the same with initial tuning. At the same time, we mix these with the initial data in a 1:1 ratio to prevent the model from forgetting prior knowledge.

A.2 Case Study

As illustrated in Table A.2, the case studies evaluate the responses generated by our MOAT and the independent training method. Our findings show that through joint alignment tuning, the models are able to align their capabilities. Specifically, for the given case, we observe that the independently trained method struggles with subgoal decomposition in planning agent, making it difficult for grounding agent to resolve, leading to a failure in solving the task. However, after the joint alignment training, the capability gap is addressed, allowing the planning agent to generate subgoals that are easier for the grounding agent to understand. Consequently, the grounding agent successfully produced the correct action sequence.

A.3 Prompts

We provide our prompts used in the multi-agent system. The prompt of the planning agent to generate the subgoal sequences is in Table 6. The prompt of the grounding agent to generate the action sequences is in Table 7. The prompt of the critic model to correct the action sequences generated by grounding agents is in Table 8.

[2] https://github.com/huggingface/trl

Case Study

Question:
Richard lives in an apartment building with 15 floors. Each floor contains 8 units, and 3/4 of the building is occupied. What's the total number of unoccupied units In the building?

MOAT:

Subgoals:
Subgoal 1: Calculate the total number of units in the entire building.;
Subgoal 2: Calculate the number of floors that are occupied.;
Subgoal 3: Calculate the total number of units that are occupied.;
Subgoal 4: Calculate the total number of unoccupied units in the building.

Actions:
R1 = Calculator(15 * 8);
R2 = Calculator(15 * 3/4);
R3 = Calculator(R2 * 8);
R4 = Calculator(R1 - R3)

Answer:
30

Independent Training Method:

Subgoals:
Subgoal 1: Calculate the total number of units in the building.;
Subgoal 2: Calculate the number of units that are unoccupied.;
Subgoal 3: Calculate the total number of unoccupied units in the building.

Actions:
R1 = Calculator(15 * 8);
R2 = Calculator(R1 * 3/4);
R3 = Calculator(R2 * 2)

Answer:
30

Table 5: A case study in the GSM8K test dataset.

Prompt to generate subgoal sequences

Please provide a reasonable subgoal-based plan to solve the given task.

Task:{TASK}

Table 6: The prompt of planning agent to generate subgoal sequences.

Prompt to generate action sequences

Please ground the given subgoal to corresponding executable actions for solving the given task. The grounded actions must be the one in available action list.

The available action list is:{ACTION_LIST}

Task:{TASK}

Subgoals to be grounded:{SUBGOALS}

Table 7: The prompt of grounding agent to generate action sequences.

Prompt to correct action sequences

Given a task and a corresponding series of subgoals and their corresponding actions that may be incomplete, your task is to judge whether the subgoals and actions can reached a final answer or conclusion for the problem.
The grounded actions must be the one in available action list.The available action list is {ACTION_LIST}
If the actions can reached a final answer, you should directly output "Final answer reached". Otherwise, you should give corrections to the original subgoals and their corresponding actions. It is not necessary to be similar to the original subgoals and actions.

Task:{TASK}
Reference subgoals: {REF_SUBGOALS}
Reference actions: {REF_ACTIONS}
Judged subgoals: {SUBGOALS}
Judged actions: {ACTIONS}

Your output should follow the format:
If can reached a final answer, directly output "Final answer reached". Else, output corrected subgoals and actions following this format:
Corrected Subgoals: <series of subgoals to complete the task in one line, Each Subgoal begins with Subgoal idx>
Corrected Actions: <corresponding actions in one line>

Table 8: The prompt of critic model to correct action sequences.

A.4 Action Interfaces and Execution Tools for Complex Interactive Tasks

For each defined action in the action interfaces, a corresponding backend execution tool is provided to enable the implementation of that action. Our setup follows the approach described in Yin et al. (2024b). We have adopted the same configuration to ensure comparability between our work and theirs.

As shown in Table 9a, for QA tasks, we use Wikipedia and Google Search APIs to find relevant knowledge about entities. Additionally, we use a semantic matching model, `dpr-reader-multiset-base`[3], employed in Dense Passage Retrieval (DPR) (Karpukhin et al., 2020), to retrieve paragraphs based on the query. Following the approach from ReWOO (Xu et al., 2023), we also utilize GPT-series models as a straightforward QA tool to respond to queries based on the retrieved knowledge or prior interactions.

In Table 9b, web tasks involve real mouse and keyboard operations such as typing, clicking, and selecting HTML tags. To identify the appropriate HTML tags to operate on, we use a DeBERTa model[4] that ranks and retrieves relevant tags based on the current action, as seen in the AgentBench evaluation.

As illustrated in Table 9c, WolframAlpha API [5] serves as the main tool for mathematical tasks, as it is capable of executing a wide range of mathematical functions, including formula computation and equation solving. For more advanced math operations like sorting, we leverage OpenAI Codex (Chen et al., 2021) to generate short code snippets for execution.

For the unseen task WebShop, the actions include `Search`, `FeatureRetrieve`, `Pick`, and `Click`. The `Search` and `Click` actions are implemented using the embedded features provided in the official WebShop virtual environment[6] following (Liu et al., 2024). Meanwhile, `FeatureRetrieve` and `Pick` rely on the `dpr-reader-multiset-base`, which helps select the most relevant items and their features based on the query.

[3] https://huggingface.co/facebook/dpr-reader-multiset-base.

[4] https://huggingface.co/osunlp/MindAct_CandidateGeneration_deberta-v3-base.

[5] https://www.wolframalpha.com/.

[6] https://github.com/princeton-nlp/WebShop.

Task Type	Action Types	Function Descriptions	Tools
QA	`KnowledgeQuery(Entity) -> Knowledge`	Query the entity knowledge	Wikipedia, Google Search
	`ParagraphRetrieval(Knowledge, Query) -> Paragraphs`	Retrieve relevant paragraphs based on the query	`dpr-reader-multiset-base`
	`QA(Context, Query) -> Answer`	Answer the query based on the provided context	GPT-series/open LLMs
	`Calculator(Expression) -> Value`	Calculate given mathematical expressions	WolframAlpha

(a) Actions used in complex QA tasks.

Task Type	Action Types	Function Descriptions	Implementation
Web	`Click(Env, Query) -> Tag`	Locate the tag to be clicked based on the query	HTML Simulator
	`Type(Env, Query, Text) -> Tag, Text`	Locate the relevant tag based on the query and output the typed text	
	`Select(Env, Query, Text) -> Tag, Text`	Locate the relevant tag based on the query and output the selected option	

(b) Actions used in web tasks.

Task Type	Action Types	Function Descriptions	Implementation
Math	`Calculator(Expression) -> Value`	Calculate mathematical expressions	WolframAlpha
	`SetEquation(Expression) -> Equation`	Set equations based on the given expression	
	`SolveEquation(Equation) -> Solutions`	Solve the system of equations	
	`Define(Variable) -> Variable`	Define a variable	
	`SolveInequality(Inequality) -> Solutions`	Solve the inequality	
	`Code(Function_Description) -> Code`	Generate code for mathematical functions	`gpt-3.5-turbo`
	`Count(List) -> Number`	Count the number of elements in a list	Python

(c) Actions used in math tasks.

Table 9: Action interfaces and execution module implementations for complex interactive tasks.