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 spurred the development of multi-agent systems for complex tasks, yet existing approaches often train agents independently, leading to capability gaps and coordination failures. To address this, we propose MOAT, a Multi-Agent Joint Alignment Tuning framework that bridges the capability gap between planning and grounding agents through iterative joint alignment. MOAT alternates between two key phases: (1) Planning Agent Alignment, which optimizes subgoal generation by rewarding sequences that reduce grounding perplexity, and (2) Grounding Agent Improving, which enhances action generation using high-quality subgoal-action pairs filtered by a critic model. Theoretical analysis proves that MOAT ensures non-decreasing performance and convergence. 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 with 7B-scale models. Notably, MOAT surpasses GPT-4 on Mind2Web by over 50%, showcasing its ability to harmonize smaller open-source LLMs into a competitive multi-agent system. [1]

1 Introduction

The rapid development 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). With the powerful capability of these models, a number of autonomous frameworks using LLM-based agents have been proposed and demonstrated impressive potential in a wide range of applications (Yang et al., 2023; Hong et al., 2024; Team, 2023). These frameworks relying on closed-source models (Achiam et al., 2023; Team et al., 2023) often incur higher costs and raise privacy concerns. In contrast, increasingly powerful open-source alternatives (Touvron et al., 2023; Jiang et al., 2023; Guo et al., 2024) are emerging as the mainstream solution, owing to their flexibility that enables better adaptation.

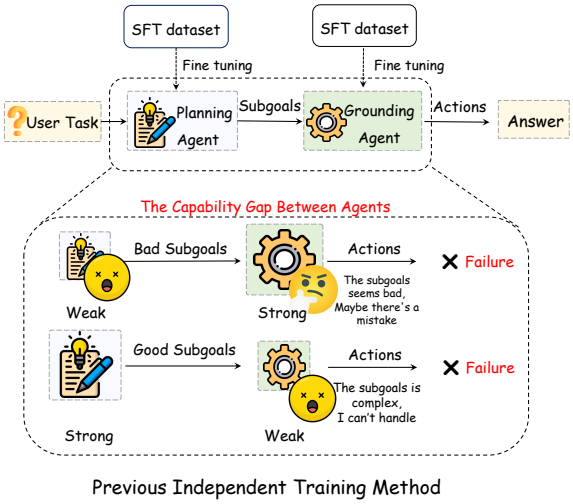

Figure 1: Previous multi-agent training framework, which separately train the *planning* and *Grounding* agents. This leads to a capability gap, e.g., misaligned strengths (strong planning *vs.* weak grounding, or vice versa), causing coordination failures during action execution.

To enhance the reasoning capability of open-source LLM-based agents, several works propose to construct reasoning trajectory data for training (Yao et al., 2023; Madaan et al., 2024) or directly distill from closed-source models (Zeng et al., 2024; Xu et al., 2024; Chen et al., 2024). However, most LLM-based agent frameworks focus on training one model to perform various tasks, which may struggle with complex long-reasoning problems (Liu et al., 2024, 2023) due to the limited capacity of a single model. To address this chal-

[1] Code is available on Anonymous GitHub.

lenge, some recent studies propose the multi-agent approaches (Yin et al., 2024a; Qiao et al.; Shen et al., 2024; Wang et al., 2024). Figure 1(a) illustrates the most commonly-used multi-agent framework. Given a task, a planning agent decomposes it into a sequence of subgoals, and another grounding agent generates corresponding tool-calling actions based on these subgoals, thereby obtaining the final answer. To enable these specialized agents, they first construct the training data for each step and then train the models independently. While performance can be improved through this multi-agent decomposition, we argue that effective collaboration among agents is not guaranteed in these methods. Different agents may exhibit varying proficiency levels in handling assigned tasks due to independent training. This imbalance in specialized capabilities, coupled with the lack of awareness regarding peer competencies of agents, can lead to suboptimal task routing and coordination challenges.

To address the above problems, we propose MOAT, a **M**ulti-agent J**o**int **A**lignment **T**uning framework that iteratively alternates two main steps to enable the alignment in the multi-agent system, including (i) **Planning Agent Alignment** and (ii) **Grounding Agent Improving**. In the Planning Agent Alignment step, MOAT optimizes the planning agent to formulate subgoals that can better guide the grounding agent to generate correct actions. To achieve this, we sample multiple candidate subgoal sequences from the planning agent for each input task. Then, we compute the perplexity of the grounding agents in generating correct tool callings on the condition of the generated subgoal sequences, respectively. Using this perplexity as a reward, we determine the utility of each subgoal sequence, thereby training the planning agent to adapt to the grounding agent's preference through direct preference optimization algorithm (Rafailov et al., 2024) In the Grounding Agent Improving step, we enhance the expertise of the grounding agent to understand the subgoals from the planning agent. Specifically, we remain the sampled subgoal with the lowest perplexity as input for the grounding agent, reinforcing it to generate correct tool-calling actions through a standard language modeling loss. Through theoretic analysis, we demonstrate that the holistic performance of the multi-agent system is improved progressively by alternating the above two steps. Since directly training models on self-generated datasets may introduce noise and potentially results in training collapse (Huang et al., 2024), we introduce a data selection strategy where we use a critic model to judge the correctness of the generated tool invocation sequences and filter out low-quality data.

We apply MOAT to several open-source model families (Llama-2 and Mistral) and evaluate it on three agent tasks such as Web, Math, and QA. The results show that MOAT consistently outperforms existing baselines, both on in-distribution training sets and out-of-distribution test sets. This validates the effectiveness of joint alignment tuning and demonstrates its strong generalization ability. Furthermore, we conduct an in-depth analysis of factors such as the number of samples, iteration counts, and the selection of the critic model. Our contributions can be summarized as follows:

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. LLM-based agent systems can interact with environments to accomplish 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). Single LLM agent frameworks like AutoGPT (Yang et al., 2023), Xagent (Team, 2023), and LangChain (Chase, 2022) can effectively handle diverse complex tasks by integrating external functions. To further enhance performance, recent studies have proposed LLM-based multi-agent systems that accomplish tasks through agent collaboration. For instance, systems like CAMEL (Li et al., 2023), AutoGen (Wu et al., 2024), and ChatEval (Chan et al., 2024) employ role-playing with predefined roles to enable agents to efficiently complete different tasks and achieve common goals. However, existing systems primarily rely on closed-source model APIs, limiting applications in specific domains.

Agent tuning. Recent research has laid the foundation for building language agents based on open-source models by utilizing larger models to generate training data for fine-tuning smaller models, enabling better instruction following and reasoning capabilities. For example, studies such as AgentBank (Song et al., 2024), FireAct (Chen et al., 2023), AgentTuning (Zeng et al., 2024), AgentOhana (Zhang et al., 2024a), and fine-tune the open-source LLMs like Llama by collecting agent trajectory data from strong teacher models like GPT-4. However, single small agent still struggle with complex long-reasoning tasks (Liu et al., 2024). To address this, Lumos (Yin et al., 2024a) and (Shen et al., 2024) proposed a multi-agent training framework that promotes collaboration among agents to solve complex tasks. Meanwhile, Autoact (Qiao et al.) proposes a multi-agent learning framework that utilizes self-instruct to autonomously generate self-improving data, thereby eliminating the reliance on synthesis trajectories from closed-source models.

However, existing works either focus on building data to train individual models or separately train multiple models, neglecting the joint optimization of multiple agents. To address this limitation, we propose a multi-agent joint optimization algorithm for multi-agent systems.

3 Agent Task Preliminary

A multi-agent system typically comprises three core components: (1) a planning agent that decomposes tasks into subgoals, (2) a grounding agent that translates subgoals into executable actions, and (3) an execution module that implements the generated actions to get final answer. Given a task x, the planning agent is responsible for decomposing the user task x into a sequence of subgoals, $S = \pi_{\mathrm{p}}(x) = \{s_1, s_2, \ldots, s_{|S|}\}$, where each s_i represents a subgoal that contributes to the solution of the task x and π_{p} is the planning agent parameters. The grounding agent, based on the user's task x, the set of available tools I, and the decomposed subgoals S, then generates a sequence of tool calls $A = \pi_{\mathrm{g}}(x, I, S) = \{a_1, a_2, \ldots, a_{|A|}\}$, where each $a_i \in I$ represents an individual tool invocation required to complete the subgoal s_i and π_{g} is the agent parameters of grounding agent. Finally, the *execution* module is responsible for executing the generated tool-call sequence A to accomplish the user task x.

4 Multi-agent Joint Alignment Tuning

In this section, we introduce the MOAT, a multi-agent joint alignment tuning framework, which aligns the planning and grounding agents in an iteratively manner, thereby improving the holistic performance. Prior to applying our framework, we perform an initial fine-tuning to equip the model with foundational problem-solving abilities.

As illustrated in Figure 2, MOAT alternates between two steps: (i) **Planning Agent Alignment** and (ii) **Grounding Agent Improving**. The Planning Agent Alignment aims to facilitate exploration of the planning agent in generating subgoals. This exploration encourages the generation of subgoals that better assist the grounding agent in generating the correct action sequence A.

The Grounding Agent Improving focuses on constructing a high-quality and diverse supervised fine-tuning (SFT) dataset of subgoal sequences and tool invocation sequences, aiming to enhance the grounding agent's generalization ability to the planning agent.

This process is iterative: in each round, the planning and grounding agents continuously adapt by learning from the more effective behaviors of the previous round, leading to improved collaboration and overall performance consistency.

4.1 Initial Tuning

To equip the model with the ability to initially address user tasks, we first conduct an initial tuning using the SFT dataset collected in previous work (Yin et al., 2024a).

Formally, given an input task x and the ground truth subgoals S, the planning agent training objective of SFT is to minimize the following loss:

$$\mathcal{L}_{\mathrm{P}} = -\sum_{i=1}^{|S|} \log P_{\pi_{\mathrm{p}}}(S_i \mid S_{<i}; x), \qquad (1)$$

Given the generated subgoals S from the planning agent, the grounding agent learns to ground it to the corresponding tool-calling actions A, which can be formulated as:

$$\mathcal{L}_{\mathrm{G}} = -\sum_{i=1}^{|A|} \log P_{\pi_{\mathrm{g}}}(A_i \mid A_{<i}; x, I, S), \qquad (2)$$

where I is the list of available tools. The final answer is obtained by executing the tool-callings A.

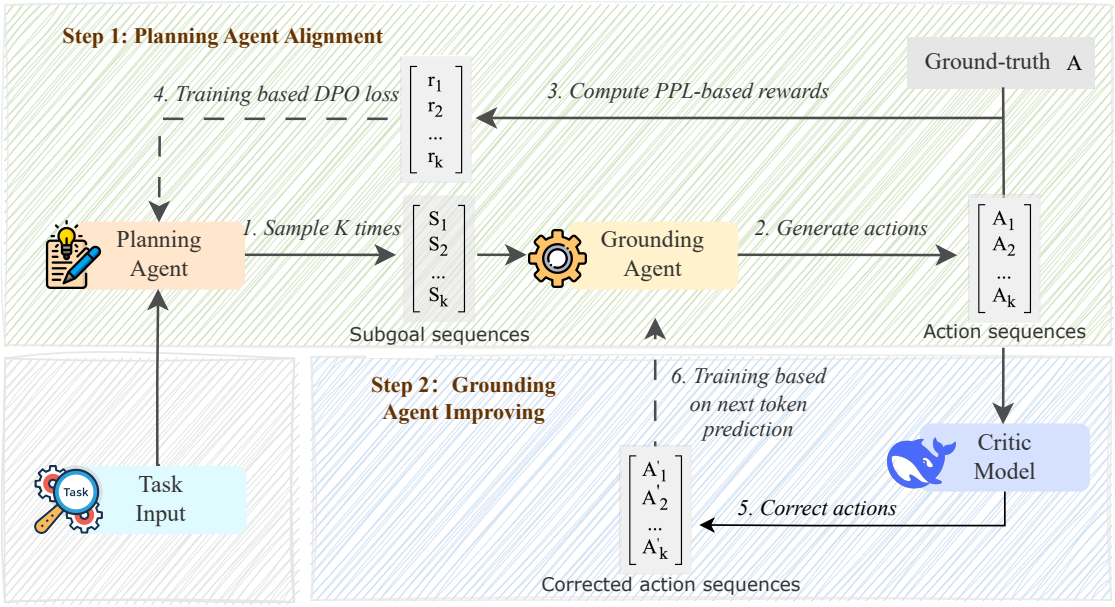

Figure 2: The proposed MOAT framework iteratively alternates between two steps: (1) Planning Agent Alignment: The planning agent samples K candidate subgoal sequences , and the grounding agent generates corresponding tool-calling actions. Subgoal sequences are ranked by PPL, and the planning agent is 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.

4.2 Planning Agent Alignment

As highlighted in previous work (Hou et al., 2025), models demonstrate strong capabilities within their parameter space. Therefore, performing multiple samplings within this space to explore valuable reasoning paths is crucial for optimizing the model. Therefore, after initialization, we sample diverse subgoal sequences from the planning agent at a high temperature to improve its alignment with the grounding agent.

Formally, given the task x , we sample K candidate subgoal sequences from the planning agent:

$$\mathcal{D}_{\text{sample}} = \{S_1, S_2, \ldots, S_K\}. \tag{3}$$

For each $S_k \in \mathcal{D}_{\text{sample}}$, we calculate its perplexity (PPL) with respect to the ground agent as a measure of the planning agents's behavior reward. A lower perplexity indicates that the subgoal sequence is more helpful to the grounding agent, facilitating the generation of correct responses. Therefore, the PPL can be used as a reward to determine the value of the S to the end-to-end task performance. This PPL are formulated as follows:

$$\text{PPL}_{\text{G}}(A \mid x, I, S_k) := \tag{4}$$
$$\exp\Big\{-\frac{1}{|A|}\sum_{i=1}^{|A|}\log P_{\text{g}}(A_i \mid A_{<i}, x, I, S_k)\Big\}.$$

We aim to align the planning agent with subgoal sequences that help the grounding agent generate action sequences with low perplexity. To achieve this, we employ contrastive alignment to reinforce desirable behaviors while penalizing undesirable ones. Specifically, we adopt the Direct Preference Optimization (DPO) framework (Rafailov et al., 2024), which provides an effective approach to align agent behaviors with human preferences. The corresponding loss function is formulated as follows:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(t,(S_w,S_l))\sim D}$$
$$\left[\log\sigma\left(\beta\log\frac{\pi_{\text{p}}(S_w|x)}{\pi_{\text{ref}}(S_w|x)} - \beta\log\frac{\pi_{\text{p}}(S_l|x)}{\pi_{\text{ref}}(S_l|x)}\right)\right],$$

where (S_w, S_l) represents a response pair for the task x, with S_w denoting the response with the lowest PPL-based reward and S_l the highest. Here, π_{p} denotes the current planning agent being optimized, while 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 hyper-parameter.

4.3 Grounding Agent Improving

To enhance the generalization capability of the grounding agent and improve its adaptability to the diverse subgoal sequences generated by the planning agent, we construct a SFT dataset using the diverse subgoal sequences sampled in step 1 along with the corresponding tool invocation sequences generated by the grounding agent for fine-tuning. Specifically, for a given task x and the sampled set of subgoal sequences $(S_1, S_2, \ldots, S_K)$ in Equation 3, the grounding agent π_{g} generates the corresponding response $A_k = \pi_{\mathrm{g}}(x, I, S_k)$ for each subgoal sequence S_k.

Prior research has shown that relying solely on model-generated responses for training may introduce noise and potentially lead to training collapse without external feedback (Huang et al., 2024). To address this issue, we introduce an external feedback mechanism to validate the correctness of the generated tool invocation sequence A_k. Since multiple tool invocation sequences can achieve the same goal in agent tasks, direct result comparison is insufficient for determining correctness. Therefore, we leverage a more powerful language model as a critic model to evaluate whether the generated action sequence A_k aligns with the ground truth—i.e., whether it successfully accomplishes the intended task. If the sequence fails to complete the task, the critic model references the correct answer and provides corrections. The dataset construction procedure is shown in Algorithm 1, where $\mathcal{C}(\cdot)$ denotes the evaluation function of the critic model, and A_{correct} is the corrected invocation sequence provided by the critic model when $\mathcal{C}(\langle x, I, S_k\rangle, A_k) = \text{False}$. The prompt for the critic model is provided in Appendix A.3.

Next, the two stages described above will be iteratively executed to achieve joint optimization.Through iterative process, the planning agent gradually adapts to the grounding agent, generating subgoal sequences that better align with its inference process. Simultaneously, the grounding agent, in turns, improves its generalization capability to handle the diverse outputs of the planning agent. More details of the training algorithm are provided in Algorithm 2.

Algorithm 1: Dataset Construction

```
Initialize SFT dataset D̂_G ← ∅;
for each task x and S_k ∈ D_sample do
    Generate A_k ← π_g(x, I, S_k);
    if C(⟨x, I, S_k⟩, A_k) = True then
        D̂_G ← D̂_G ∪ {(x, I, S_k), A};
    end
    else
        D̂_G ← D̂_G ∪ {(x, I, S_k), A_correct};
    end
end
return SFT dataset D̂_G;
```

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:

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

Here the reward function $R(s,t)$ evaluate the quality of tool-calling action t given sub-goal sequence 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:

$$\max_{\theta_{\mathrm{p}}} \mathbb{E}_{s\sim\pi_{\mathrm{p}}(x)}\left[-\mathrm{PPL}(s;\pi_{\mathrm{g}})\right]. \quad (6)$$

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

$$\max_{\theta_{\mathrm{p}}} \mathbb{E}_{s\sim\pi_{\mathrm{p}}(x)}\left[R(s,t)\right]. \quad (7)$$

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

$$\mathbb{E}[R]^{(t+1)} \geq \mathbb{E}[R]^{(t)}. \quad (8)$$

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 (s, a) generated by the planning agent. The corresponding optimization objective is:

$$\min_{\theta_g} \mathbb{E}_{(s,t)\sim\mathcal{D}} \left[\mathcal{L}(\pi_g(t|s))\right], \tag{9}$$

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:

$$\max_{\theta_g} \mathbb{E}_{(s,t)\sim\mathcal{D}} \left[\log \pi_g(t|s)\right]. \tag{10}$$

Since improved log-likelihood corresponds to reduced PPL and, consequently, higher reward, it follows that:

$$\mathbb{E}[R]^{(t+1)} \geq \mathbb{E}[R]^{(t)}. \tag{11}$$

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 have established that both optimization steps ensure non-decreasing expected rewards:

$$\mathbb{E}[R]^{(t+1)} \geq \mathbb{E}[R]^{(t)}. \tag{12}$$

Additionally, the expected reward $\mathbb{E}[R]$ is upper-bounded due to the following reasons: (i) The reward function $R(s,t)$ is bounded in practical scenarios; and (ii) The PPL has a lower bound. Based on the *Monotone Convergence Theorem*, the non-decreasing and upper-bounded nature of $\{\mathbb{E}[R]^{(i)}\}_{i=1}^{\infty}$ ensures this sequence converges to a finite limit, thereby proving the training convergence.

6 Experimental Setup

6.1 Benchmarks

Following previous work (Song et al., 2024; Chen et al., 2024), we evaluate our MOAT under both held-in and held-out settings. In each setting, we experiment on a wide range of tasks, including Mathematical reasoning (Math), Web interaction (Web), and question answering. As listed in Table 1, the held-in setting includes the GSM8K (Cobbe et al., 2021) StrategyQA (Geva et al., 2021), and Mind2Web (Deng et al., 2023); while the held-out setting includes SVAMP (Patel et al., 2021), WebShop (Yao et al., 2022), and HotpotQA (Yang et al., 2018). For solving interactive tasks, we integrate commonly used actions for each task into the pre-defined action interfaces as tool set I. Details of supported executable actions are included in Appendix A.4.

Task	Skill Dim.	#Inst.	Metric
Held-in Tasks			
HotpotQA (Yang et al., 2018)	QA	100	Exact Match
GSM8K (Cobbe et al., 2021)	Math	500	Exact Match
Mind2Web (Deng et al., 2023)	Web	100	Step SR
Held-out Tasks			
StrategyQA (Geva et al., 2021)	QA	100	Exact Match
SVAMP (Patel et al., 2021)	Math	1000	Exact Match
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.

6.2 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 well-known multi-agents training framework that separately fine-tines each agent on part of overall datasets to obtain specialized agents. Besides, we also select GPT-3.5-Turbo and GPT-4 (Achiam et al., 2023) as strong baselines.

6.3 Implementation Details

We mainly the `Llama2-7b-hf` as backbone LLMs for agents in MOAT and baseline methods. For a comprehensive evaluation, we also alternate it with `Mistral-7B-Instruct-v0.2`, validating our method across various LLMs. For the initial tuning, we use the publicly available datasets from Lumos (Yin et al., 2024a) to fine-tuning the base model. We set the number of sampled subgoal sequences K to 15 and the number of training iteration to 2. And we set the temperature to 1.0 in the sampling process to ensure the sampling of diverse subgoal sequences. We employ `DeepSeek-R1-Distill-Qwen-32B` as the critic model. More detailed training settings can be found in Appendix A.1.

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	47.2	32.6	45.0	41.6	65.5	58.3	25.0	49.6
MOAT	Llama-2-7B	**47.8**	**35.4**	49.7	**44.3**	**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**

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

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 in 7B-scale models. Furthermore, the MOAT with Llama-7B demonstrates an average improvement of 15.8% compared to Agent-Tuning with `Llama-13B`. These results validate the effectiveness of our method in jointly training interconnected specialized agents to enhance overall task-solving performance.

Held-out Tasks. We further investigate the generalizability of our method in solving unseen tasks, i.e., out-of-domain tasks. As illustrated in Table 2, our method achieves the highest performance compared to open-source baselines. For instance, the MOAT with `Mistral-7B` outperforms Lumos with an average performance improvement of 4.4%. A potential 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 tasks.

Comparison with API-based Agents. Although our model was trained using `Llama-7B` and `Mistral-7B-Instruct-v0.2`, it achieves more than a 50% performance improvement over GPT-4 on the Mind2Web task. These findings further validate the effectiveness of our method in synergizing smaller open-source models to achieve competitive performance.

Method	Mind2Web	WebShop
Full MOAT	35.43	60.63
-w/o stage 1	$31.94_{\downarrow 3.49}$	$58.76_{\downarrow 1.87}$
-w/o stage 2	$34.72_{\downarrow 0.71}$	$60.29_{\downarrow 0.34}$

Table 3: Ablation study on two web datasets.

Figure 3: Evaluation results of MOAT using different numbers of sampled subgoals responses (K) on three held-in tasks.

7.2 Ablation Study

To further analyze the effectiveness of planning alignment and grounding agent alignment in MOAT, we conduct an ablation study by removing each component individually. We denote these variants as `w/o stage 1` and `w/o stage 2` As shown

in Table 3, both variants exhibit significant performance degradation, underscoring the importance of our proposed joint alignment tuning method. Notably, the most substantial performance drop occurs when stage 1 is removed, highlighting the critic role of aligning the planning agent to generate coherent subgoal sequences for the grounding agent. This finding suggests that effective multi-agent cooperation relies more heavily on the planning agent's ability to provide accuracy subgoals, which in turn enables the grounding agent to execute tasks more efficiently.

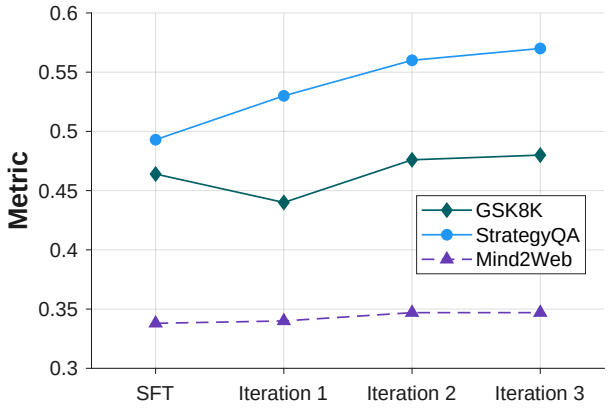

Figure 4: Performance trends of MOAT system (with K=15) across three held-in tasks as the number of iterations increases.

7.3 Detailed 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 conduct additional experiments by varying K from 5 to 15 during the training of `Mistral-7B` at iteration 2. As shown in Figure 3, we observe a positive correlation between the sampling number and the final performance.

We also identify a performance drop on the GSM8K and Mind2Web benchmarks when K=5. We attribute this to the possibility that with insufficient sampling, the model may fail to sample high-quality subgoal sequences that align well with the grounding agent. In such cases, even the subgoal sequence with the highest reward can negatively impact model optimization.

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, beyond a certain threshold, the performance gain becomes marginal. We hypothesize that this is because, after a certain point, the planning and grounding agents reach an equilibrium in performance, as demonstrated in Section 5.

Model	Mind2Web	WebShop
MOAT-Qwen-14B	33.52	60.63
MOAT-Qwen-32B	34.03	**60.78**
MOAT-GPT-4O	**35.28**	60.57

Table 4: Model Performance on WebShop and Mind2Web benchmark using different critic model

Impact of Different Critic Model. We employ the `DeepSeek-R1-Distill-Qwen-32B` as a critic model, to refine the action sequences generated by the grounding agent. To qualitatively analyze the impact of the critic model, we conduct experiments using both a more powerful critic model (i.e., GPT-4O) and a weaker one (i.e., `DeepSeek-R1-Distill-Qwen-14B`). The results, as presented in Table 4, reveal a clear trend: as the capability of the critic model improves, the overall performance of MOAT also increases. These observations highlight the importance of selecting a robust critic model to optimize the performance of our framework.

7.4 Case Study

We conduct case studies to demonstrate the effectiveness of our joint alignment tuning framework in solving complex tasks. The results show that our MOAT effectively improves the task planning performance and enhances the adaptability of the grounding agent. Concrete examples and 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 aligned 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. Extensive experiments across six benchmarks demonstrate the superiority of MOAT over existing methods.

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 and Mistral) 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*.

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.

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)*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

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*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

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*.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, et al. 2023. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *arXiv preprint arXiv:2308.05960*.

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*.

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*.

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*.

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

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

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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*.

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*.

Yiheng Xu, SU Hongjin, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, et al. 2024. Lemur: Harmonizing natural language and code for language agents. In *The Twelfth International Conference on Learning Representations*.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

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.

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*.

Algorithm 2: Iterative Optimization

Input: The number of iterations N_r, the number of samples K, the tasks set T, the set of available tools I, the critic model CRITIC , planning agent PLANNING_0 , grounding agent GROUNDING_0

for *iteration r = 1... N_r* **do**
  for *each task $x_i \in T$* **do**
    # Sample K response
    for $j \leftarrow 1$ **to** K **do**
      $\widetilde{\boldsymbol{S}}_{i,j} \leftarrow \text{PLANNING}_{r-1}(x_i)$
      $\boldsymbol{r}_{i,j} \leftarrow \text{PPL}_{G_r}(A_i|x_i, I, \widetilde{\boldsymbol{S}}_{i,j})$
    end
  end
  # PLANNING Optimization
  $S_{win} \leftarrow \widetilde{\boldsymbol{S}}[\text{argmax}(\boldsymbol{r}, axis = 1)]$
  $S_{lose} \leftarrow \widetilde{\boldsymbol{S}}[\text{argmin}(\boldsymbol{r}, axis = 1)]$
  $\boldsymbol{\theta}^{\text{PLANNING}_r} \leftarrow$
    $\boldsymbol{\theta}^{\text{PLANNING}_{r-1}} - \frac{\partial \mathcal{L}_{\text{DPO}}[(S_{win}, S_{lose}), T]}{\partial \boldsymbol{\theta}^{\text{PLANNING}_{r-1}}}$
  # GROUNDING Optimization
  for *each task $x_i \in T$* **do**
    # Sample K response
    for $j \leftarrow 1$ **to** K **do**
      $\widetilde{\boldsymbol{A}}_{i,j} \leftarrow$
      $\text{GROUNDING}_{r-1}(A_i|x_i, I, \widetilde{\boldsymbol{S}}_{i,j})$
    end
  end
  $\widetilde{\boldsymbol{A}}'_{i,j} = \text{CRITIC}(\widetilde{\boldsymbol{A}}_{i,j})$
  $\boldsymbol{\theta}^{\text{GROUNDING}_r} \leftarrow$
    $\boldsymbol{\theta}^{\text{GROUNDING}_{r-1}} - \frac{\partial \mathcal{L}_{\text{SFT}}[(\widetilde{\boldsymbol{S}}, I, T), \widetilde{\boldsymbol{A}'}]}{\partial \boldsymbol{\theta}^{\text{GROUNDING}_{r-1}}}$
end

Output: $\text{PLANNING}_{N_r}, \text{GROUNDING}_{N_r}$

A Appendix

A.1 Implementation Details

We show more training details about our experiments.All our experiments are conducted on 2 × NVIDIA A800 (80GB) GPUs.

For initial tuning, we implement training over two epochs with a learning rate of 2×10^{-5} and a batch size 128. And We set the maximum sequence length to 1024. 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. To reduce the impact of noisy data, we select only the (S_w, A) pairs with the highest reward from $\widehat{D}_{\text{G}}$ for training. 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.

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

[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}
Original subgoals: {SUBGOALS}
Original 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.

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.