Small LLMs Are Weak Tool Learners: A Multi-LLM Agent

Anonymous ACL submission

Abstract

Large Language Model (LLM) agents significantly extend the capabilities of standalone LLMs, empowering them to interact with external tools (e.g., APIs, functions) and complete various tasks in a self-directed fashion. The challenge of tool use demands that LLMs not only understand user queries and generate answers accurately but also excel in task planning, tool invocation, and result summarization. While traditional works focus on training a single LLM with all these capabilities, performance limitations become apparent, particularly with smaller models. To overcome these challenges, we propose a novel approach that decomposes the aforementioned capabilities into a planner, caller, and summarizer. Each component is implemented by a single LLM that focuses on a specific capability and collaborates with others to accomplish the task. This modular framework facilitates individual updates and the potential use of smaller LLMs for building each capability. To effectively train this framework, we introduce a two-stage training paradigm. First, we fine-tune a backbone LLM on the entire dataset without discriminating sub-tasks, providing the model with a comprehensive understanding of the task. Second, the fine-tuned LLM is used to instantiate the planner, caller, and summarizer respectively, which are continually fine-tuned on respective sub-tasks. Evaluation across various tool-use benchmarks illustrates that our proposed multi-LLM framework surpasses the traditional single-LLM approach, highlighting its efficacy and advantages in tool learning.

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing with remarkable proficiency in understanding and generating text. Despite their impressive capabilities, LLMs are not without limitations. Notably, they lack domain specificity, real-time information, and face challenges in solving specialized problems such as mathematics (Gou et al., 2023) and program compilation (OpenAI, 2023a). Hence, integrating LLMs with external tools, such as API calls and Python functions, becomes imperative to extend their capabilities and enhance the overall performance. Consequently, LLM agents have become a prominent area for both academia and industry, employing large language models to determine when and how to utilize external tools to tackle various tasks.

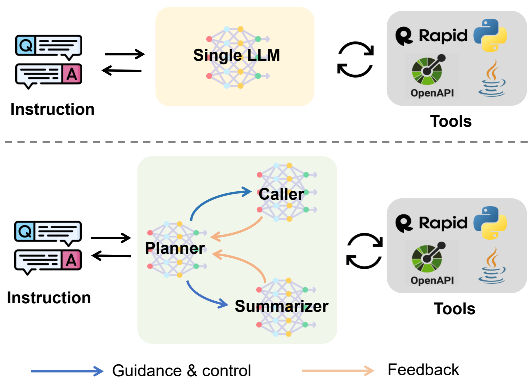

Figure 1: A conceptual comparison of the traditional single-LLM agent framework (top) and the proposed multi-LLM agent framework, α-UMi (bottom).

In addition to exploring proprietary LLMs like GPT-4, researchers have also actively engaged in developing customizable agent systems by fine-tuning open-source LLMs on diverse tool-use datasets (Patil et al., 2023; Tang et al., 2023; Qin et al., 2023b; Gou et al., 2023). The challenge of tool learning demands sufficiently large and complex LLMs. These models must not only comprehend user queries but also excel in task planning, tool selection and invocation, and result summarization (Yujia et al., 2023). These capabilities draw upon different facets of the LLMs; for instance, planning relies more on reasoning ability, while tool selection and invocation demand legal and accurate request writing, and result summarization requires adept conclusion-drawing skills. While conventional approaches (Qin et al., 2023b; Gou et al., 2023; Zeng et al., 2023) focus on training a single open-source LLM with all these capabilities, notable performance limitations have been observed, especially with smaller open-source LLMs (Touvron et al., 2023a,b). Moreover, the tools could be updated frequently in practical scenarios, when the entire LLM requires potential retraining.

To address these challenges, we propose a multi-LLM agent framework for tool learning, α-UMi[1]. As illustrated in Figure 1, α-UMi decomposes the capabilities of a single LLM into three components, namely planner, caller, and summarizer. Each of these components is implemented by a single LLM and trained to focus on a specific capability. The planner is designed to generate the rationale based on the current state of the system and weighs between selecting the caller or summarizer to generate downstream output, or even deciding to terminate the execution. The caller is directed by the rationale and responsible for invocating specific tools. The summarizer is guided by the planner to craft the ultimate user answer based on the execution trajectory. These components collaborate seamlessly to accomplish various tasks. Compared to previous approaches, our modular framework has three distinct advantages. First, each component undergoes training for a designated role, ensuring enhanced performance for each capability. Second, the modular structure allows for individual updates to each component as required, ensuring adaptability and streamlined maintenance. Third, since each component focuses solely on a specific capability, potentially smaller LLMs can be employed.

To effectively train this multi-LLM framework, we introduce a novel global-to-local progressive fine-tuning strategy (GLPFT). First, an LLM backbone is trained on the original training dataset without discriminating between sub-tasks, enhancing the comprehensive understanding of the tool-learning task. Three copies of this LLM backbone are created to instantiate the planner, caller, and summarizer, respectively. In the second stage, the training dataset is reorganized into new datasets tailored to each LLM's role in tool use, and continual fine-tuning of the planner, caller, and summarizer is performed on their respective datasets.

We employ LLaMA-2 (Touvron et al., 2023b) series to implement the LLM backbone and evaluate our α-UMi agent on several tool learning benchmarks (Qin et al., 2023b; Tang et al., 2023). Experimental results demonstrate that our proposed framework outperforms the single-LLM approach across various model and data sizes. Moreover, we show the necessity of the GLPFT strategy for the success of our framework and delve into the reasons behind the improved performance. Finally, the results confirm our assumption that smaller LLMs can be used in our multi-LLM framework to cultivate individual tool learning capabilities and attain a competitive overall performance.

To sum up, this work makes three critical contributions. First, we demonstrate that small LLMs are weak tool learners and introduce α-UMi, a multi-LLM framework for building LLM agents, that outperforms the existing single-LLM approach in tool use. Second, we propose a GLPFT fine-tuning strategy, which has proven to be essential for the success of our framework. Third, we perform a thorough analysis, delving into data scaling laws and investigating the underlying reasons behind the superior performance of our framework.

2 Related Works

2.1 Tool Learning

The ability of LLMs to use external tools has become a pivotal component in the development of AI agents, attracting rapidly growing attention (Qin et al., 2023b; Schick et al., 2023; Yang et al., 2023b; Shen et al., 2023; Patil et al., 2023; Qin et al., 2023a). Toolformer (Schick et al., 2023) was one of the pioneering work in tool learning. Subsequently, a diverse array of external tools has been employed to enhance LLMs in various ways, including the knowledge retriever (Yang et al., 2023a; Nakano et al., 2021), visual models (Yang et al., 2023b; Wu et al., 2023a; Yang et al., 2023c; Shen et al., 2023), code and math reasoning (Gou et al., 2023; OpenAI, 2023a), and APIs (Li et al., 2023; Qin et al., 2023b). Different from previous approaches relying on a single LLM for tool learning, we introduce a novel multi-LLM collaborated tool learning framework designed for smaller open-source LLMs.

2.2 LLM-powered Agents

Leveraging the capabilities of LLMs such as ChatGPT (OpenAI, 2022) and GPT-4 (OpenAI, 2023b),

[1] In astronomy, the name "α-UMi" is an alias of the Polaris Star (`https://en.wikipedia.org/wiki/Polaris`), which is actually a triple star system consisting of a brighter star (corresponding to the planner) and two fainter stars (corresponding to the caller and the summarizer).

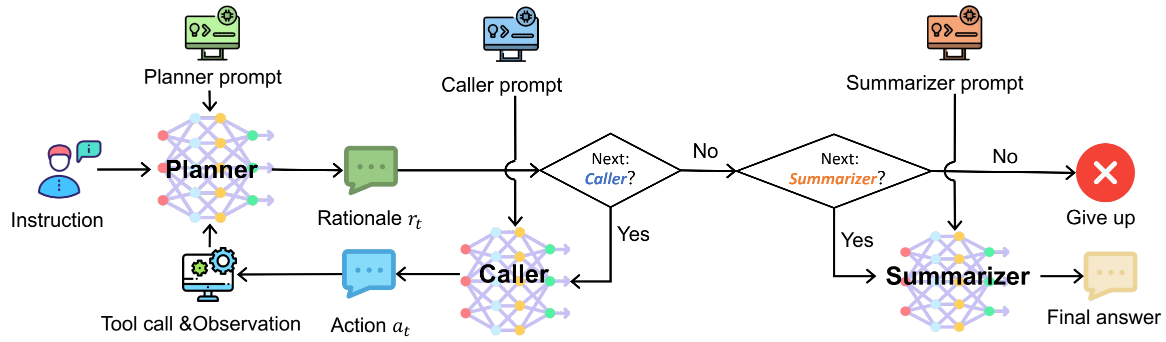

Figure 2: An illustration of how α-UMi works to complete a task.

AI agent systems have found application in diverse scenarios. For instance, BabyAGI (Nakajima, 2023) and AutoGPT (Gravitas, 2023) have been developed to address daily problems, while Voyager (Wang et al., 2023) and Ghost (Zhu et al., 2023) engage in free exploration within Minecraft games. Additionally, MetaGPT (Hong et al., 2023), ChatDev (Qian et al., 2023a), and AutoGen (Wu et al., 2023b) contribute to the development of multi-agent frameworks tailored for software development and problem-solving.

3 Methodology

3.1 Preliminary

Agents for tool learning are systems designed to assist users in completing tasks through a series of decision-making processes and tool use (Yujia et al., 2023). In recent years, these agents commonly adhere to the ReACT framework (Yao et al., 2022). The backbone of the agent is an LLM denoted as $\mathcal{M}$. Given the user instruction q and the system prompt $\mathcal{P}$, the agent solves the instruction step by step. In the tth step, the LLM $\mathcal{M}$ generates a rationale r_t and an action a_t based on the instruction and the current state of the system:

$$r_t, a_t = \mathcal{M}(\mathcal{P}, \tau_{t-1}, q), \tag{1}$$

where $\tau_{t-1} = \{r_1, a_1, o_1, ..., r_{t-1}, a_{t-1}, o_{t-1}\}$ denotes the previous execution trajectory. Here, o_t denotes the observation returned by tools when the action a_t is supplied. In the final step of the interaction, the agent generates rationale r_n indicating that the instruction q is solved along with the final answer a_n or that it will abandon this execution run. Therefore, no observation is included in this step.

3.2 The α-UMi Framework

As previously mentioned, the task of tool learning imposes a significant demand on the capabilities of LLMs, including task planning, tool invocation, and result summarization. Coping with all these capabilities using a single open-source LLM, especially when opting for a smaller LLM, appears to be challenging. To address this challenge, we introduce the α-UMi framework, which breaks down the tool learning task into three sub-tasks and assigns each sub-task to a dedicated LLM. Figure 1 presents an illustration of our framework, which incorporates three distinct LLM components: planner $\mathcal{M}_{\text{plan}}$, caller $\mathcal{M}_{\text{call}}$, and summarizer $\mathcal{M}_{\text{sum}}$. These components are differentiated by their roles in tool use, and each component model has a unique task definition, system prompt[2], and model input.

The workflow of α-UMi is shown in Figure 2. Upon receiving the user instruction q, the planner generates a rationale comprising hints for the this step. This may trigger the caller to engage with the tools and subsequently receive observations from the tools. This iterative planner-caller-tool loop continues until the planner determines that it has gathered sufficient information to resolve the instruction. At this point, the planner transitions to the summarizer to generate the final answer. Alternatively, if the planner deems the instruction unsolvable, it may abandon the execution.

Planner: The planner assumes responsibility for planning and decision-making, serving as the "brain" of our agent framework. Specifically, the model input for the planner comprises the system prompt $\mathcal{P}_{\text{plan}}$, the user instruction q, and the previous execution trajectory τ_{t-1}. Using this input, the planner generates the rationale r_t:

$$r_t = \mathcal{M}_{\text{plan}}(\mathcal{P}_{\text{plan}}, \tau_{t-1}, q). \tag{2}$$

Following the rationale, the planner generates the decision for the next step: (1) If the decision is "Next: Caller", the caller will be activated and an

[2]The prompts for each LLM are provided in Appendix A.

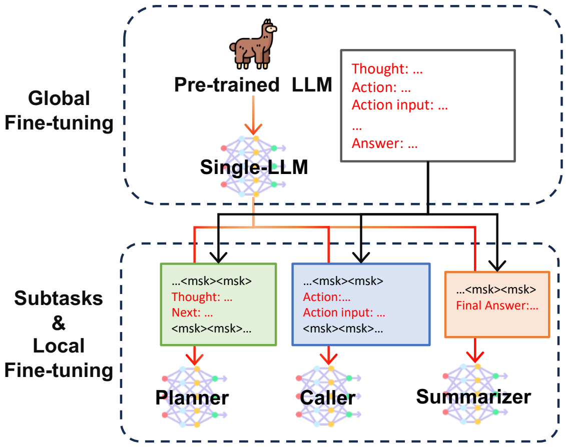

Figure 3: Global-to-local progressive fine-tuning.

action will be generated for calling tools. (2) If the decision is "Next: Summarizer", the summarizer will be activated to generate the final answer for the user, and the agent execution will finish. (3) If the decision is "Next: Give up", it means that the user's instruction cannot be solved in the current situation, and the system will be terminated.

Caller: Interacting with the tools requires the LLM to generate legal and useful requests, which may conflict with other abilities such as reasoning and general response generation during fine-tuning. Therefore, we train a specialized caller to generate the action for using tools. The caller takes the user instruction q and the previous execution trajectory τ_{t-1} as input. To make the caller focus on the planner's rationale r_t in the current step, we also design a prompt $\mathcal{P}_{\text{call}}$ to explicitly remind the caller:

$$a_t = \mathcal{M}_{\text{call}}(\mathcal{P}_{\text{call}}, \tau_{t-1}, q, r_t). \tag{3}$$

Summarizer: The agent's final response, which aims to offer informative and helpful information to the user, is distinct from the rationales that primarily focus on planning and reasoning. Therefore, we employ a dedicated summarizer tasked with generating the final answer a_n. This model utilizes a concise prompt $\mathcal{P}_{\text{sum}}$ designed to guide the model in concentrating on summarizing the execution trajectory and presenting the answer to the user:

$$a_n = \mathcal{M}_{\text{sum}}(\mathcal{P}_{\text{sum}}, \tau_{n-1}, q, r_n). \tag{4}$$

In Figure 7 and Figure 8, we show several cases of our α-UMi on downstream tasks.

3.3 Global-to-Local Progressive Fine-Tuning

To effectively fine-tune the above multi-LLM system is a complex endeavor: On one hand, generating the rationale, action, and final answer can facilitate each other during the training process, and enhance the model's comprehension of the entire agent task (Chen et al., 2023). On the other hand, the constraints on model capacity make it challenging to fine-tune a small LLM to achieve peak performance in generating rationales, actions, and final answers simultaneously (Dong et al., 2023). Taking into account these two points, we propose a global-to-local progressive fine-tuning (GLPFT) strategy for α-UMi. The motivation behind this strategy is to first exploit the mechanism by which the generation of rationale, action, and final answer can mutually enhance each other. Then, once the single LLM reaches its performance ceiling, it is subsequently split into planner, caller and summarizer for further fine-tuning, in order to enhance its capabilities in the subtasks and mitigate the performance constraints due to limited model capacity.

As depicted in Figure 3, this GLPFT strategy comprises two distinct stages. The first stage involves global fine-tuning, where we fine-tune a backbone LLM on the original training dataset without distinguishing between sub-tasks. After this stage, the backbone LLM is trained to sequentially output the rationale, action, and answer as introduced in Section 3.1. Then, we create three copies of the backbone LLM, designated as the planner, caller, and summarizer, respectively.

The second stage is local fine-tuning, where we reorganize the training dataset tailored to each LLM's role, as introduced in Section 3.2. We then proceed to fine-tune the planner, caller, and summarizer on their respective datasets, thereby further enhancing their specific abilities in each sub-task. During this local fine-tuning stage, we opt to reuse the set of user instructions curated in the global fine-tuning stage. The only adjustment made to the training set is the change in the format of the training data. As illustrated in Figure 3, the fine-tuning objective during the second stage for the planner, caller, and summarizer is oriented towards generating the rationale, action, and final answer, respectively. While the gradients from other text spans are stopped. Simultaneously, we refine the system prompts for the training data of the planner, caller, and summarizer, as detailed in Appendix A.

3.4 Discussions

Recent studies have explored multi-agent systems based on LLMs across various domains, such as social communication (Park et al., 2023; Wei et al., 2023), software development (Qian et al., 2023a; Hong et al., 2023), and tool learning (Song et al., 2023; Qian et al., 2023b). However, these frameworks typically rely on robust closed-source LLMs, demanding advanced functionalities such as automatic cooperation and feedback, capabilities that surpass those available in open-source LLMs. In contrast, our α-UMi aims to alleviate the LLM's workload in tool-use tasks by integrating multiple LLMs into an agent, particularly well-suited for open-source, smaller LLMs. Besides, we introduce the GLPFT method for fine-tuning the multi-LLM system, a novel contribution not present in existing multi-agent frameworks. We plan to incorporate these discussions in the upcoming revision.

4 Experimental Settings

4.1 Benchmarks

We evaluate the effectiveness of our α-UMi on the well recognized tool learning benchmark: ToolBench (Qin et al., 2023b). This benchmark involve integrating API calls to accomplish tasks, where the agent must accurately select the appropriate API and compose necessary API requests. Moreover, we partition the test set of ToolBench into in-domain and out-of-domain based on whether the tools used in the test instances have been seen during training. This division allows us to evaluate performance in both in-distribution and out-of-distribution scenarios. For additional details and statistics regarding these datasets, please refer to Appendix B. We also evaluate α-UMi on other benchmarks such as ToolAlpaca (Tang et al., 2023) and program-aided agent for mathematical reasoning (Hendrycks et al., 2021; Cobbe et al., 2021). The results are shown in Appendix E.

4.2 Metrics

The tasks in ToolBench involve calling APIs through RapidAPI[3]. This process frequently encounters problems such as API breakdowns, which impacts the fairness of the comparison. To address this problem, we introduce two types of evaluations for ToolBench. In Section 5.1, we first compare the output of agent with the annotated reference at each step[4], which avoids real-time API callings. The metrics for this evaluation include Action EM (Act. EM), Argument F1 (Arg. F1), and Rouge-L (R-L) as proposed by Li et al. (2023). Moreover, we examine the frequency of API name hallucinations (Hallu.) and the accuracy (Plan ACC) of the agent's planning decisions at each step for using tools invocation, generating answer, or giving up. The reference annotations are based on verified ChatGPT execution results provided in ToolBench. We also provide the results based on real-time RapidAPI calling in Section 5.2, which is the original evaluation method used by the ToolBench team.

[3]https://rapidapi.com/hub.

[4]Refer to Appendix C for more details of the evaluation.

4.3 Implementation Details

We opt for LLaMA-2-chat-7B/13B (Touvron et al., 2023b) as the backbone to implement our framework. In the first stage of our GLPFT, we conduct fine-tuning for the backbone LLM with a learning rate of 5e-5 for 2 epochs. Then, we create three copies of this fine-tuned backbone to instantiate the planner, caller, and summarizer, respectively. In the second stage, we fine-tune the three LLMs with a reduced learning rate of 1e-5. The planner and caller undergo fine-tuning for 1 epoch, while the summarizer undergoes fine-tuning for 2 epochs. We set the global batch size to 48 and employ DeepSpeed ZeRO Stage3 (Rajbhandari et al., 2021) to speed up the fine-tuning process. All experimental results are obtained using greedy decoding, with the maximum sequence length set at 4096.

4.4 Baselines

We compare our method with three baseline methods, namely Single-LLM, Multi-LLM$_{\text{one-stage}}$ and Single-LLM$_{\text{multi-task}}$. Single-LLM refers to the traditional single-LLM tool learning approach. Multi-LLM$_{\text{one-stage}}$ involves directly fine-tuning the planner, caller, and summarizer on their own sub-task datasets, without employing our two-stage fine-tuning strategy. Single-LLM$_{\text{multi-task}}$ refers to using the same LLM to fulfill the roles of planner, caller, and summarizer. This particular LLM is fine-tuned on a combined dataset comprising the three sub-task datasets and functions similarly to our multi-LLM framework. We also evaluate the performance of ChatGPT and GPT-4 with 0-shot setting, and ToolLLaMA (Qin et al., 2023b), which is a 7B LLaMA model fine-tuned on ToolBench.

5 Results and Analysis

5.1 Overall Results

The main results are presented in Table 1. We elaborate on our observations from six perspectives:

Model	ToolBench (in-domain)					ToolBench (out-of-domain)				
	Plan ACC	Act. EM	Hallu.	Arg. F1	R-L	Plan ACC	Act. EM	Hallu.	Arg. F1	R-L
Close-Source LLM										
ChatGPT (0-shot)	83.33	58.67	7.40	45.61	23.08	81.62	54.67	8.19	40.08	22.85
GPT-4 (0-shot)	80.28	55.52	5.98	48.74	28.69	77.80	55.26	5.12	47.45	30.61
Model Size = 7B (LoRA)										
Multi-LLM$_{\text{one-stage}}$ (LoRA)	77.76	41.20	2.18	33.21	22.02	79.05	39.25	2.58	33.29	24.66
α-UMi (LoRA)	83.45	44.34	9.61	38.35	34.75	85.84	50.61	4.58	44.65	43.89
Model Size = 7B										
ToolLLaMA (len = 4096)	66.42	19.47	33.94	15.98	2.06	68.21	30.75	25.35	25.07	5.78
ToolLLaMA (len = 8192)	77.02	47.56	4.03	42.00	15.26	77.76	45.07	3.45	40.41	18.10
Single-LLM	81.92	53.26	2.32	45.57	42.66	84.61	56.54	2.26	50.09	**47.99**
Multi-LLM$_{\text{one-stage}}$	87.52	45.11	7.71	38.02	41.01	88.42	53.40	2.52	45.79	46.39
Single-LLM$_{\text{multi-task}}$	85.06	51.83	2.96	44.25	27.40	86.55	56.89	2.77	49.50	32.58
α-UMi$_{w/o\ \text{reuse}}$	88.24	55.50	**0.53**	48.97	39.98	87.91	58.02	2.32	50.55	42.59
α-UMi$_{w/\ \text{reuse}}$	**88.92**	**58.94**	0.57	**52.24**	**43.17**	**89.72**	**60.47**	**0.45**	**53.60**	46.26
Model Size = 13B										
Single-LLM	81.01	59.67	1.53	52.35	42.16	86.74	60.04	2.03	52.94	**48.46**
Multi-LLM$_{\text{one-stage}}$	86.49	50.54	5.11	41.96	36.21	87.45	56.71	3.23	47.49	41.62
Single-LLM$_{\text{multi-task}}$	86.36	58.96	2.00	49.28	28.41	86.64	62.78	3.42	53.29	35.46
α-UMi$_{w/o\ \text{reuse}}$	86.33	60.07	0.39	53.11	35.09	87.75	61.63	2.95	52.54	37.70
α-UMi$_{w/\ \text{reuse}}$	**87.87**	**63.03**	**0.37**	**57.65**	**43.46**	**88.73**	**64.21**	**0.24**	**57.38**	42.50

Table 1: Overall evaluation results on ToolBench.

α-UMi v.s. Existing Methods: When compared to ChatGPT and ToolLLama, α-UMi outperforms them on all metrics. α-UMi exceeds these two baselines in terms of Plan ACC and R-L considerably, demonstrating its alignment with annotated reference in terms of planning execution steps and generating final answers. It is worth mentioning that ToolLLaMA only exhibits acceptable performance when the input length is 8192. At an input length of 4096, ToolLLaMA shows deterioration across various metrics, particularly exhibiting a very high hallucination rate. In contrast, α-UMi only requires the input length of 4096 to achieve a satisfying performance.

α-UMi v.s. Single-LLM: α-UMi outperforms the Single-LLM agent. On ToolBench, we unveil substantial improvements with α-UMi, particularly in Plan ACC, Act. EM, Hallu., and Arg. F1. This finding not only confirm the effectiveness of α-UMi in enhancing the agent's planning and API calling capabilities but also suggest a notable decrease in hallucinations, which can significantly elevate user satisfaction.

Model Scales: When comparing the results of methods with different model sizes, we note that agents with a 13B backbone exhibit superior performance compared to their 7B counterparts. This observation implies that the shift from a 7B to a 13B model results in a improvement in tool utilization capabilities. Significantly, α-UMi with a 7B backbone even outperforms the Single-LLM baseline with a 13B LLM, confirming our earlier assumption that smaller LLMs can be utilized in our multi-LLM framework to develop each capability and achieve competitive overall performance.

Multi-LLM Fine-tuning: α-UMi outperforms Multi-LLM$_{\text{one-stage}}$ and Single-LLM$_{\text{multi-task}}$. Multi-LLM$_{\text{one-stage}}$ even exhibits suboptimal performance compared to the Single-LLM baseline in metrics assessing API calling abilities, such as Act. EM, Hallu., and Arg. F1. This finding highlights the limitations of training each LLM on individual sub-tasks, compromising the comprehensive understanding of the tool-use task. Moreover, the subpar performance of Single-LLM$_{\text{multi-task}}$ indicates that, the limited capacity of small LLMs hinders the agent from effectively fulfilling the roles of planner, caller, and summarizer simultaneously. In contrast, through the application of the GLPFT strategy, α-UMi successfully mitigates this limitation, showcasing its effectiveness in achieving comprehensive tool learning capabilities.

Full Fine-tuning v.s. LoRA: In Multi-LLM$_{\text{one-stage}}$ (LoRA), we directly fine-tuned three LoRAs (Hu et al., 2022) for planner, caller and summarizer, respectively. This strategy is similar to AutoACT (Qiao et al., 2024), while its performance fails to outperform α-UMi with GLPFT. Moreover, we can implement LoRA on the backbone LLM obtained from the first stage of the GLPFT (α-UMi (LoRA)). Applying LoRA on top of this backbone yields better results than Multi-LLM$_{\text{one-stage}}$ (LoRA), but still underperforms the full parameter updating strategy GLPFT. Therefore, we conclude

Method	Model	I1-Inst.		I1-Tool		I1-Cat.		I2-Inst.		I2-Cat.		I3-Inst.		Average	
		Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win
ReACT	Claude-2	5.5	31.0	3.5	27.8	5.5	33.8	6.0	35.0	6.0	31.5	14.0	47.5	6.8	34.4
	ChatGPT	41.5	-	44.0	-	44.5	-	42.5	-	46.5	-	22.0	-	40.2	-
	ToolLLaMA	25.0	45.0	29.0	42.0	33.0	47.5	30.5	50.8	31.5	41.8	25.0	55.0	29.0	47.0
	GPT-4	53.5	60.0	50.0	58.8	53.5	63.5	67.0	65.8	72.0	60.3	47.0	78.0	57.2	64.4
DFSDT	Claude-2	20.5	38.0	31.0	44.3	18.5	43.3	17.0	36.8	20.5	33.5	28.0	65.0	43.1	43.5
	ChatGPT	54.5	60.5	65.0	62.0	60.5	57.3	75.0	72.0	71.5	64.8	62.0	69.0	64.8	64.3
	ToolLLaMA	57.0	55.0	61.0	55.3	62.0	54.5	77.0	68.5	77.0	58.0	66.0	69.0	60.7	60.0
	GPT-4	60.0	**67.5**	**71.5**	**67.8**	**67.0**	**66.5**	79.5	73.3	77.5	63.3	**71.0**	**84.0**	71.1	**70.4**
	α-UMi (7B)	65.0	59.5	68.0	66.0	64.0	57.0	81.5	**76.5**	76.5	72.0	70.0	63.0	70.9	65.9
	α-UMi (13B)	**65.5**	61.5	69.0	66.0	65.0	62.5	**84.5**	75.0	**81.0**	**74.5**	**71.0**	66.0	**72.2**	67.7

Table 2: Results of real-time evaluation on ToolBench. "ReACT" and "DFSDT" denote reasoning strategies used to construct agents, as detailed in Section 5.2. "Win" measures the relative win rate of each agent compared to ChatGPT-ReACT ("Method"=ReACT, "Model"=ChatGPT), which does not have an associated win rate.

that employing full fine-tuning is necessary when constructing multi-LLM frameworks.

Instruction Reusing: α-UMi$_{w/o\ \text{reuse}}$ represents that instead of reusing the user instructions in the first fine-tuning stage of GLPFT, a new set of user instructions are employed for the second stage of GLPFT. Previous works(Chung et al., 2022) has demonstrated that increasing the diversity of user instructions during fine-tuning can improve the performance and generalizability of LLMs. However, as presented in Table 1 and visualized in Figure 4, despite the increased diversity of instructions compared to α-UMi$_{\text{w/ reuse}}$, α-UMi$_{\text{w/o reuse}}$ does not outperform α-UMi$_{\text{w/ reuse}}$. We attribute this unexpected result to the following explanation: Since the objectives of the two training stages are different, using distinct sets of user instructions, each with its unique distribution, may introduces a harmful inductive bias that solving one group of the instructions in single-LLM format while the other group in multi-LLM format. In contrast, through the reuse of user instructions, the impact of varying distributions from different sets is mitigated.

5.2 Real-Time Test on ToolBench

To assess the performance of LLMs for solving real tasks via RapidAPI, we follow the ToolEval method (Qin et al., 2023b) proposed by the ToolBench team to conduct a real-time evaluation on the test set of ToolBench. The LLMs under consideration include Claude-2 (Anthropic, 2023), ChatGPT, GPT-4, and ToolLLaMA. We apply two reasoning strategies for these LLMs to construct tool learning agents: the ReACT method, as introduced in Section 3.1, and the Depth First Search-based Decision Tree (DFSDT) (Qin et al., 2023b), which empowers the agent to evaluate and select between different execution paths. Two metrics are included to measure these LLMs' performance: *pass rate*, which calculates the percentage of tasks successfully completed, and *win rate*, which compares the agent's solution path with that of the standard baseline, ChatGPT-ReACT. The above two metrics are assessed by a ChatGPT evaluator with carefully crafted criteria. The empirical results presented in Table 2 demonstrate that our α-UMi (7B) surpasses both ChatGPT and ToolLLaMA by significant margins in terms of *pass rate* (+6.1 and +10.2, respectively) and *win rate* (+1.6 and +5.9, respectively). While α-UMi underperforms GPT-4 in *win rate*, it exhibits *pass rates* on par with GPT-4 or even exceeds it in certain test groups such as *I1-Inst.* and *I2-Inst.*. Combining the findings from Section 5.1 and this section, we note that our multi-LLM agent outperforms several established baselines across diverse metrics on ToolBench, validating its efficacy.

5.3 Data Scaling Law

To assess the impact of the amount of training data on performance, we conduct a data scaling law analysis with the 7B backbone on ToolBench, varying the number of annotated training instances from 12.1k to 62.7k. The results in different metrics are depicted in Figure 4[5]. Several observations can be drawn from the results. Firstly, when comparing α-UMi (solid red curves) to Single-LLM (solid blue curves), there are significant and consistent enhancements in metrics such as Plan ACC, Act. EM, Hallu., and Arg. F1 across various scales of training data. While only minor improvements are observed in the R-L metric, which directly reflects the performance of the summarizer, this suggests

[5]The trend of Arg. F1 is similar to that of Act. EM., therefore its results are not displayed to save space. We have included the complete results in Figure 6 in Appendix.

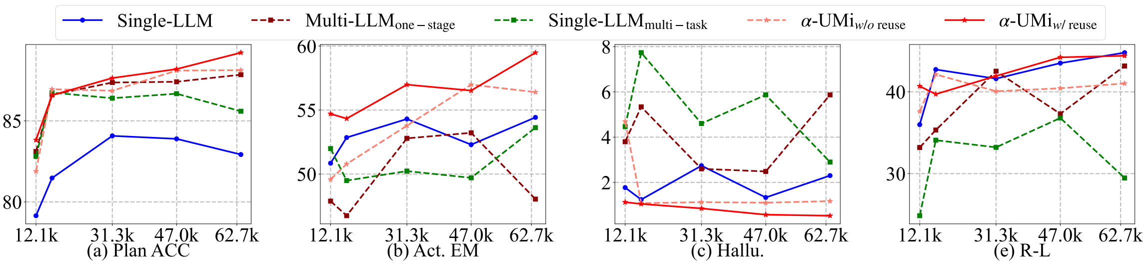

Figure 4: Results of data scaling law study on ToolBench with different evaluation metrics: (a) Plan ACC, (b) Act. EM, (c) Hallu, and (d) R-L. We randomly sampled five training sets with the scales of 12.1k, 15.7k, 31.3k, 47.0k, and 62.7k instances, accounting for 19.2%, 25%, 50%, 75%, and 100% of the training set, respectively.

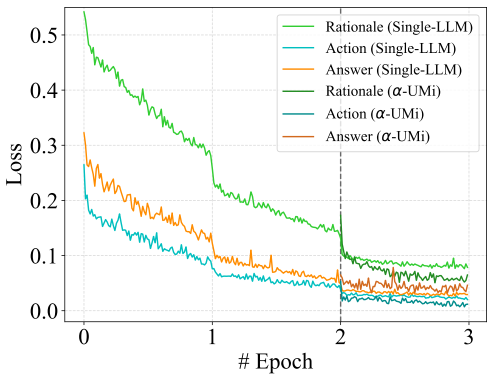

Figure 5: Curves of training loss.

that the performance enhancement of our framework is mainly attributed to the separation of the planner and the caller. Secondly, the performances of Multi-LLM$_{\text{one-stage}}$ and Single-LLM$_{\text{multi-task}}$ exhibit severe fluctuations in all metrics except for Plan ACC, indicating instability in training the framework through direct fine-tuning or multi-task fine-tuning. Thirdly, Single-LLM achieves optimal results in different metrics at different data scales. For example, it attains peak performance in Plan ACC with 31.3k instances and the best Arg. F1 and R-L with 62.7k instances. This suggests the challenge of obtaining a single LLM that uniformly performs well across all metrics. In contrast, the performance of our framework consistently improves with increased data scale across all metrics.

5.4 Why α-UMi Works?

We track the training process of our α-UMi approach to examine what makes it different from the Single-LLM baseline. To further investigate how each capability of the model evolves during training, we track the training loss on the rationale, action, and answer components of target responses. The results are depicted in Figure 5. As introduced in Section 4.3, α-UMi employs GLPFT and deviates from Single-LLM after two training epochs. Therefore, our discussion focuses on the training curves of α-UMi from the third epoch.

The plotted curves reveal a consistent decrease in the training loss for rationale, action, and answer components during the initial two epochs. However, in the third epoch, the losses of Single-LLM exhibit a nearly stagnant trend. In contrast, α-UMi experiences continued reductions in the losses associated with rationale and action, indicating further optimization within our α-UMi framework.

These observations suggest that the key factor contributing to the success of α-UMi lies in its ability to surpass the performance upper-bound of Single-LLM. This is achieved by leveraging GLPFT and decomposing the agent into a multi-LLM system, even after Single-LLM has attained its upper-bound abilities via sufficient fine-tuning.

6 Conclusion

The objective of this paper is to address the challenge of designing and fine-tuning a single small LLM to acquire the extensive abilities required for a tool learning agent. To this end, we introduce α-UMi, a multi-LLM tool learning agent framework that breaks down the tool learning task into three distinct sub-tasks delegated to three small LLMs: planner, caller, and summarizer. Moreover, we propose a global-to-local progressive fine-tuning strategy and demonstrate its effectiveness in training the multi-LLM framework. We evaluate our approach against single-LLM baselines on four tool learning benchmarks, supplemented by various in-depth analyses, including a data scaling law experiment. Our findings highlight the significance of our proposed method, validating that the system's design for decomposing tool learning tasks and the progressive fine-tuning strategy contribute to enhancing the upper-bound ability of a single LLM. Besides, we acknowledge the potential to utilize small LLMs to surpass the capabilities of an agent framework that relies on a single, larger LLM.

7 Limitations

While our framework has been demonstrated to outperform the single-LLM framework in tool learning tasks, there are still some limitations to this work. Firstly, there are additional avenues for exploration, such as integrating small LLMs with a powerful closed-source LLM like GPT-4 to create a "large + small" collaborative multi-LLM tool learning agent. Secondly, our framework could be further optimized to enhance its flexibility and applicability to a wider range of agent tasks.

8 Ethical Statement

The α-UMi framework is trained on the public ToolBench and ToolAlpaca benchmarks, with their original purpose being to enhance the tool invocation capabilities of LLMs and improve their performance in assisting users to complete tasks. This framework has not been trained on any data that poses ethical risks. The backbone model it uses, LLaMA-2-chat, has undergone safety alignment.

References

Anthropic. 2023. Claude-2. Website. https://www.anthropic.com/news/claude-2.

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*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

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

Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2023. How abilities in large language models are affected by supervised fine-tuning data composition. *arXiv preprint arXiv:2310.05492*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

Significant Gravitas. 2023. Autogpt: the heart of the open-source agent ecosystem. Github repository. https://github.com/Significant-Gravitas/Auto-GPT.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Chenliang Li, Hehong Chen, Ming Yan, Weizhou Shen, Haiyang Xu, Zhikai Wu, Zhicheng Zhang, Wenmeng Zhou, Yingda Chen, Chen Cheng, Hongzhu Shi, Ji Zhang, Fei Huang, and Jingren Zhou. 2023. Modelscope-agent: Building your customizable agent system with open-source large language models.

Yohei Nakajima. 2023. Babyagi. Github repository. https://github.com/yoheinakajima/babyagi.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

OpenAI. 2022. Chatgpt: Conversational ai language model. Website. https://openai.com/chatgpt.

OpenAI. 2023a. Gpt-4 code interpreter. Website. https://chat.openai.com/?model=gpt-4-code-interpreter.

OpenAI. 2023b. Gpt-4 technical report.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023a. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*.

Chen Qian, Yufan Dang, Jiahao Li, Wei Liu, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2023b. Experiential co-learning of software-developing agents.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. 2024. Autoact: Automatic agent learning from scratch via self-planning.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. 2023a. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*.

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

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE Computer Society.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *arXiv preprint arXiv:2303.17580*.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world restful apis.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Jimmy Wei, Kurt Shuster, Arthur Szlam, Jason Weston, Jack Urbanek, and Mojtaba Komeili. 2023. Multi-party chat: Conversational agents in group settings with humans and models. *arXiv preprint arXiv:2304.13835*.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023a. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023b. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Linyao Yang, Hongyang Chen, Zhao Li, Xiao Ding, and Xindong Wu. 2023a. Chatgpt is not enough: Enhancing large language models with knowledge graphs for fact-aware language modeling. *arXiv preprint arXiv:2306.11489*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023b. Gpt4tools: Teaching large language model to use tools via self-instruction. *arXiv preprint arXiv:2305.18752*.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023c. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Qin Yujia, Yankai Lin Shengding Hu, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, et al. 2023. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. 2023. Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

A System prompts

A.1 $\mathcal{P}_{\text{plan}}$ for ToolBench and ToolAlpaca

```
You have assess to the following apis:
{doc}
The conversation history is:
{history}
You are the assistant to plan what
to do next and whether is caller's or
conclusion's turn to answer.
Answer with a following format:
The thought of the next step, followed by
Next: caller or conclusion or give up.
```

A.2 $\mathcal{P}_{\text{call}}$ for ToolBench and ToolAlpaca

```
You have assess to the following apis:
{doc}
The conversation history is:
{history}
The thought of this step is:
{thought}
Base on the thought make an api call in
the following format:
Action: the name of api that should be
called in this step, should be exactly in
{tool_names},
Action Input: the api call request.
```

A.3 $\mathcal{P}_{\text{sum}}$ for ToolBench and ToolAlpaca

```
Make a conclusion based on the
conversation history:
{history}
```

A.4 $\mathcal{P}_{\text{plan}}$ for MATH and GSM8K

```
Solve the math problem step by step by
integrating step-by-step reasoning and
Python code,
The problem is: {instruction}
The historical execution logs are:
{history} You are the assistant to plan
what to do next, and shooce caller to
generate code or conclusion to answer the
problem.
Answer with a following format:
The thought of the next step, followed by
Next: caller or conclusion.
```

A.5 $\mathcal{P}_{\text{call}}$ for MATH and GSM8K

```
The problem is: {instruction}
The historical execution logs are:
{history}
The thought of this step is:
```

```
{thought}
generate the code for this step
```

A.6 $\mathcal{P}_{\text{sum}}$ for MATH and GSM8K

```
The problem is: {instruction}
The historical execution logs are:
{history} Make a conclusion based on the
conversation history
```

B Details of Benchmarks

B.1 ToolBench

ToolBench (Qin et al., 2023b) is a benchmark for evaluating an agent's ability to call APIs. The ToolBench team collects 16,464 real-world APIs from RapidAPI and a total of 125,387 execution trajectories as the training corpus. We randomly sample 62,694 execution trajectories as the training set, and the average number of execution steps is 4.1.

The test set of ToolBench is divided into 6 groups, namely I1-instruction, I1-tool, I1-category, I2-instruction, I2-category, and I3-instruction. The groups whose name ends with "instruction" means the test instructions in these groups use the tools in the training set, which is the in-domain test data. Otherwise, the groups whose name ends with "tool" or "category" means the test instructions do not use the tools in the training set, which is the out-of-domain test data. Each group contains 100 user instructions, therefore the total in-domain test set contains 400 instructions, while the out-of-domain test set contains 200 instructions.

The original evaluation metrics in ToolBench are the pass rate and win rate judged by ChatGPT. However, as introduced in Section 4.2, the APIs in RapidAPI update every day, which can cause network block, API breakdown, and exhausted quota. Therefore, to make a relatively fair comparison, we adopt the idea of Modelscope-Agent (Li et al., 2023) to compare the predictions of our model with the annotated GPT-4 outputs on the step level. Specifically, for the tth step, we input the model with the previous trajectory of GPT-4, ask our framework to generate the rationale and action of this step, and then compare the generated rationale and action of this step with the output of GPT-4.

C Static Evaluation on ToolBench

The evaluation method for ToolBench introduced in Section 4.2 is a static approach that assesses the output of the agent at each step individually. Specifically, for each step t, given the ground-truth annotation of the previous execution trajectory $\tau^*_{<t}$, the agent generates the rationale $\hat{r}_t$ and action $\hat{a}_t$ for this step:

$$\hat{r}_t, \hat{a}_t = \text{Agent}(\tau^*_{<t}). \tag{5}$$

Then, metrics are computed by comparing the generated $\hat{r}_t$ and $\hat{a}_t$ with the annotated ground-truth rationale r^*_t and action a^*_t for this step:

$$\text{Metric} = \text{Evaluate}(\hat{r}_t, \hat{a}_t, r^*_t, a^*_t). \tag{6}$$

The advantage of this evaluation method is as follows. At each step, the agent only needs to take the previous ground-truth trajectory as input and outputs the current step's rationale and action. This prevents error propagation due to factors such as network blocks, API breakdowns, and exhausted quotas in any particular step, which could affect the fairness of comparison. This evaluation method is an effective complement to real-time evaluation.

Model	Storage	Train Flops	Train Time	Train GPU Mem.	Infer. Time (Per Inst.)
Model Size = 7B					
Single-LLM	7B	$4.8 * 10^{15}$	41.54h	206G	6.41s
α-UMi	7B*3	$6.2 * 10^{15}$	63.34h	206G	6.27s
Model Size = 13B					
Single-LLM	13B	$7.2 * 10^{15}$	89.56h	308G	11.91s
α-UMi	13B*3	$9.7 * 10^{15}$	129.96h	308G	11.09s

Table 3: The cost of training and inference.

C.1 Cost of α-UMi

Given that α-UMi operates as a multi-LLM framework, it introduces potential additional costs in terms of training, storage, and deployment. Table 3 provides a summary of the costs associated with Single-LLM and α-UMi, based on execution logs on 8 Nvidia A100 GPUs with a 40G capacity. Our observations are threefold. Firstly, owing to its composition of a planner, a caller, and a summarizer, α-UMi demands three times the storage capacity compared to the Single-LLM framework, assuming they employ backbones of the same size. Secondly, the training of α-UMi requires 1.3 times the computational resources and 1.5 times the training duration compared to Single-LLM, while the GPU memory cost for training remains consistent between the two methods. Thirdly, during inference, the time required for both Single-LLM and α-UMi is similar, as we only distribute sub-tasks (rationale, action, and answer) to the three LLMs, without forcing them to generate extra contents, thus bringing nearly no extra cost when inference.

Note that based on the findings presented in Table 1, α-UMi with a 7B backbone can outperform

Model	ToolAlpaca Proc.	ToolAlpaca Ans.	MATH ACC	GSM8K ACC
Model Size = 7B				
Single-LLM	11	11	17.38	37.90
Multi-LLM$_{\text{one-stage}}$	2	9	15.46	38.96
Single-LLM$_{\text{multi-task}}$	28	18	14.18	27.97
α-UMi	**41**	**38**	**25.60**	**49.73**
Model Size = 13B				
Single-LLM	33	29	20.26	44.88
Multi-LLM$_{\text{one-stage}}$	22	19	20.32	44.57
Single-LLM$_{\text{multi-task}}$	28	16	15.34	34.79
α-UMi	**41**	**35**	**28.54**	**54.20**

Table 4: Overall results on ToolAlpaca, MATH and GSM8K.

Single-LLM with a 13B backbone. Furthermore, the cost associated with α-UMi featuring a 7B model is lower than that of Single-LLM featuring a 13B model, both in terms of training and inference. This underscores the cost-effectiveness of α-UMi as a means to achieve, and even surpass, the performance of a larger LLM.

D Case Study

Figure 7 and Figure 8 show two cases of our α-UMi executing real tasks in ToolBench. In the case of Figure 7, the user specifies the available tools in the instructions, making the tool invocation process simpler. The α-UMi framework completes the task within two steps through the collaboration of the planner, caller, and summarizer. In the case of Figure 8, α-UMi initially attempts to use the "video_for_simple_youtube_search" to obtain detailed video information at step 0. However, it realizes that this API has broken and cannot be invoked. Therefore, the planner informs the caller to try an alternative API and obtain accurate information. Ultimately, the user's task is successfully resolved.

To further analyze the specific advantages of our α-UMi and Single-LLM frameworks in task execution, we have presented some comparative examples of the two frameworks in Tables 5, 6, 7, and 8. Tables 5 and 6 illustrate simple tasks that require only a single step tool invocation to be completed, in which case both α-UMi and Single-LLM can successfully accomplish the tasks. However, in the complex tasks presented in Tables 7 and 8, where the tasks require the models to accomplish some composite objectives, α-UMi's planner can quickly understand the user's intentions and plan out steps based on the prompts provided by the caller and summarizer. On the other hand, Single-LLM exhibited some behaviors that did not align with the user's intentions during planning, such as invoking APIs that did not match the intent and entering loops in these misaligned APIs, ultimately failing to provide sufficient information to complete the user's instructions. This result indicates that α-UMi's decomposing Single-LLM into a planner, caller, and summarizer reduces the burden on the model during reasoning, allowing the planner model to focus solely on understanding the user's intentions and making effective plans, thereby better accomplishing the tasks.

E α-UMi on Other Benchmarks

Apart from ToolBench, we also evaluate α-UMi on ToolAlpaca (Tang et al., 2023), MATH (Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021).

ToolAlpaca is another benchmark for evaluating API calling. Unlike ToolBench, the APIs and API calling results in ToolAlpaca are mocked from ChatGPT by imitating how the real APIs work. The total number of training instances in ToolAlpaca is 4098, with an average of 2.66 execution steps per instance. The test set of ToolAlpaca contains 100 user instructions. The evaluation of ToolAlpaca is carried out by a simulator where the agent solves the instruction with the tools mocked by ChatGPT. Finally, GPT-4 judges if the execution process of the agent is consistent with the reference process pre-generated by ChatGPT (Proc. correctness) and whether the final answer generated by the agent can solve the user instruction (Ans. correctness).

The MATH (Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021) benchmarks are originally designed to test the mathematical reasoning ability of LLMs. Following ToRA (Gou et al., 2023), we employ a program-aided agent to solve the mathematical problems presented in these datasets. In our scenario, the planner will generate certain rationales and comments to guide the generation of program, the caller will generate program to conduct mathematical calculation, and finally the summarizer will conclude the final answer. Since ToRA has not released its training data, to facilitate the training of our framework, we utilize `gpt-3.5-turbo-1106` (OpenAI, 2022) and `gpt-4` (OpenAI, 2023b) to collect execution trajectories in the training set of MATH and GSM8K and filter out the trajectories that do not lead to the correct final answer. Finally, we collect 5536 trajectories from ChatGPT, 573 trajectories from GPT-4 on MATH, and 6213 from ChatGPT on GSM8K.

The test set sizes of MATH and GSM8K are 5000 and 1319, respectively. During testing, we feed our agent with each of the test instructions and execute the agent with a Python code interpreter. We follow the original evaluation methods of MATH and GSM8K to evaluate the performance of the agent with the accuracy of the final answer.

As the evaluation results shown in Table 4, our α-UMi can still outperform the baselines on ToolAlpaca, MATH and GSM8K, verifying its effectiveness.

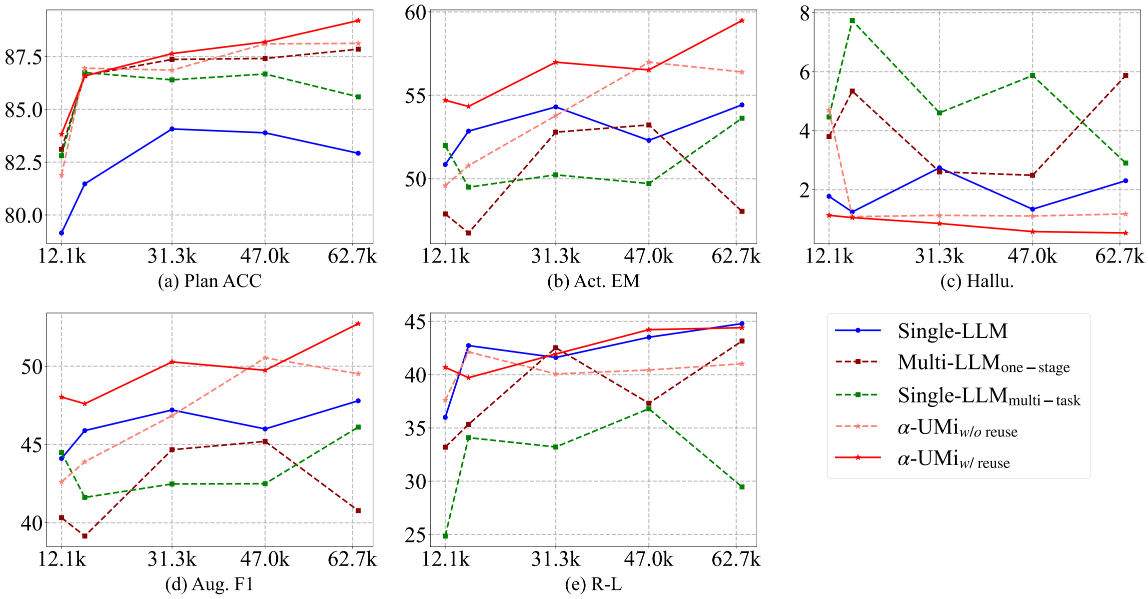

Figure 6: Results of data scaling law study on ToolBench with different evaluation metrics: (a) Plan ACC, (b) Act. EM, (c) Hallu, (d) Arg. F1, and (e) R-L. We randomly sampled five training sets with the scales of 12.1k, 15.7k, 31.3k, 47.0k, and 62.7k instances, accounting for 19.2%, 25%, 50%, 75%, and 100% of the training set, respectively.

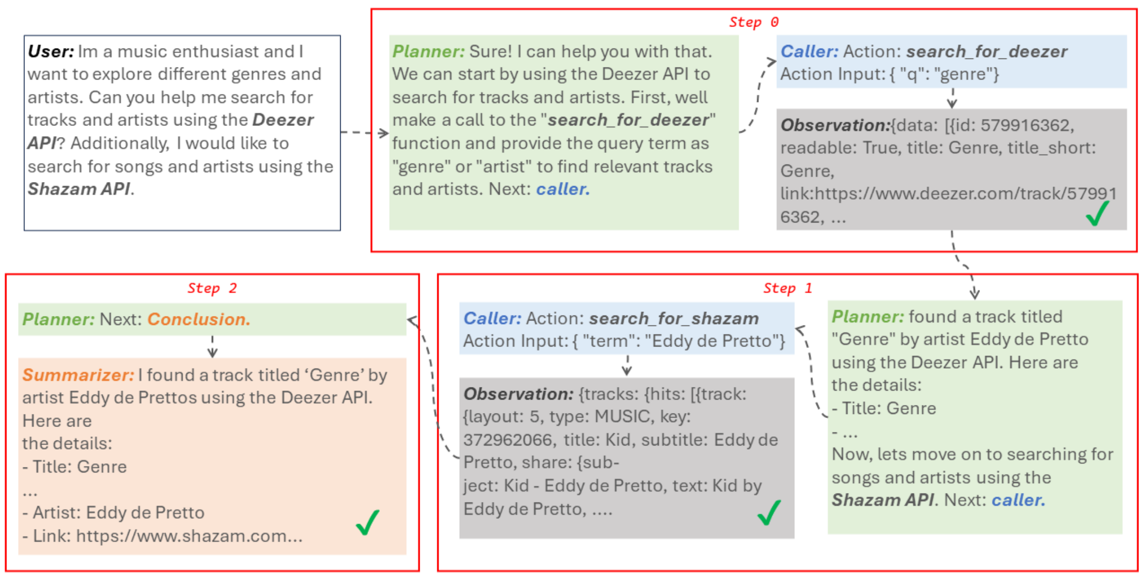

Figure 7: A case study of α-UMi. In this case, the user specifies the available tools in the instructions, making the tool invocation process simpler. The α-UMi framework completes the task within two steps through the collaboration of the planner, caller, and summarizer.

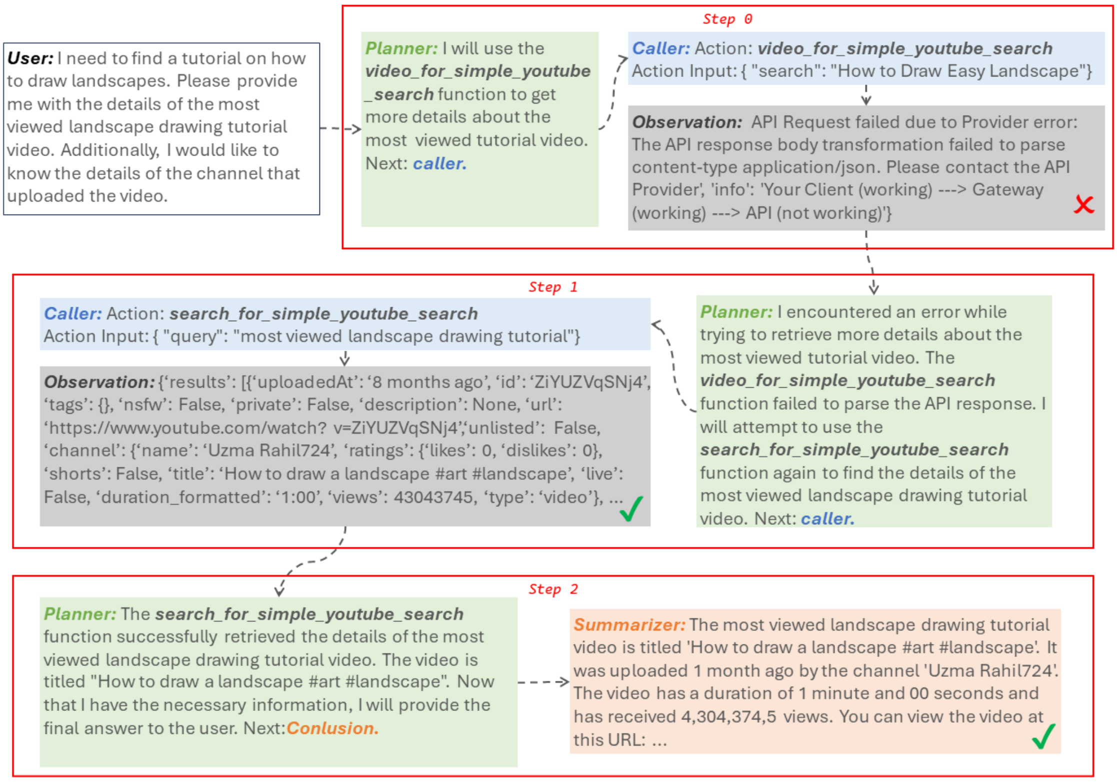

Figure 8: A case study of α-UMi, in which α-UMi initially attempts to use the "video_for_simple_youtube_search" to obtain detailed video information at step 0. However, it realizes that this API has broken and cannot be invoked. Therefore, the planner informs the caller to try an alternative API and obtain accurate information. Ultimately, the user's task is successfully resolved.

Case #1 of α-UMi In ToolBench	
Instruction: I want to explore the catalog of products available in the 'Electronics' category. Please provide me with the list of products along with their details.	
Planner	Based on the user's request to explore the catalog of products in the 'Electronics' category, I need to call the 'get_products_in_category_for_demo_project_v13' function with the arguments 'limit': 10, 'skip': 0, and 'category': 'Electronics'. This will allow me to retrieve the list of products along with their details specifically from the 'Electronics' category. By calling this function, I can fulfill the user's request and provide them with the desired information. Next: caller.
Caller	Action: get_products_in_category_for_demo_project_v13 Action Input: "{ "limit": 10, "skip": 0, "category": "Electronics"}"
Observation	{'products': [{'id': 893292, 'name': 'TV', 'category': 'electronics'}, {'id': 893293, 'name': 'TV', 'category': 'electronics'}, {'id': 894481, 'name': 'Gorgeous Plastic Bench', 'category': 'Electronics'}, {'id': 896484, 'name': 'Fantastic Rubber Gloves', 'category': 'Electronics'}, {'id': 912179, 'name': 'daniel', 'category': 'electronics'}, {'id': 916400, 'name': 'Tasty Frozen Pants', 'category': 'Electronics'}, {'id': 917106, 'name': 'Tasty Wooden Chips', 'category': 'Electronics'}, {'id': 917821, 'name': 'Pizza', 'category': 'Electronics'}, {'id': 917875, 'name': 'Tuna', 'category': 'Electronics'}, {'id': 917998, 'name': 'Intelligent Frozen Pizza', 'category': 'Electronics'}], 'summary': {'count': 105}}
Planner	I have retrieved the list of products available in the 'Electronics' category. Here are some of the products along with their details: 1. Product ID: 893292, Name: TV, Category: Electronics 2. Product ID: 893293, Name: TV, Category: Electronics 3. Product ID: 894481, Name: Gorgeous Plastic Bench, Category: Electronics 4. Product ID: 896484, Name: Fantastic Rubber Gloves, Category: Electronics 5. Product ID: 912179, Name: daniel, Category: Electronics 6. Product ID: 916400, Name: Tasty Frozen Pants, Category: Electronics 7. Product ID: 917106, Name: Tasty Wooden Chips, Category: Electronics 8. Product ID: 917821, Name: Pizza, Category: Electronics 9. Product ID: 917875, Name: Tuna, Category: Electronics 10. Product ID: 917998, Name: Intelligent Frozen Pizza, Category: Electronics Next: conclusion.
Summarizer	Here are some of the products available in the 'Electronics' category: 1. Product ID: 893292, Name: TV, Category: Electronics 2. Product ID: 893293, Name: TV, Category: Electronics 3. Product ID: 894481, Name: Gorgeous Plastic Bench, Category: Electronics 4. Product ID: 896484, Name: Fantastic Rubber Gloves, Category: Electronics 5. Product ID: 912179, Name: daniel, Category: Electronics 6. Product ID: 916400, Name: Tasty Frozen Pants, Category: Electronics 7. Product ID: 917106, Name: Tasty Wooden Chips, Category: Electronics 8. Product ID: 917821, Name: Pizza, Category: Electronics 9. Product ID: 917875, Name: Tuna, Category: Electronics 10. Product ID: 917998, Name: Intelligent Frozen Pizza, Category: Electronics

Table 5: A case of α-UMi in ToolBench.

Case #1 of Single-LLM In ToolBench

Instruction: I want to explore the catalog of products available in the 'Electronics' category. Please provide me with the list of products along with their details.	
Assistant	Based on the previous actions, it seems that the function "get_categories_for_demo_project_v13" was called to retrieve the categories, but it returned an error. To proceed differently, I will call the function "get_products_in_category_for_demo_project_v13" with the category set to "Electronics" to fetch the products in the Electronics category. Action: get_products_in_category_for_demo_project_v13 Action Input: { "limit": 10, "skip": 0, "category": "Electronics" }
Observation	{'products': [{'id': 893292, 'name': 'TV', 'category': 'electronics'}, {'id': 893293, 'name': 'TV', 'category': 'electronics'}, {'id': 894481, 'name': 'Gorgeous Plastic Bench', 'category': 'Electronics'}, {'id': 896484, 'name': 'Fantastic Rubber Gloves', 'category': 'Electronics'}, {'id': 912179, 'name': 'daniel', 'category': 'electronics'}, {'id': 916400, 'name': 'Tasty Frozen Pants', 'category': 'Electronics'}, {'id': 917106, 'name': 'Tasty Wooden Chips', 'category': 'Electronics'}, {'id': 917821, 'name': 'Pizza', 'category': 'Electronics'}, {'id': 917875, 'name': 'Tuna', 'category': 'Electronics'}, {'id': 917998, 'name': 'Intelligent Frozen Pizza', 'category': 'Electronics'}], 'summary': {'count': 105}}
Assistant	I have retrieved the list of products available in the 'Electronics' category. Here are some of the products along with their details: 1. Product ID: 893292, Name: TV, Category: Electronics 2. Product ID: 893293, Name: TV, Category: Electronics 3. Product ID: 894481, Name: Gorgeous Plastic Bench, Category: Electronics 4. Product ID: 896484, Name: Fantastic Rubber Gloves, Category: Electronics 5. Product ID: 912179, Name: daniel, Category: Electronics 6. Product ID: 916400, Name: Tasty Frozen Pants, Category: Electronics 7. Product ID: 917106, Name: Tasty Wooden Chips, Category: Electronics 8. Product ID: 917821, Name: Pizza, Category: Electronics 9. Product ID: 917875, Name: Tuna, Category: Electronics 10. Product ID: 917998, Name: Intelligent Frozen Pizza, Category: Electronics

Table 6: A case of Single-LLM in ToolBench.

Case #2 of α-UMi In ToolBench	
Instruction: I am planning a vacation with my family and we are interested in scuba diving. Can you recommend some dive sites in Thailand? Additionally, we would like to know the top-tier stays available in that area for accommodation options.	
Planner	Based on the user's request, I need to recommend dive sites in Thailand and provide information about the top-tier stays in the area. To accomplish this, I will call the "esites_by_a_country_or_a_region_for_world_scuba_diving_sites_api" function with the argument "country" set to "Thailand". This will allow me to retrieve the relevant dive sites and accommodation options specifically for Thailand. By using this API, I can efficiently gather the necessary information for the user's vacation planning. Next: caller.
Caller	Action: esites_by_a_country_or_a_region_for_world_scuba_diving_sites_api Action Input: "{ "country": "Thailand" }"

Observation	{'data': [{'id': '0126ccc3-6d33-4cb9-a6dd-1519273acf43', 'name': 'The Rock', 'region': 'Racha Noi Island, Phuket', 'lat': '7.552420449639636', 'lng': '98.33433151245119', 'ocean': 'Indian Ocean, Andaman Sea, Strait Of Malacca', 'Location': 'Asia, Thailand, Southern Thailand, Phuket, Racha Noi Island'}, {'id': '01c6e49f-45ab-4bb4-bfee-eff9c67f9d22', 'name': 'Paradise Reef', 'region': 'Phuket, Southern Thailand', 'lat': '7.893819147272116', 'lng': '98.2584571838379', 'ocean': 'Indian Ocean, Andaman Sea, Strait Of Malacca', 'Location': 'Asia, Thailand, Southern Thailand, Phuket'}, {'id': '02446f8e-9466-4cec-8a7e-940fa2946c44', 'name': 'Shark Fin Reef', 'region': 'Payang Island, Similan Islands', 'lat': '8.50966017107524', 'lng': '97.66388654708864', 'ocean': 'Indian Ocean, Andaman Sea', 'Location': 'Asia, Thailand, Southern Thailand, Phang Nga, Similan Islands, Payang Island'}, {'id': '02dea021-ef79-4ee4-929a-46dfb03e5f6e', 'name': 'Bang Bao Reef', 'region': 'Kut Island, Trat', 'lat': '11.612304159956192', 'lng...
Planner	Based on the user's request, I need to recommend dive sites in Thailand and provide information about the top-tier stays in the area. To accomplish this, I will call the "get_top_tier_stays_filter_for_airbnb_v2" function. Since the user specifically mentioned the need for top-tier accommodations, I will call this function without any arguments to retrieve the relevant options. By using this API, I can efficiently gather the necessary information for the user's vacation planning and ensure they have a great experience during their trip. Next: caller.
Caller	Action: get_top_tier_stays_filter_for_airbnb_v2 Action Input: "{}"
Observation	{'status': True, 'message': 'Success', 'timestamp': 1688973421406, 'data': [{'id': '1', 'title': 'Airbnb Plus', 'subtitle': 'Every Plus home is reviewed for quality'}, {'id': '2', 'title': 'Airbnb Luxe', 'subtitle': 'Handpicked luxury homes with personally arranged services'}]}
Planner	The user has requested recommendations for dive sites in Thailand and information about top-tier stays in the area. To fulfill their request, I called the "esites_by_a_country_or_a_region_for_world_scuba_diving_sites_api" function with the argument "country" set to "Thailand". This API allows me to efficiently retrieve relevant dive sites and accommodation options specifically for Thailand. By using this function, I can provide the user with accurate and up-to-date information for their vacation planning. Additionally, I called the "get_top_tier_stays_filter_for_airbnb_v2" function to retrieve the available options for top-tier accommodations. By using these APIs, I can ensure that the user has a great experience during their trip. Next: conclusion.
Summarizer	Here are some dive sites in Thailand: The Rock in Racha Noi Island, Phuket; Paradise Reef in Phuket, Southern Thailand; Shark Fin Reef in Payang Island, Similan Islands; Bang Bao Reef in Kut Island, Trat. For accommodation options, you can consider Airbnb Plus and Airbnb Luxe, which offer high-quality and luxury homes respectively. Enjoy your scuba diving adventures in Thailand!

Table 7: A case of α-UMi in ToolBench.

Case #2 of Single-LLM In ToolBench
Instruction: I am planning a vacation with my family and we are interested in scuba diving. Can you recommend some dive sites in Thailand? Additionally, we would like to know the top-tier stays available in that area for accommodation options.

Assistant	Based on the previous action, it seems that the API call to get dive sites in Thailand was not successful. To generate a different action, I will try using the "ranked_world_crime_cities_for_ranked_crime_cities" function to get a list of ranked crime cities. This will help me identify if Thailand is listed as a crime-free city. Action:ranked_world_crime_cities_for_ranked_crime_cities Action Input: {}
Observation	{’products’: [{’id’: 893292, ’name’: ’TV’, ’category’: ’electronics’}, {’id’: 893293, ’name’: ’TV’, ’category’: ’electronics’}, {’id’: 894481, ’name’: ’Gorgeous Plastic Bench’, ’category’: ’Electronics’}, {’id’: 896484, ’name’: ’Fantastic Rubber Gloves’, ’category’: ’Electronics’}, {’id’: 912179, ’name’: ’daniel’, ’category’: ’electronics’}, {’id’: 916400, ’name’: ’Tasty Frozen Pants’, ’category’: ’Electronics’}, {’id’: 917106, ’name’: ’Tasty Wooden Chips’, ’category’: ’Electronics’}, {’id’: 917821, ’name’: ’Pizza’, ’category’: ’Electronics’}, {’id’: 917875, ’name’: ’Tuna’, ’category’: ’Electronics’}, {’id’: 917998, ’name’: ’Intelligent Frozen Pizza’, ’category’: ’Electronics’}], ’summary’: {’count’: 105}}
Assistant	Based on the previous action, it seems that the API call to get dive sites in Thailand was not successful. To generate a different action, I will try using the "get_city_by_filter_for_ranked_crime_cities" function to get the details of a specific city in Thailand. This will help me determine if Thailand is listed as a crime-free city. Action: get_city_by_filter_for_ranked_crime_cities Action Input: {"id":1}
Observation	{"message": "Function executing from my_tools.Travel.ranked_crime_cities.api import get_city_by_filter error... get_city_by_filter() got an unexpected keyword argument ’id’"}
Assistant	Based on the current state and the previous actions, it seems that the API calls to get dive sites in Thailand have been unsuccessful. To generate a different action, I will try using the "esites_by_a_country_or_a_region_for_world_scuba_diving_sites_api" function to get a list of dive sites in Thailand. This might provide the desired information about dive sites in the country. Action: esites_by_a_country_or_a_region_for_world_scuba_diving_sites_api Action Input: {"country": "Thailand"}
Observation	{’data’: [{’id’: ’0126ccc3-6d33-4cb9-a6dd-1519273acf43’, ’name’: ’The Rock’, ’region’: ’Racha Noi Island, Phuket’, ’lat’: ’7.552420449639636’, ’lng’: ’98.33433151245119’, ’ocean’: ’Indian Ocean, Andaman Sea, Strait Of Malacca’, ’Location’: ’Asia, Thailand, Southern Thailand, Phuket, Racha Noi Island’}, {’id’: ’01c6e49f-45ab-4bb4-bfee-eff9c67f9d22’, ’name’: ’Paradise Reef’, ’region’: ’Phuket, Southern Thailand’, ’lat’: ’7.893819147272116’, ’lng’: ’98.2584571838379’, ’ocean’: ’Indian Ocean, Andaman Sea, Strait Of Malacca’, ’Location’: ’Asia, Thailand, Southern Thailand, Phuket’}, {’id’: ’02446f8e-9466-4cec-8a7e-940fa2946c44’, ’name’: ’Shark Fin Reef’, ’region’: ’Payang Island, Similan Islands’, ’lat’: ’8.50966017107524’, ’lng’: ’97.66388654708864’, ’ocean’: ’Indian Ocean, Andaman Sea’, ’Location’: ’Asia, Thailand, Southern Thailand, Phang Nga, Similan Islands, Payang Island’}, {’id’: ’02dea021-ef79-4ee4-929a-46dfb03e5f6e’, ’name’: ’Bang Bao Reef’, ’region’: ’Kut Island, Trat’, ’lat’: ’11.612304159956192’, ’lng...

Assistant	Based on the current state and the previous actions, it seems that the API calls to get dive sites in Thailand have been successful. I have retrieved a list of dive sites in Thailand, including The Rock in Racha Noi Island, Phuket, Paradise Reef in Phuket, Southern Thailand, and Shark Fin Reef in Payang Island, Similan Islands. These dive sites offer potential accommodation options for your family vacation. Now, I will proceed to provide you with the final answer. conclusion:Here are some dive sites in Thailand that you might find interesting: 1. The Rock - Racha Noi Island, Phuket 2. Paradise Reef - Phuket, Southern Thailand 3. Shark Fin Reef - Payang Island, Similan Islands 4. Bang Bao Reef - Kut Island, Trat These dive sites offer potential accommodation options for your family vacation. Enjoy your scuba diving adventure in Thailand!

Table 8: A case of Single-LLM in ToolBench.

Case In ToolAlpaca	
Instruction: I'm moving to a new apartment and need to update my address with various companies. Can you first verify if my new address is valid and then standardize it? Here's the address: 321 Maple Dr, Apt 12C, Houston, TX, 77002.	
Planner	I need to verify the address and then standardize it. Next: caller.
Caller	Action: verifyUSAddress Action Input: {"addressLine1": "321 Maple Dr", "addressLine2":"Apt 12C", "city": "Houston", "state": "TX", "zipCode": "77002"}
Observation	{"status":"valid","standardizedAddress":{"addressLine1":"123 Main St","addressLine2":"Apt 4B","city":"New York","state":"NY","zipCode":"10001"}}
Planner	The address is valid and has been successfully standardized. Next: conclusion.
Summarizer	The address you provided is valid and has been successfully standardized. The standardized address is 123 Main St, Apt 4B, New York, NY, 10001

Table 9: A case study of α-UMi in ToolAlpaca.

Case In MATH	
Instruction: Compute the range of the function $f(x) = sqrt{x**2}$.	
Planner	Next: caller.

Caller	from sympy import symbols, Interval, oo
	def function_range():
	    x = symbols('x')
	    f = sqrt(x**2)
	    range_f = Interval(0, oo)
	    return range_f
	result = function_range()
	print(result)

Observation	NameError: name 'sqrt' is not defined
Planner	The code provided for the solution to the given problem will result in an error. To fix this error, the 'sqrt' function needs to be imported from the 'sympy' library. Next: caller.

Caller	from sympy import symbols, Interval, oo, sqrt
	def function_range():
	    x = symbols('x')
	    f = sqrt(x**2)
	    range_f = Interval(0, oo)
	    return range_f
	result = function_range()
	print(result)

Observation	Interval(0, oo)
Planner	Next: conclusion.
Summarizer	The range of the function $f(x) = sqrt{x*2}$ is $boxed{[0, infty)}$.

Table 10: A case study of α-UMi in MATH.