Learning to Use Tools via Cooperative and Interactive Agents with Large Language Models

Anonymous ACL submission

Abstract

Tool learning empowers large language models (LLMs) as agents to use external tools and extend their utility. Existing methods employ one single LLM-based agent to iteratively select and execute tools, thereafter incorporating execution results into the next action prediction. Despite their progress, these methods suffer from performance degradation when addressing practical tasks due to: (1) the pre-defined pipeline with restricted flexibility to calibrate incorrect actions, and (2) the struggle to adapt a general LLM-based agent to perform a variety of specialized actions. To mitigate these problems, we propose ConAgents, a **Co**operative and i**n**teractive **Agents** framework, which coordinates three specialized agents for tool selection, tool execution, and action calibration separately. ConAgents introduces two communication protocols to enable the flexible cooperation of agents. To effectively generalize the ConAgents into open-source models, we also propose specialized action distillation, enhancing their ability to perform specialized actions in our framework. Our extensive experiments on three datasets show that the LLMs, when equipped with the ConAgents, outperform baselines with substantial improvement (*i.e.,* up to 14% higher success rate)[1].

1 Introduction

Although large language models (LLMs) have achieved remarkable performance in a broad range of natural language processing tasks (Wang et al., 2023c; Chang et al., 2023), they still encounter inherent limitations such as out-of-date information (Qin et al., 2023b; Mallen et al., 2023). **Tool learning** is proposed to equip LLMs with various auxiliary resources, *e.g.,* a search engine (Qin et al., 2023a; Nakano et al., 2021)

[1] https://anonymous/ConAgents

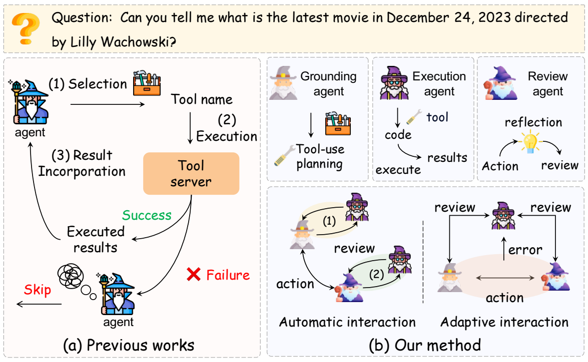

Figure 1: Comparison between (a) existing single-agent tool learning method and (b) our cooperative agent framework ConAgents. The ConAgents coordinates three agents through two proposed communication protocols, *e.g.,* automatic and adaptive interaction.

or a calculator (Schick et al., 2023; Gao et al., 2023a), which empower them as tool-use agents and improve their proficiency in tackling concrete complex tasks. As shown in Figure 1(a), most previous studies allow the LLM-based agent to interleave multiple actions in a pre-defined order to interact with tools (Yao et al., 2023; Yang et al., 2023b; Qin et al., 2024). The agent first breaks down the task and **`plans`** a series of tools in a step-by-step manner. For each step, the agent **`executes`** the tools by passing arguments and continuously **`incorporates`** useful intermediates into the next action prediction.

Despite the advancement of existing methods, they face two challenges in practice. **First**, most of them alternate the planning and execution with a pre-defined pipeline (Yang et al., 2023b; Song et al., 2023), which *inevitably constrains their flexibility in handling exceptional errors* that frequently occur during a tool-use workflow (Zhuang et al., 2023; Wang et al., 2023b; Prasad et al., 2023). When failing to invoke tools, it is crucial to enable agents to revise their incorrect actions instead of directly shifting to the next step with the error response of previous steps. **Second**, *it is struggling to adapt a single LLM-based agent to learn a variety of specialized actions in solving a task* (Dziri et al., 2023; Yin et al., 2023). Solving a practical task involves varied actions with substantial differences, *e.g.,* planning, execution, and reflection, drawing upon different facets of the LLMs (Shen et al., 2024; Qiao et al., 2024). Therefore, developing effective agent flow and adapting tool-use models to solve practical tasks remains a challenging research topic.

In this work, we propose ConAgents, a **Co**operative and **in**terative **Agents** framework for tool learning tasks. As illustrated in Figure 1, ConAgents decomposes the overall tool-use workflow using three specialized agents: *Grounding*, *Execution*, and *Review* agents. The grounding agent reasons the task description and grounds it into planning by specifying which tool to use. The execution agent follows the planning to execute the selected tool by generating executable code. The review agent reviews the incorrectness in planning or execution, providing feedback for revision. To enable the dynamic cooperation of these specialized agents, we propose two communication protocols, including automatic and adaptive interaction. In the process of *automatic interaction*, the review agent provides real-time reviews to calibrate incorrect actions. Thus, the agent flow alternates between the planning-review and execution-review phases as shown in Figure 1. In the process of *adaptive interaction*, the review agent only provides feedback when exceptional errors are captured while executing the tools.

For a comprehensive evaluation, we conduct experiments on two benchmarks, *i.e.,* ToolBench and RestBench, using various LLMs as backbones. We find that ConAgents outperforms the state-of-the-art baseline with both communication protocols (6% improvement in Success Rate on average).

Despite closed-source LLMs performing well with our framework, we find the open-source models may struggle with the modulized agent flow. Thus, we propose an approach called **sp**ecialized **a**ction distillatio**n** (SPAN), enhancing the performance of open-source models in ConAgents. We heuristically sample 2,919 high-quality tasks from the ToolBench (Qin et al., 2024) training set, and cluster them based on their similarity, retaining only one task in each cluster to avoid duplication. For each task, we guide the GPT-4 to generate solutions using ConAgents, and reorganize them into actions tailored to specialized agent functionalities in ConAgents. These actions are separately distilled into different student models through instruction tuning. We employ parameter-efficient tuning techniques, *i.e.,* LoRA (Hu et al., 2021), further extending our distillation method into low-resource scenarios. Experiment results show that our distillation method empowers open-source models with strong performance with only 500 training examples.

Our contributions are summarized as follows: (1) We propose ConAgents, a cooperative and interactive agents framework, for tool learning tasks. ConAgents coordinates three specialized agents with two communication protocols to solve a complex task. (2) We propose specialized action distillation (SPAN), which more effectively enables open-source models to work with the ConAgents; (3) Both automatic and human evaluation conducted on two benchmarks validate the superiority of ConAgents.

2 Related Work

LLMs for tool learning. Enhancing LLMs with external tools has been proven a promising method for solving practical tasks (Bran et al., 2023; Qu et al., 2024; Wang et al., 2024b). Previous works empower a tool-learning agent typically by supervised fine-tuning (Patil et al., 2023; Yang et al., 2023a; Gao et al., 2023b) or prompt learning (Lu et al., 2023; Shen et al., 2023). Specifically, the former trains LLMs on tool-use dataset (Wang et al., 2023c), teaching LLMs how to use tools from the data. The latter directly demonstrates tool usages to LLM using in-context examples (Paranjape et al., 2023; Kim et al., 2023). However, solving complex tasks with tools involves various actions, *e.g.,* deciding which tools to use, what arguments to pass, and how to utilize the results (Schick et al., 2023; Qiao et al., 2024). Compelling one single agent to learn all abilities places even greater pressure on it (Yin et al., 2023; Prasad et al., 2023). In addition, as the tasks become complex, LLMs-based agents struggle to incorporate lengthy task-solving contexts to predict the next actions correctly due to their limited working memory (Shi et al., 2023). In contrast, our proposed ConAgents coordinates three specialized agents, generating a solution through agent cooperation.

Multi-agent cooperation. Synergizing multiple agents has demonstrated strong performance on a variety of tasks (Liu et al., 2023; Sun et al., 2023; Zhang et al., 2023), enhancing the capabilities of individual agents (Talebirad and Nadiri, 2023; Mohtashami et al., 2023; Qian et al., 2023). Recent studies take multiple agents into a debate for a fixed number of rounds (Wang et al., 2023a; Liang et al., 2023), boosting their factuality (Cohen et al., 2023) and reasoning abilities (Du et al., 2023; Fu et al., 2023). In the tool learning tasks, recent work separately implements the task planning and execution with different agents, thereby reducing the workload of a single agent (Shen et al., 2024; Song et al., 2023; Qiao et al., 2024). Despite their progress, their agent flow is simplified into a predefined pipeline (Prasad et al., 2023), struggling to handle exceptional errors that frequently occur during the tool-use workflows (Zhuang et al., 2023; Wang et al., 2023b). In our work, we propose two communication protocols, which enable the action calibrations and dynamic cooperation of agents.

3 Methodology

3.1 Overall Framework

Our cooperative framework, ConAgents, is proposed to enable the dynamic cooperation of agents to solve complex tasks. As shown in Figure 2, ConAgents streamlines and modularizes the workflow of tool learning tasks into a grounding agent $\mathcal{M}_G$, execution agent $\mathcal{M}_E$, and review agent $\mathcal{M}_R$. These agents are implemented with different system prompt or learnable parameters. Given a complex task x, the $\mathcal{M}_G$ first decomposes x into simpler sub-tasks and generates tool-use planning t in a step-by-step manner. For each step, the $\mathcal{M}_E$ executes the selected tool by writing executable code following the planning t. The execution result r is then incorporated into the context of the grounding agent $\mathcal{M}_G$ to predict planning in the next iteration. The $\mathcal{M}_R$ is employed to simulate an expert to provide feedback to agent $\mathcal{M}_G$ and $\mathcal{M}_E$, guiding them to revise their incorrect planning or execution. To coordinate these three specialized agents, we explore and analyze two communication protocols, including the *automatic* and *adaptive* interactions.

3.2 Specialized Agents

Grounding Agent. The grounding agent is designed to break down an input task and generate a series of tool-use planing. At ith iteration, the grounding agent generates planning t_i on the condition of the task x and current trajectory $\mathcal{H}_i = \{(t_j, r_j)|j < i\}$, consisting of the accumulation of previous planning $t_{<i}$ and results $r_{<i}$. It can be formulated as:

$$t_i = \mathcal{M}_G(x, \mathcal{S}, \mathcal{H}_i), \tag{1}$$

where t_i contains a tool selected from the provided toolset S and necessary arguments to invoke the tool, such as "`Use the Bing search to find a movie shown on Dec 24, 2023`".

Execution Agent. Following the generated planning t_i, the execution agent $\mathcal{M}_E$ executes the selected tool by generating executable code c with the assistance of the tool documentation d. This process can be formulated as:

$$c_i = \mathcal{M}_E(d, t_i).$$

The execution result r_i is obtained by running the generated code c_i to request the data from the backend servers of tools, denoted as $r_i = \text{Execute}(c_i)$. When the tool fails to execute, the r_i indicates an error message as a failure signal. When the tool executes successfully, the result r_i contains the targeted information in response to the planning t_i.

Review Agent. Incorrect *planning* and *execution* are frequently observed during the tool-use workflow. The review agent $\mathcal{M}_R$ is employed as an expert, providing feedback to agent $\mathcal{M}_G$ and $\mathcal{M}_E$ for revision. Specifically, if the planning generated by $\mathcal{M}_G$ is vague or selects a non-existing tool, the agent $\mathcal{M}_R$ generates verbal feedback to instruct the $\mathcal{M}_G$ to reformulate planning. It can be formulated as:

$$f_{R\rightarrow G} = M_R(x, \mathcal{S}, t_i) \tag{2}$$

Similarly, if $\mathcal{M}_E$ hallucinates generating a wrong program to execute the tool, the agent $\mathcal{M}_R$ reviews execution results (or errors) and re-checks the tool documentation, providing instructions for calibration:

$$f_{R\rightarrow E} = M_R\left(x, d, c_i, r_i\right) \tag{3}$$

We denote the maximum turns of interaction between agent $\mathcal{M}_R$ and agent $\mathcal{M}_G$ (or $\mathcal{M}_E$) is denoted as α (or β). Their communication protocol and action flow are explained in § 3.3.

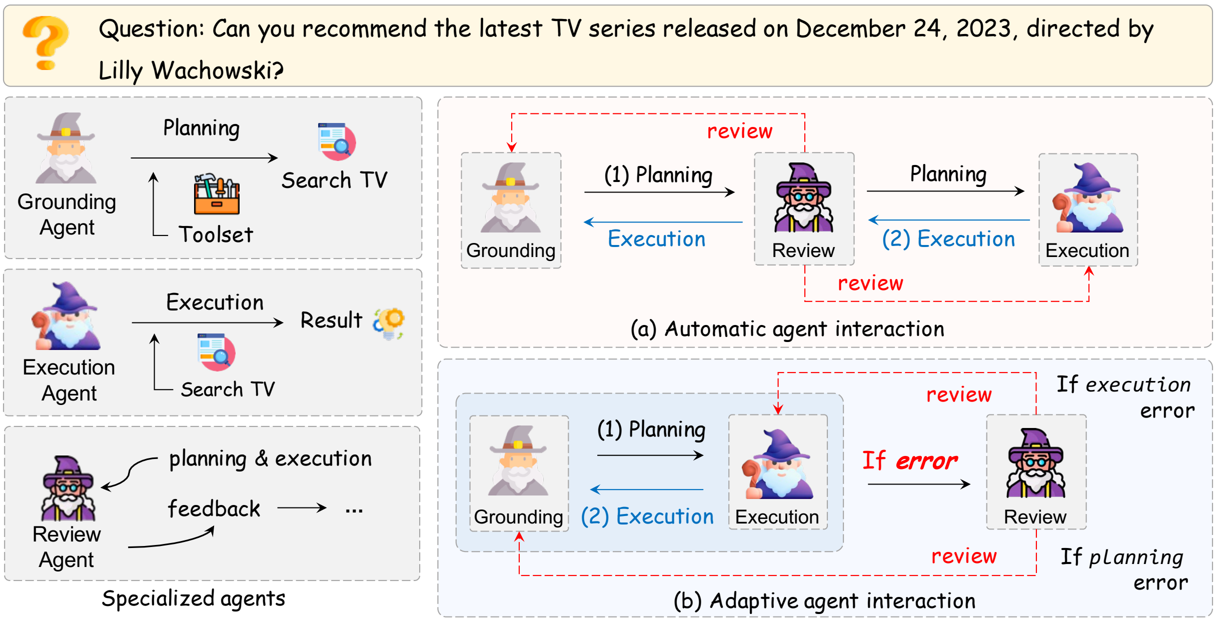

Figure 2: Our proposed cooperative and interactive agent framework. The **left** shows the three specialized agents in our framework (§ 3.1). The **right** illustrates two proposed communication protocols to coordinate these specialized agents, including the *automatic* and *adaptive* communication (§ 3.3).

3.3 Agent communication protocols

We propose two agent communication protocols, including *automatic* and *adaptive* interaction.

Automatic interaction. As illustrated in Figure 2, our automatic interaction alternates between *planning-review* and *execution-review* phases. For the ith step, it starts with the interaction between the agent $\mathcal{M}_G$ and $\mathcal{M}_R$ until a correct planning t_i is determined or up to the maximum turns α. Formally, it can be formulated as:

$$t_i^j = M_G(x, \mathcal{S}, \mathcal{H}_i, \underbrace{\{t_i^{<j}, f_{R\to G}^{<j}\}}_{\text{planning calibration}}) \qquad (4)$$

Here, j indicates jth interaction of two agents. Following the planning t, the agent $\mathcal{M}_E$ generates executable programs to execute the selected tool and calibrates the incorrect result r with the feedback of agent $\mathcal{M}_R$ for up to β turns. This process can be formulated as:

$$c_i^j = \mathcal{M}_E(t_i, d, \underbrace{\{c_i^{<j}, f_{R\to E}^{<j}\}}_{\text{execution calibration}}) \qquad (5)$$

The calibrated result is then incorporated into the context of $\mathcal{M}_G$ for the next planning generation.

Adaptive interaction. In our adaptive interaction strategy, the agent flow primarily alternates from (1) generating tool-use planning by agent $\mathcal{M}_G$ and (2) generating execution code by agent $\mathcal{M}_E$, in a step-by-step manner. The review agent $\mathcal{M}_R$ is adaptively triggered to provide feedback only when the generated code fails to execute correctly. Specifically, a runtime error can be caused by either unfeasible planning or coding faulty. Thus, the agent $\mathcal{M}_R$ first reviews the generated planning and code, routines the errors to agent $\mathcal{M}_G$ or $\mathcal{M}_E$ accordingly, and provides feedback for revision.

4 Specialization by Agent Distillation

Our initial experiment shows that powerful LLMs such as GPT-4, achieve promising results when equipped with our framework. However, these model are often considered black boxes (Qin et al., 2023a; Gao et al., 2023b) with potential privacy issues. Thus, we aim to adapt our framework to open-source models. We propose *specialized action distillation* (SPAN), which distills the task-solving trajectory of powerful commercial LLMs into different open-source LLM agents tailored to specific functionalities in ConAgents.

4.1 Synthesize the Training Dataset

Our distillation method collects the task-solving trajectory of specialized agents simulated by GPT-4, in ConAgents (§ 3.1). To achieve this, we first sample tasks from ToolBench (Qin et al., 2024), which contains nearly 200k practical tasks across

Statistic	
# The data scale	500
# The average tokens of input task	52.48
# The average number of candidate tools	20
# The average number of ground truth tools per task	3.39
# The average turns of planning-review interaction	4.62
# The average turns of execution-review interaction	5.21

Table 1: The statistics of our synthetic dataset in our *specialized action distillation* method.

3,451 tools. We select 2,919 tasks using various heuristic strategies (see Appendix A.2 for more details). Each task x is paired with a list of relevant tools. Since we find that some tasks in ToolBench are very similar to each other, we cluster them based on the semantic similarities between task descriptions and retain one instance for each cluster. Next, we supplement each of these selected tasks with a detailed solution. Specifically, we separately implement our grounding, execution, and review agent with GPT-4, and coordinate them using the proposed automatic communication protocol (§ 3.3) to generate solutions. Finally, we synthesize a dataset with 500 diverse examples. Each example contains a task x, a candidate toolset $\mathcal{S}$, and the task-solving trajectory of three agents. The statistics of our synthetic dataset are provided in Table 1.

4.2 Agent Training

Due to the large number of parameters of the LLM, we employ a parameter-efficient tuning technique (*i.e.,* LoRa (Hu et al., 2021)) to train each specialized agent separately. The objective is to optimize the delta parameters $\Delta\theta$ of the LLM θ to minimize the loss function.

We reorganize the dataset according to the agents' functionality (§ 3.1), thereby distilling specific abilities into different student models. Formally, given a task x, in the ith step, the $\mathcal{H}_i$ contains historical planning and execution results. We train the agent $\mathcal{M}_G$ to generate the ith tool-use planning t_i on the condition of H_i and revise its incorrect planning following the review from agent $\mathcal{M}_R$ (Eq. 4). We train the agent $\mathcal{M}_E$ to generate programs c for tool execution following the generated planning t and feedback of agent $\mathcal{M}_R$ (Eq. 5). Similarly, the agent $\mathcal{M}_R$ are trained to provide feedback as Eq. 2 and Eq. 3. We apply the standard language modeling loss for the optimization. More details and formulations can be found in Appendix A.1.

5 Experimental Setup

5.1 Datasets and Evaluation Metrics

Datasets. We conduct experiments on two well established benchmarks, *i.e.,* RestBench (Song et al., 2023) and Toolbench (Qin et al., 2024). The RestBench consists of two subsets, including: (1) TMDB, a high-quality human annotated dataset consisting of 54 movie-related tools; and (2) Spotify, a dataset with 40 music-related tools. The Toolbench contains various practical tasks across diverse scenarios. We provide more details for these datasets in Appendix A.3.

Evaluation metrics. Following Yang et al. (2023a); Gao et al. (2023b), we use two evaluation metrics: (1) Success Rate (**Success%**) measuring the proportion of successful query completions, and (2) Correct Path Rate (**Path%**) calculating the F1 score between the generated tool sequence and ground-truth tool sequence. We also conduct a human evaluation, in which three well-educated volunteers are invited to evaluate 30 randomly sampled cases with a three-scale rating in two aspects: (1) Executability (Exec): whether multiple tools are invoked in a correct logical order; and (2) Utility: whether the execution results of tools can be used to generate an answer.

5.2 Baselines

We compare our method with agent-based tool learning methods, including: (1) *Chameleon* (Lu et al., 2023), an LLM-based agent that directly generates multi-step plans for tool use and then sequentially executes the plan; (2) *ReAct* (Yao et al., 2023), which prompts LLM to generate the chain-of-thought and actions in an interleaved manner.; (3) *CodeAct* (Wang et al., 2024a), which allows the LLM to generate executable code snippets as actions to use tools; (4) *ToolLLM* (DFSDT, Qin et al., 2024), which enhances LLMs with the Depth First Search-based Decision Tree (DFSDT) to select tools to solve a task. For further comparison, Since our ConAgents coordinates three specialized agents, we also establish two baselines, *i.e.,* ReAct@N and ToolLLM@N, which are up to N times runs of their vanilla method (ReAct or ToolLLM) until an input task is completed.

We also consider baselines with multi-agent architecture, including (1) *RestGPT* (Song et al., 2023): which consists of a planning module, a

Method	RestBench-TMDB		RestBench-Spotify		ToolBench	
	Success Rate	Path%	Success Rate	Path%	Success Rate	Path%
gpt-3.5-turbo						
ReAct (Yao et al., 2023)	40.00	71.19	51.28	60.35	39.39	65.04
Chameleon (Lu et al., 2023)	63.00	66.10	56.20	64.55	37.44	67.55
CodeAct (Wang et al., 2024a)	63.00	80.91	54.30	76.64	–	–
ToolLLM (DFSDT, Qin et al., 2024)	68.00	76.77	61.40	74.77	66.39	**86.43**
Reflexion (Shinn et al., 2023)	53.00	55.00	49.10	50.90	–	–
α-UMi (Shen et al., 2024)	62.00	70.23	66.74	70.27	67.55	78.37
RestGPT (Song et al., 2023)	65.00	69.21	67.10	70.75	63.88	77.40
ConAgents *w/ Ada*	78.00	79.57	69.43	77.54	69.84	81.58
ConAgents *w/ Auto*	**79.00**	**81.97**	**71.21**	**79.17**	**72.15**	83.33
ReAct@N $\rightarrow$ N = 2	54.00	67.90	56.71	59.47	41.41	63.67
ReAct@N $\rightarrow$ N = 3	62.00	65.40	58.13	63.26	42.67	66.12
ToolLLM@N $\rightarrow$ N = 2	70.00	76.54	63.16	75.27	68.37	86.43
ToolLLM@N $\rightarrow$ N = 3	71.00	78.11	63.16	76.30	68.77	87.54

Table 2: **The results on three datasets.** The metrics Success% and Path% indicate the Success Rate and Correct Path Rate, respectively. The icon denotes the single-agent method and symbolizes multi-agent architecture.

Method	TMDB		Spotify	
	Success%	Path%	Success%	Path%
ConAgents (*Mistral-8x7B*)				
w/ Auto (Distilled)	53.00	79.32	36.09	73.92
w/ Auto (Vanilla)	49.00	76.22	34.21	68.14
w/ Ada (Distilled)	51.00	78.74	35.47	69.86
w/ Ada (Vanilla)	47.00	74.05	33.33	66.41
Baselines (*Mistral-8x7B*)				
ReAct	26.00	61.21	21.35	47.21
ReAct@3	33.00	63.27	26.93	50.31
ToolLLM	37.00	64.32	28.07	52.31
ToolLLM@3	45.00	74.40	31.58	57.68
RestGPT	34.00	72.20	31.58	67.82

Table 3: We employ the Mixtral-8x7B as the backbone LLM of for our method and baselines. The *Vanilla* and *Distilled* indicate enable our framework by prompting and our action distillation, respectively.

tool selector, an executor, and a response parsing module; (2) *Reflexion* (Shinn et al., 2023), which employs an LLM for task execution and uses another LLM to verbally reflect on task feedback signals; and (3) *α-UMi* (Shen et al., 2024), which consists of a planner, an executor, and an answer generator.

5.3 Implementation Details

We use *gpt-3.5-turbo*[2] from OpenAI as the LLM backbone for each agent in our method and all baselines. We instruct the three agents to perform specific actions with different system prompts shown in Appendix A.6. The decoding temperature is set to 0 for the most deterministic generation. We also repeat the experiment with an open-source model Mistral-8x7B[3] for further comparison. In our agent communication (§ 3.3), we set the maximum iteration of interactions $\alpha = 3$ and $\beta = 3$, respectively. For each sample in the test set, we provide all the baselines with the same candidate toolset for a fair comparison, which contains the required tools and ten randomly sampled tools.

Our action distillation separately trains three Mistral-8x7B using the corresponding optimization objectives in § 4.2 with the learning rate of 5×10^{-5}. The training of our model can be done within 4 hours with 3 NVIDIA A800-PCIE-80GB GPUs using LoRA (Hu et al., 2021).

6 Results and Analysis

6.1 Experimental Results

Overall performance. Table 2 demonstrates the experimental performances of all methods. We find that our proposed ConAgents outperforms all the baselines in three datasets in terms of all metrics. A reason here is that our cooperative framework design enables each agent to perform specialized actions instead of grasping all required capabilities, thereby reducing the workload encountered by a single agent. The significant improvement over ReAct@N and ToolLLM@N baselines can further validate the effectiveness of our framework. Compared with baselines with multi-agent architecture like RestGPT, ConAgents

[2] https://openai.com/chatgpt

[3] https://huggingface.co/mistralai

Method	TMDB		Spotify	
	Success%	Path%	Success%	Path%
Ours w/ Auto	79.00	81.97	71.43	77.54
w/o $\mathcal{M}_R \rightarrow \mathcal{M}_G$	77.00$\downarrow_{2.0}$	78.10$\downarrow_{3.9}$	68.42$\downarrow_{3.0}$	75.33$\downarrow_{2.2}$
w/o $\mathcal{M}_R \rightarrow \mathcal{M}_E$	75.00$\downarrow_{4.0}$	74.23$\downarrow_{7.7}$	64.91$\downarrow_{6.5}$	72.41$\downarrow_{5.1}$
w/ static coop.	75.00$\downarrow_{4.00}$	75.74$\downarrow_{6.2}$	67.12$\downarrow_{4.3}$	75.07 $\downarrow_{2.5}$

Table 4: The ablation study on two datasets with *gpt-3.5-turbo* as backbone. See § 6.3 for details

Method	TMDB		Spotify	
	Exec	Utility	Exec	Utility
gpt-3.5-turbo				
ReAct	1.89	1.93	1.77	2.10
ToolLLM	2.26	1.87	2.26	2.30
RestGPT	2.35	2.45	2.30	2.40
Ours *w/ Auto*	2.47	2.56	2.43	2.50
Ours *w/ Ada*	2.43	2.50	2.38	2.45

Table 5: Human evaluation on Executability (**Exec**) and Correct Rate of Parsing (**Parsing**).

achieves about 12% higher Success Rate. The potential reason for our improvement is that the proposed two communication protocols enable the dynamic interaction of agents, which is more flexible to handle exception errors.

Performance with the open-source LLM. We further evaluate our ConAgents by swapping the backbone LLM with Mistral-8x7B and repeating the experiment under the same conditions. As shown in Table 3, we implement our framework in two ways with Mistral-8x7B: (1) directly prompting (*w/ Auto* and *w/ Ada*); (2) tuning with our proposed action distillation (*w/ Auto*† and *w/ Ada*†). We observe that directly prompting Mistral-8x7B with ConAgents yields better performance than baselines. The action distillation further improves overall performance substantially, such as pushing the Success Rate from 47.00 to 51.00 in the TMDB dataset. These results further prove the effectiveness of our cooperative framework.

6.2 Human Evaluation

Table 5 shows the results of the human evaluation. We find that ConAgents achieves the best results in the Executability aspect with 0.08~0.12 improvement. These results further validate the necessity of agent specialization and cooperation. The overall Kappa statistics for Executability and Utility are 0.75 and 0.71, illustrating substantial agreement (Landis and Koch, 1977) among the annotators.

6.3 Ablation Study

To better understand the impact of different components of our method, we make the following modifications to the architecture and measure the effect.

- **w/o** $\mathcal{M}_R \rightarrow \mathcal{M}_G$. We remove the interaction between agent $\mathcal{M}_R$ and $\mathcal{M}_G$ in our framework. As shown in Table 4, the Success Rate has a average 2.50 decline, while the Correct Path Rate has a 3.05 average decline on two datasets. This results validate the necessity of feedback of $\mathcal{M}_R$ which can instruct the $\mathcal{M}_G$ to revise incorrect planning.
- **w/o** $\mathcal{M}_R \rightarrow \mathcal{M}_E$. We remove the interaction between agent $\mathcal{M}_R$ and $\mathcal{M}_E$ in our framework when programming to execute tools. As shown in Table 4, the Success Rate suffers from obvious decrease in both two datasets. These results indicate that the agent $\mathcal{M}_R$ can review the generated programs of agent $\mathcal{M}_E$ and provide useful instruction for calibrating errors.
- **w/ static cooperation**. We implement the $\mathcal{M}_R$ with a code compiler, which is triggered to provide static feedback only when runtime errors are raised during executing tools by agent $\mathcal{M}_E$. This allows us to compare our framework with a static algorithm for agent cooperation. Table 4 present the results, where we observe a 4.12 average decrease in the Success Rate, *e.g.,* dropping from 79.00 to 75.00 on the TMDB dataset. The same trend is also observed in the Correct Path Rate, *e.g.,* a 2.5 decrease on the Spotify dataset. These results indicate the superiority of our dynamic agent cooperation framework.

6.4 Case Study

We conduct the case studies and find that our cooperative agent framework is more effective at executing various tools and handle exceptional errors in solving tasks. We also provide examples to explain the detailed process of agent cooperation. The details can be found in Appendix A.5.

7 Discussion

Qualitative analysis for the maximum number of interactions. In our *automatic agent interaction*, agents $\mathcal{M}_G$ and $\mathcal{M}_E$ revise their actions following the feedback of agent $\mathcal{M}_R$ for up to α and β turns, respectively. To further explore the impact of the interaction times on

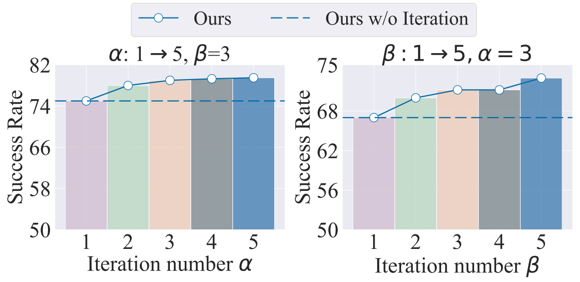

Figure 3: The qualitative analysis for the maximum interaction turns α and β in our agent communication protocols (Section 3.3) on the TMDB dataset.

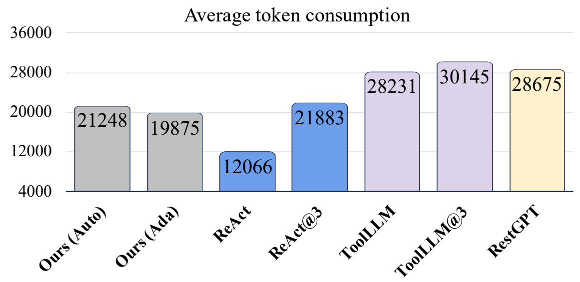

Figure 4: The efficiency analysis for different methods, where we count the average consumed tokens.

overall performance, we conduct a quantitative and qualitative analysis by varying α and β from 1 to 5. Then we evaluate our framework using the RestBench-TMDB dataset with the same settings as in Table 2. As illustrated in Figure 3, we find an increasing Success Rate when the maximum iteration turns shifts from 1 to 3. In addition, a relatively stable trend is observed when the α and β keep increasing (from 3 to 5), which indicates the agents can correct most errors within 3 turns. We also look at the poorly performing cases where we find that since the planning from agent $\mathcal{M}_G$ is typically open-ended, the $\mathcal{M}_R$ struggles to detect all the incorrect planning. For example, the planning may be plausible and clear but lacks the required arguments to execute tools, thus resulting in a failure of $\mathcal{M}_E$ in subsequent steps.

Qualitative analysis for the efficiency of inference. Due to the intensive inference cost of LLMs-based agents, we further explore the efficiency of our ConAgents. To explain more intuitively, we compare the token consumption for the ConAgents and baselines using the RestBench-TMDB dataset with the same settings as in Table 2. As illustrated in Figure 4, we find that although our framework achieves better performance, we spend fewer tokens compared with strong baselines such as RestGPT and ToolLLM@3. The reason is that the cooperative framework ConAgents enables each agent to perform specific tasks more efficiently, reducing the length exploration trajectory by the single agent.

The quality of generated review. We further analyze the quality of reviews given by review agent $\mathcal{M}_R$. Specifically, we randomly sample 50 task-solving trajectories in Table 2 (w/ Auto) manually analyze the review of review agent. For most tasks, we find that the agent $\mathcal{M}_R$ can assist agent $\mathcal{M}_E$ to revise its generated code or provides useful reviews for the planning generated by agent $\mathcal{M}_G$, such as `only select tools from given list`. In addition, we find that in less than 5% of tasks, the agent $\mathcal{M}_R$ hallucinates giving an incorrect review, indicating its reliability.

Runtime consistency. Considering the non-deterministic nature of LLM generation, we analyze the consistency of our framework. We repeat our method multiple times with the same settings as in Table 2. The statistical significance of differences observed between the performance of two runs is tested using a two-tailed paired t-test. We find no significant difference between the results of two randomly conducted experiments (significance level $\alpha = 0.05$).

8 Conclusions

We present a cooperative and interactive agents framework (ConAgents) for tool learning, which diverges from previous work by allowing the cooperation of agents to solve complex tasks. The ConAgents first modularizes the overall workflow with three specialized agents for tool planning, tool execution, and action calibration, respectively. Then, two communication protocols are introduced to enable the dynamic cooperation of these agents. To generalize our framework to open-source models, we propose specialized action distillation, enhancing the models' capability to perform specific actions. Extensive experiments conducted on three datasets demonstrate the superiority of our ConAgents, *e.g.,* pushing the success rate to 77.00 with 13.2% point improvement. Our future work includes: (1) extending our method to agents empowered by multi-modal foundation models, incorporating image and sound; (2) coordinating the cooperation between strong and weak agents.

Limitations

The main limitation is that our LLM-based agent is limited when perceiving multi-modal tasks. When executing the tools, we represent the image and speech input with `url`, following previous works. In the future, we plan to extend our method to agents empowered by multi-modal foundation models.

Ethics Statement

The paper proposes a cooperative agent framework, synergizing specialized agents to solve complex tasks. The modularized design enables the agents to utilize feedback from the tool environment to calibrate themselves adaptively. In addition to the use of state-of-the-art commercial LLMs, we have experimented with an open-source LLM, for reproducibility reasons and to allow the use of our method in lower-resource contexts. All the tools used in our experiment are provided by open-source platforms, including TMDB, Spotify, and Rapid API, thus ensuring a high level of transparency and reproducibility.

We have made every effort to ensure that our research does not harm individuals or groups, nor does it involve any form of deception or potential misuse of information.

References

Andres M Bran, Sam Cox, Andrew D White, and Philippe Schwaller. 2023. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*.

Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2023. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*.

Roi Cohen, May Hamri, Mor Geva, and Amir Globerson. 2023. Lm vs lm: Detecting factual errors via cross examination. *arXiv preprint arXiv:2305.13281*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jian, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D Hwang, et al. 2023. Faith and fate: Limits of transformers on compositionality. *arXiv preprint arXiv:2305.18654*.

Yao Fu, Hao Peng, Tushar Khot, and Mirella Lapata. 2023. Improving language model negotiation with self-play and in-context learning from ai feedback. *arXiv preprint arXiv:2305.10142*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023a. PAL: Program-aided language models. In *PMLR*, pages 10764–10799.

Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. 2023b. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *ArXiv*, abs/2303.17491.

J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. 2023. Dynamic llm-agent network: An llm-agent collaboration framework with agent team optimization. *arXiv preprint arXiv:2310.02170*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *ArXiv*, abs/2304.09842.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9802–9822.

Amirkeivan Mohtashami, Florian Hartmann, Sian Gooding, Lukas Zilka, Matt Sharifi, et al. 2023. Social learning: Towards collaborative learning with large language models. *arXiv preprint arXiv:2312.11441*.

Reiichiro Nakano, Jacob Hilton, S. Arun Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen

Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv*, abs/2112.09332.

Bhargavi Paranjape, Scott M. Lundberg, Sameer Singh, Hanna Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. Art: Automatic multi-step reasoning and tool-use for large language models. *ArXiv*, abs/2303.09014.

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*.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2023. Adapt: As-needed decomposition and planning with language models. *arXiv preprint arXiv:2311.05772*.

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

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. *arXiv preprint arXiv:2401.05268*.

Yujia Qin, Zihan Cai, Dian Jin, Lan Yan, Shihao Liang, Kunlun Zhu, Yankai Lin, Xu Han, Ning Ding, Huadong Wang, Ruobing Xie, Fanchao Qi, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023a. WebCPM: Interactive web search for Chinese long-form question answering. In *ACL*, pages 8968–8988.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shi Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bo Li, Ziwei Tang, Jing Yi, Yu Zhu, Zhenning Dai, Lan Yan, Xin Cong, Ya-Ting Lu, Weilin Zhao, Yuxiang Huang, Jun-Han Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023b. Tool learning with foundation models. *ArXiv*, abs/2304.08354.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *ICLR*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*.

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*, abs/2302.04761.

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. *arXiv preprint arXiv:2401.07324*.

Yongliang Shen, Kaitao Song, Xu Tan, Dong Sheng Li, Weiming Lu, and Yue Ting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *ArXiv*, abs/2303.17580.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning.(2023). *arXiv preprint cs.AI/2303.11366*.

Yifan Song, Weimin Xiong, Dawei Zhu, Chengzu Li, Ke Wang, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world applications via restful apis. *ArXiv*, abs/2306.06624.

Qiushi Sun, Zhangyue Yin, Xiang Li, Zhiyong Wu, Xipeng Qiu, and Lingpeng Kong. 2023. Corex: Pushing the boundaries of complex reasoning through multi-model collaboration. *arXiv preprint arXiv:2310.00280*.

Yashar Talebirad and Amirhossein Nadiri. 2023. Multi-agent collaboration: Harnessing the power of intelligent llm agents. *arXiv preprint arXiv:2306.03314*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023a. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024a. Executable code actions elicit better llm agents. In *ICML*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023b. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023c. Self-instruct: Aligning language models with self-generated instructions. In *ACL*.

Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. 2024b. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452*.

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

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

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 *ICLR*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2023. Lumos: Learning agents with unified data, modular design, and open-source llms. *arXiv preprint arXiv:2311.05657*.

Junjie Zhang, Yupeng Hou, Ruobing Xie, Wenqi Sun, Julian McAuley, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. 2023. Agentcf: Collaborative learning with autonomous language agents for recommender systems. *arXiv preprint arXiv:2310.09233*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *arXiv preprint arXiv:2306.13304*.

A Appendix

A.1 Details of Action Distillation

Our specialized action distillation (SPAN) trains three student models separately using the task-solving trajectory of a powerful model, *i.e.,* GPT-4 in our implementation. These three student models are trained to conduct specific actions of the grounding agent, execution agent, and review agent, respectively. Their initial parameters weights θ are initialized from the same open-source model $\mathcal{M}_\theta$. Since we use LoRa (Hu et al., 2021) for parameter-efficient tuning, the optimization objective of our distillation is to search for the delta parameter $\Delta\theta$ to minimize the loss function. Here, we introduce their detailed optimization objectives.

Notations. As mentioned in § 3, we denote an input task as x, which is solved in a step-by-step manner while the task-solving context is denoted as $\mathcal{H}$. In ith step, the context $\mathcal{H}_i$ contains historical planning $t_{<i}$ and execution results $r_{<i}$. The planning t specifies a tool to use in a current step which is selected from a candidate toolset $\mathcal{S}$.

Training of grounding agent. Given a task x, we train the grounding agent $\mathcal{M}_G$ to decompose x into simpler sub-tasks and ground each sub-task into tool-use planning t on the condition of the current context H and revise incorrect planning following the feedback $f_{R\to G}$ of the review agent $\mathcal{M}_R$. For each step t_i, we use the standard language modeling loss for optimization, which can be formulated as:

$$\mathcal{L}_G = -\log P_{\theta+\Delta\theta_G}\left(t_i^j | x, \mathcal{H}_i, \mathcal{S}, \{t_i^{<j}, f_{R\to G}^{<j}\}\right)$$

Here, the j indicate the jth interaction between the agent $\mathcal{M}_G$ and $\mathcal{M}_R$. The $\{t_i^{<j}, f_{R\to G}^{<j}\}$ indicates the planning-review alternated from agent $\mathcal{M}_G$ to $\mathcal{M}_R$. The LoRa parameter of agent $\mathcal{M}_G$ is denoted as $\Delta\theta_G$.

Training of execution agent. Similarly, in the ith step, we train the execution agent $\mathcal{M}_E$ to execute a tool following the planning t_i by generating an executable program, and then calibrate incorrect code following the review of agent $\mathcal{M}_R$. Formally, the optimization objective can be formulated as:

$$\mathcal{L}_E = -\log P_{\theta+\Delta\theta_E}\left(c_i^j | x, t, d, \{c_i^{<j}, f_{R\to E}^{<j}\}\right)$$

Here, d indicates the tool documentation. The LoRa parameter of agent $\mathcal{M}_E$ is denoted as $\Delta\theta_E$.

Training of review agent. The review agent agent is trained to provide reviews for agent $\mathcal{M}_E$ and $\mathcal{M}_R$, calibrating their incorrect actions, *i.e.,* planning or execution. Thus, the optimization objective can be formulated as:

$$\mathcal{L}_R = -\sum_{j=1}^{\alpha}\log P_{\theta+\Delta\theta_R}\left(f_{R\to G}^j | x, S, t_i^{j-1}\right) - \sum_{j=1}^{\beta}\log P_{\theta+\Delta\theta_R}\left(f_{R\to E}^j | x, d, c_i^{j-1}, r_i^{j-1}\right)$$

Here, the LoRa parameter of agent $\mathcal{M}_R$ is denoted as $\Delta\theta_R$.

A.2 Heuristic Strategies for Data Selection

We employ the following heuristic methods to filter low-quality tasks in the original ToolBench:

- Each task in ToolBench is paired with a list of candidate tools. Generally, the more candidate tools there are, the more complex the task. Thus, we filter out tasks with fewer than 10 candidate tools to ensure the overall complexity of the sampled tasks.

- To improve the quality of our training dataset, we remove tasks if their tools are not callable or deprecated.

- We remove tasks if their tools lack the required documentation or if the documentation is less than 100 words in length.

A.3 Datasets

Experiment dataset We conduct experiments on three commonly-used datasets with tool learning tasks, including:

- RestBench (Song et al., 2023): a high-quality human annotated dataset consisting of 54 tools about movie scenarios.

- RestBench-Spotify (Song et al., 2023): a dataset with 40 tools for music scenarios.

- ToolBench (Qin et al., 2024): a dataset containing diverse real-world tools across various applications, which contains the simple tasks, *i.e.,* solving a task with one single tool, and complex tasks, *i.e.,* executing multiple tools in a logic order to solve a task.

We conduct experiments on the full dataset of TMDB and Spotify. Due to the intensive inference cost of LLMs-based agents, we randomly sample 117 cases as test sets from the complex tasks in Toolbench datasets to evaluate the performance of our cooperative agent framework in solving practical tasks. We will release the sampled task for the transparency consideration.

Extend existing datasets. The original ToolBench benchmark only provides a step-by-step task-solving trajectory of GPT-3.5, which consists of both valid ground truth tools and irrelevant tools. However, our evaluation involves computing the overlap between model-selected tools with ground truth tools. Therefore, we repurpose the ToolBench to support our evaluation methods. Specifically, for each task, we extract the tools in the original solution provided by ToolBench and only retain the relevant tools that are required for solving the task. We invite three well-educated masters with relevant research backgrounds to implement this process. To guarantee annotation quality, we ask at least two annotators to annotate the same task repeatedly. If there is a discrepancy between the two annotators (i.e., two annotators give different answers), we ask a third annotator to recheck it. We hold regular meetings and pre-annotation tests to ensure that each expert undergoes detailed training to familiarize themselves with our annotation task. We will release these repurposed tasks to facilitate future research.

A.4 Evaluation Metrics Details

Automatic evaluation. We mainly employ Success Rate and Correct Path Rate as two automatic evaluation metrics, following previous works (Yang et al., 2023a; Gao et al., 2023b). The Success Rate (**Success%**) computes the proportion of successful query completions. Specifically, when all the ground-truth tools are executed correctly, the Success Rate is set to 1; otherwise, it is set to 0. The Correct Path Rate (**Path%**) computes the F1 score between the generated tool sequence and the ground-truth tool sequence.

Human evaluation We conduct a human evaluation on two metrics, including: (1) Executability (**Exec**): whether the multiple tools are invoked in a correct logical order to complete the task; and (2) Utility: whether the execution results of tools can be used to generate an answer. We invite three well-educated volunteers to evaluate 30 cases randomly sampled from RestBench-TMDB and RestBench-Spotify datasets, respectively, with a three-scale rating. Using a 3-point scale over a binary scale provides an option for the annotators to factor in their subjective interpretation of the extent of success or failure of a system's response to satisfy a user's request. The instructions used in our human evaluation are summarized as follows.

The evaluation guideline for our human evaluation.

```
In this evaluation task, you are
provided with some question-solution
```

```
pairs. The question can be only solved
by using real-world tools (or APIs). The
 solution is a sequential tool-use
process, involving multi-step tool
callings.

Your task is to rate the quality of the
solution on a three scale based on the
following two  metrics:
1. Executability: Whether multiple tools
 are invoked in a correct logical order
to complete the task.
2. Utility: Whether the model can
observe the relevant values from lengthy
 execution results, incorporate them to
predict the next action, and finally
output a correct answer.

We also provide scoring criteria for
your reference. Please adhere to our
criteria since we will re-check the
score you provide.
Now, read the following criteria and
rate the provided question-solution
pairs. Note that, you are encouraged to
give us feedback and share any confusion
 you may have.

==Scoring Criteria==

1. For the Executability metric:
- Three points: Call all necessary tools
 correctly and solve the task. Allow for
 redundant tools or inference steps.
- Two points: Not fully calling all
necessary tools correctly, partially
solving the task.
- One point: Only some sub-steps are
solved and the entire task is not
completed. And there is a lot of
redundancy or incorrect reasoning.

2. For the Utility metric:
- Three points: A majority of the
execution results of the tools are
correctly used to address the question (
minor mistakes are allowed).
- Two points: Only part of the execution
 results of the tools are used. For
example, in a question requiring finding
 an actor's highest-grossing film, the
correct solution is to sequentially look
 at all the films the actor has appeared
 in, instead of just counting the top-k
like top-5 or top-10.
- One point: Only a small part of the
execution results of the tools are used,
 while other useful intermediates are
ignored.
```

A.5 Case Study

We conduct several case studies and find that our method is effective at executing various tools and incorporating execution results to solve the input tasks. Figure 5 presents a concrete example of the workflow of our proposed cooperative framework.

Case for our automatic agent communication. Figure 5 shows an example of our proposed *automatic communication protocol*. For each turn, the communication starts with the planning-and-review between the grounding agent and review agent. Following the planning , the execution agent generates programs to execute tools and calibrates the incorrect result with the review of review agent. For example, in the first turn, the agent $\mathcal{M}_G$ re-generate a planning following the review from agent $\mathcal{M}_R$, and finally output a clear planning. This example also illustrate the interaction between grounding agent $\mathcal{M}_G$ and review agent $\mathcal{M}_R$, where the agent $\mathcal{M}_G$ calibrates its execution programs following the feedback of $\mathcal{M}_R$, and finally generate

Case for our adaptive agent communication Figure 6 shows an example of our proposed *adaptive communication protocol*. The agent flow mainly alternates between (1) generating tool-use planning by grounding agent and (2) generating execution code by execution agent, in a step-by-step manner. The review agent is adaptively triggered to provide feedback only when the generated code fails to execute correctly. For example, in the second turn, agent $\mathcal{M}_E$ initially generates a wrong program due to the lack of necessary arguments. Then, agent $\mathcal{M}_R$ reviews the current context, routes this error to agent $\mathcal{M}_G$, and instructs $\mathcal{M}_G$ to supplement this argument, instead of directly shifting to the next state with an error response. This example intuitively illustrates the process of our adaptive interaction.

A.6 System prompt for Agents

We provide the system prompts of three specialized agents in our ConAgents as follows.

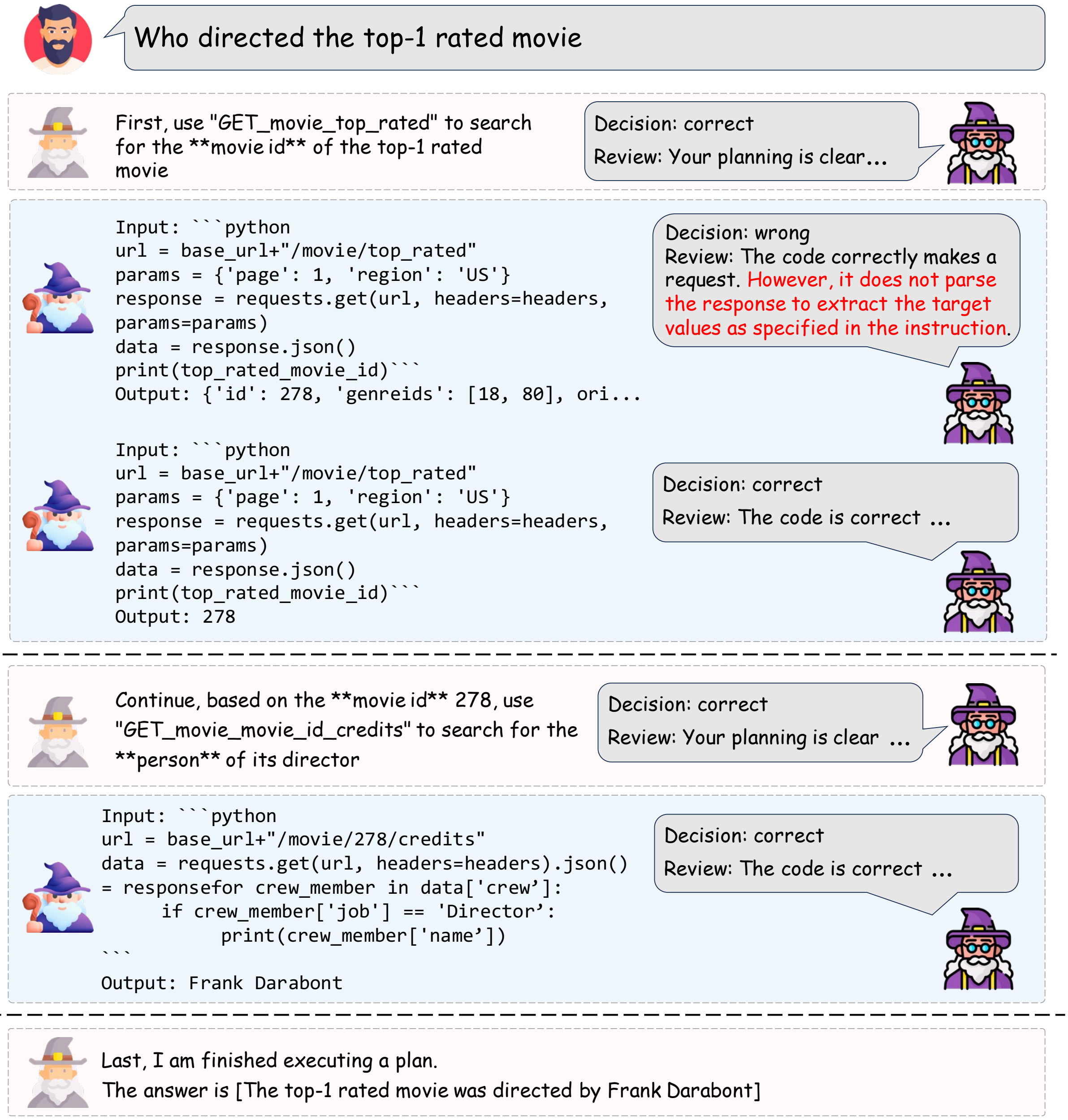

Figure 5: An example to illustrate the proposed *automatic agent communication* of our framework ConAgents. For each turn, the communication starts with the planning-and-review between the grounding agent and review agent. Following the planning , the execution agent generates programs to execute tools and calibrates the incorrect result with the review of review agent . In this figure, we highlight the useful review of review agent with red.

Figure 6: An example to illustrate the proposed *adaptive agent communication* in our framework ConAgents. The agent flow mainly alternates from (1) generating tool-use planning by grounding agent and (2) generating execution code by execution agent, in a step-by-step manner. The review agent is adaptively triggered to provide feedback only when the generated code fails to execute correctly. In this figure, we highlight the review of review agent with red.

The prompt to enable the grounding agent in our cooperative framework. We detail the system prompt and input prompt, respectively.

```
==== System prompt ====
In this task, you are provided a series of real-world APIs. Please give a solution
to plan which API to use step-by-step to solve the user's query. Specifically, you
should solve a query with interleaving Thought, Action, Observation steps.
You should break down the user's query into several simple sub-task. And give your
Thought and Action step-by-step, waiting for the Observation to continue to give the
 subsequent solution.

Starting blow, you have to output in the following format
User query: the query a User wants help with related to the API.
Thought 1: the first step of your plan for how to solve the query
Action 1: Select just one API from the provided API list and output its API name
Observation 1: the result of executing the first step of your plan
Thought 2: based on the API response, the second step of your plan for how to solve
the query.
Action 2: Select just one API from the provided API list and output its API name
Observation 2: the result of executing the second step of your plan
... (this Thought and Action step can be repeated N times)
Thought N: The task is solved and the answer is [PROVIDE YOUR ANSWER]!
Action N: Finish

{examples}

In your Thought, You are encouraged to specify what key information you want to
obtain from the full results or the selected API. For example, you are encouraged to
 give a plan "Find the **movie id** of the Matrix" instead of "Find the movie Matrix
".

==== Input task ====

Here are the details of the APIs you can use. You can ONLY use the above APIs!
{Candidate toolset}

User query: {query}
```

The prompt to enable the execution agent in our cooperative framework. We detail the system prompt and input prompt, respectively.

```
==== System prompt ====

In this task, you should write Python code to call the following API according to my
 instructions. Note that: you should always `print` the final answer.


==== Input prompt ====
Here is the documentation of the API
{tool documentation}

Any information, e.g., person id or movie id, you need to obtain it by calling
appropriate APIs. DO NOT make up value by yourself!
Instruction: {instruction}
Your output: ```python
Your Python code
```
```

*The prompt to enable the review agent in our cooperative framework. We detail the system prompt and input prompt, respectively.*

```
==== System prompt ====
Please review my solution and evaluate whether my solution is correct.

==== Input prompt ====
My solution contains the Planning, Code, and Execution results.
Planning: the specific plan for how to solve the query
Code: the Python code snippet which calls the APIs to solve the question
Code output: the result of executing the Python code

Here is the tool documentation.
{tool documentation}

Now, please review my solution. Your review should contain the type of error (
planning error and Code error) and a clear instruction on how to fix it.
(1) Planning error: The planning is vague, and lacks required arguments to call a
function or does not explicitly state what information to extract from the function'
s return value.
(2) Code error: The planning is correct and specific, but the code encounters Python
 runtime errors, e.g., KeyError or TypeError.

Starting below, please first identify the error type of my solution and then
provide a clear instruction on how to fix it. Here are some review examples:
{examples}

Here is my specific solution.
{solution}

Please give your review.
Error:
```
```