Divide-Then-Aggregate: An Efficient Tool Learning Method via Parallel Tool Invocation

Dongsheng Zhu[1†], **Weixian Shi**[1†], **Zhengliang Shi**[2 ‡],
Zhaochun Ren[3], **Shuaiqiang Wang**[1], **Lingyong Yan**[1*], **Dawei Yin**[1*]
[1]Baidu Inc., Beijing, China [2]Shandong University, Qingdao, China
[3]Leiden University, Leiden, The Netherlands
`{zhudongsheng, shiweixian}@baidu.com` `zhengliang.shii@gmail.com`
`lingyongy@gmail.com` `yindawei@acm.org`

Abstract

While Large Language Models (LLMs) demonstrate remarkable capabilities, their ability to autonomously execute complex real-world tasks remains limited. Accordingly, tool learning has emerged to enable LLMs to effectively leverage external tools to extend their capabilities. Current tool-learning paradigms like CoT/ReAct employ sequential tool invocation but suffer from constrained perception and inadequate task planning. Alternative approaches using search-based decision trees incur substantial computational overhead. To address these limitations, we propose DTA-Llama (**D**ivide-**T**hen-**A**ggregate Llama), a novel parallel tool invocation framework featuring: (1) A Directed Acyclic Graph (DAG) structure that transformed from traditional tree-based tool search paths, enabling parallel execution and contributing high-quality training data; (2) A process-thread-inspired inference mechanism that iteratively decomposes tasks into parallel tool-using subtasks while aggregating results for subsequent decisions. Experimental results show that our approach substantially enhances task performance while reducing token consumption and inference time. Llama2-7B, using our method, is comparable to the official parallel function calling method of GPT-3.5. The relevant code, dataset, and model weights are available at https://corn0205.github.io/.

1 Introduction

Large Language Models (LLMs), which are pretrained and fine-tuned on massive amounts of textual data, have demonstrated powerful proficiency in various artificial intelligence tasks, such as conversation (Zheng et al., 2023; Zhu et al., 2024b), logical reasoning (Pan et al., 2023) and coding (Nijkamp et al., 2023). However, more real-world tasks often require the LLMs to interact with the environment to get necessary external information or feedback, such as checking real-time flight status (Guan et al., 2024) and complicate calculation for data analysis (Sun et al., 2024). To this end, tool learning has emerged recently which aims to equip the LLMs with external tools and teach them how to leverage the tools to accomplish real-world tasks.

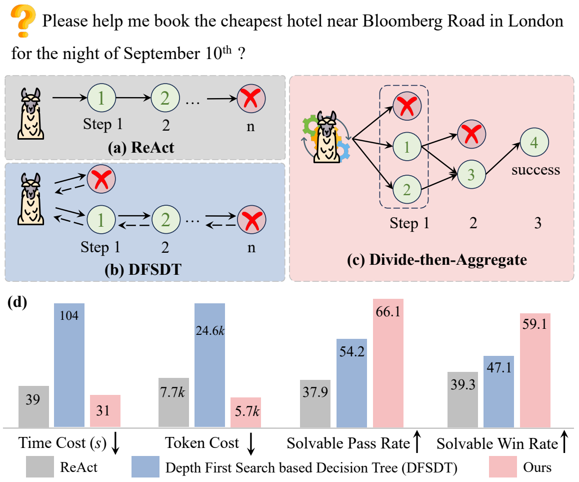


Figure 1: The top block depicts the comparison between CoT/ReAct, DFSDT and our proposed method. The bottom block provides the performance in four aspects of our method and baselines on benchmarks.

Previous tool learning methods typically work in pipelined and tree-based paradigms. Concretely, early studies usually perform the tool invocation in a pipeline, such as the Chain-of-Thought (CoT) reasoning (Kojima et al., 2022), and the ReAct mechanism (Yao et al., 2022). In these methods, tool learning agents (i.e., the LLMs) interact with the environment through a *Thought-Action-Observation* framework (as shown in Figure 1(a)). However, these methods usually focus on reflecting and planning based on local observations, rather than globally perceiving and planning the whole

---

[†] Equal contribution.
[‡] Work done during internship.
[*] Co-corresponding authors.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 28859–28875
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

task solving paths. In contrast, the tree-based methods, such as ToolLLM (Qin et al., 2023) and Toolchain* (Zhuang et al., 2023), adopt tree-based algorithms, e.g., Depth First Search based Decision Tree (DFSDT), to perform the global planning of tool invocation. However, these methods still confront the inevitable backtracking mechanism that iteratively retries, which can usually significantly increase token consumption and inference time (as shown in Figure 1(d) for an example). Moreover, both two kinds of methods suffer from the limitation that LLMs invoke only one tool at each round during tool planning, which narrows their perceptual scope and necessitates more rounds of tool invocation, thereby reducing overall efficiency.

In this paper, we introduce a novel model-based tool, **DTA-Llama** (**D**ivide-**T**hen-**A**ggregate Llama), which enables parallel tool invocation within each round of execution. Specifically, we convert traditional tree-based tool search paths into a Directed Acyclic Graph (DAG) structure with level-order traversal (as illustrated in Figure 1(c)), allowing for parallel execution of tools compared to previous sequential methods. Using the widely adopted ToolBench dataset (Qin et al., 2023), we construct a high-quality parallel tool invocation dataset, **DTA-Tool**, and train Llama (Touvron et al., 2023a,b; Dubey et al., 2024) on it to develop DTA-Llama. Additionally, we design a parallel tool invocation framework inspired by the Process/Threads mechanism (Silberschatz et al., 2006) for inference. In this framework, the **Process** component plans the tool invocation and divides parallelizable tools into separate **Threads**, which then execute independently according to the plan. After execution, an intermediate state lock aggregates the results from all threads. This design shortens the invocation path and significantly improves the efficiency of tool use in LLMs.

We evaluate our approach on StableToolBench (Guo et al., 2024), a comprehensive and reliable real-world tool-use benchmark. Performance is measured using solvable pass rate (SoPR), solvable win rate (SoWR), and actual computational cost. Compared to existing methods, our approach achieves superior tool invocation performance while reducing computational cost—even matching GPT-3.5's (OpenAI, 2022) function-calling performance using only a fine-tuned Llama2-7B (Touvron et al., 2023b). To further assess generalization, we fine-tune multiple LLMs, demonstrating the robustness and generalization ability of our method across different models.

In summary, our main contributions are as follows:

- We transform the tree-based serial data into a DAG format, contributing a high-quality and high-quantity parallel tool invocation dataset to the open-source community.

- A new tool invocation framework has been established, transforming invocation into the Process/Threads format. Combined with the parallel paradigm, this greatly simplifies the invocation path and improves efficiency.

- We comprehensively validate the superiority of DTA-Llama in real-world tool benchmarks, evaluating its performance from three aspects: effectiveness, computational cost, and generalization ability.

2 Related Work

Tool Learning The agent tool learning aims to expand LLMs capabilities by teaching LLMs to use external tools. Many early studies (Patil et al., 2023; Tang et al., 2023; Huang et al., 2023) focus on laying the groundwork for datasets yet exhibit limited variety in tool usage. To bridge this gap, Qin et al. (2023) developed a more comprehensive multi-tool benchmark and proposed an advanced tool invocation method using Depth First Search-based Decision Tree (DFSDT). Building on this, Zhuang et al. (2023) employed A* search algorithm for pruning, while Kim et al. (2023) adopted a compiler-based approach to parallelize tool invocation, both of which improved efficiency to some extent. Meanwhile, Du et al. (2024) and Chen et al. (2024) controlled the stability of LLM tool invocation through self-reflection and Direct Preference Optimization (DPO, Rafailov et al., 2024), respectively. Despite these advances, these methods remain rooted in tree-based search paradigms, lacking a broader perspective on task planning. Additionally, recent works have begun exploring tool creation and integration with agents, opening new avenues for research (Qian et al., 2023; Yuan et al., 2023; Zhu et al., 2024a; Hao et al., 2024; Schick et al., 2024; Hao et al., 2024; Ma et al., 2024; Shen et al., 2024a; Yuan et al., 2024).

Task-Planning for LLMs Task planning capability is a crucial factor for the success of LLMs in problem-solving. Some methods attempt to decompose tasks into sub-goals and then plan for each sub-goal sequentially (Huang et al., 2022; Hu et al., 2023; Lu et al., 2024; Qian et al., 2024; Wang et al., 2024; Shi et al., 2024). HuggingGPT (Shen et al., 2024b) utilizes the LLM as a controller, responsible for decomposing human-input queries into sub-tasks and ultimately generating a comprehensive response. Plan-and-Solve (Wang et al., 2023) employs a two-stage instruction prompting approach: “Let’s first devise a plan" and “Let’s carry out the plan". ProPrompt (Singh et al., 2023) converts natural language descriptions of problems into coding tasks. In other studies, researchers seek to interleave task decomposition and sub-task planning, advancing them dynamically (Gao et al., 2023; Wu et al., 2023; Yao et al., 2024; Shi et al., 2025). The Chain of Thought (CoT) (Kojima et al., 2022; Wei et al., 2022) guides LLMs in reasoning about complex problems by constructing trajectories. ReAct (Yao et al., 2022) alternates between reasoning (the thought process) and planning (the action steps). Reflection (Shinn et al., 2024) builds upon ReAct by introducing a mechanism for LLMs to reflect on previous failures.

3 Methodology

In this section, we describe: (1) the shortcoming analysis of previous methods (§ 3.1); (2) how the DTA-Llama be trained based on constructed parallel tool using data (§ 3.2); (3) how to implement an efficient, Process/Threads-based parallel framework during inference (§ 3.3).

3.1 Background

As aforementioned, most recent tool invocation studies are tree-based, which are first developed by ToolLLM (Qin et al., 2023). Compared to the ReAct (Yao et al., 2022) (or CoT (Wei et al., 2022)) that invokes tools through pipelined tool interaction, the tree-based methods replace the serial tool usage with the Depth First Search-based Decision Tree (DFSDT) algorithm. Consequentially, these methods increase the fault tolerance of LLMs and improve task planning capabilities. However, this comes at the cost of increased time complexity. This is mainly because DFSDT typically generates longer tool invocation sequences due to its backtracking mechanism, which involves multiple attempts at new nodes. While this improves task completion rates, it sacrifices execution efficiency (see Figure 1 for an example).

3.2 Divide-then-Aggregate Tool Invocation

To address the above problems, we propose the Divide-Then-Aggregate (DTA) tool invocation paradigm. This method allows LLMs to decompose the task, generate a set of parallel tool invocations, and aggregate the results after execution. DTA improves task planning and optimizes the reasoning framework for tool invocation, enabling LLMs to invoke tools efficiently in parallel and better tackle complex tasks.

Transforming the Serial Tool Using data to Parallel To steer the LLMs with the capabilities of parallel tool invocation, it is critical to construct the corresponding finetuning datasets. However, in most previous methods, e.g., vanilla CoT/ReAct methods or tree search-based algorithms like DFSDT, LLMs typically rely on invoking one tool at a time, which is not consistent with our setups. To this end, we utilize this type of data to transform it from a serial structure into a parallel structure.

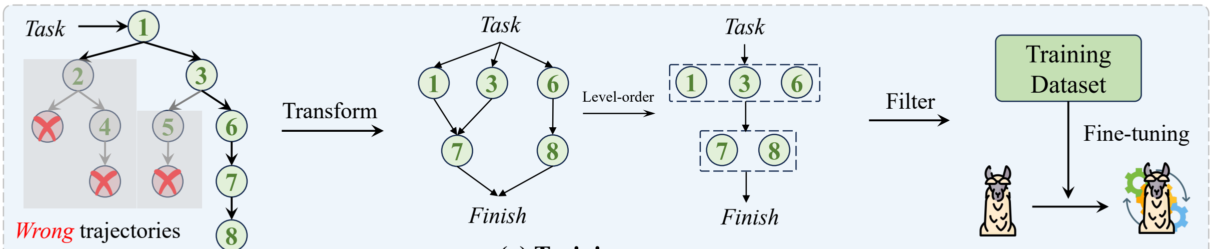
As shown in Figure 2, we first collect the serial successful tool innovation path from original tool searching trees. Given the tree-like tool searching trajectories generated by the tree search-based algorithm, it is inevitable that the trajectories contain redundant or erroneous paths. Therefore, we define the node series spanning from the root to the successful leaf node as successful path $\mathcal{P}$; we retain only the nodes in $\mathcal{P}$, while filtering out other nodes.

Next, we utilize a powerful LLM, to identify whether any tools in $\mathcal{P}$ can be executed in parallel. We choose GPT-4-turbo (OpenAI, 2023) to perform this task. If GPT-4 detects parallelizable tools, it establishes their relationships and organizes them into a Directed Acyclic Graph (DAG), represented as $\mathcal{G}$. The feasibility of parallel execution depends on input-output dependencies and logical causal relationships. For the nodes that cannot be parallelized, we retain their original structure[1].

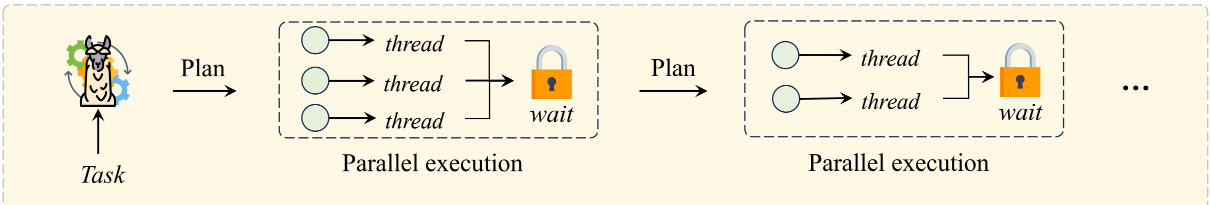
Finally, we construct the tool invocation mechanism by performing the level-order traversal on $\mathcal{G}$, enabling tools at the same level to be executed in parallel in a controlled manner and their outputs to be aggregated accordingly. This process embodies the *Divide-Then-Aggregate* strategy during the data construction phase. It is worth noting that we only transform the structure of $\mathcal{P}$ without modifying the

---

[1] For further details on transforming to the DAG structure using GPT-4, refer to Appendix A.



(a) Training process



(b) Multi-view inference

Figure 2: The figure illustrates the overall DTA-Llama pipeline. (a) depicts the construction of the DTA-Tool dataset and model training; (b) shows the tool invocation inference framework based on Process/Threads.

semantic content. Therefore, the objective of our approach is not to distill or compress the data, but rather to optimize its execution structure.

Data Filtering To ensure data quality, we apply filtering both before and after the data transformation. On the one hand, we filter out the raw serial data that contain incomplete tool invocation or cause task execution failures. Because including these noisy data could negatively affect the subsequent LLMs fine-tuning. On the other hand, we devise a rule-based filtering method to reduce some structural errors after the structural transformation. These rules leverage the acyclic nature of the DAG, along with its unique starting and ending points, to eliminate redundant tool invocation in cyclic invocation and situations where tool execution results cannot be aggregated. After these steps, we obtain DTA-Tool, a high-quality, DAG-based parallel tool invocation dataset with approximately 20k entries. We have documented the specific filtering rules in the appendix B.

Fine-tuning We fine-tune Llama-series models (Touvron et al., 2023a,b; Dubey et al., 2024) using DTA-Tool. Following previous work (Qin et al., 2023; Du et al., 2024; Liu et al., 2024), we employ a uniform system prompt to guide the LLMs in invoking tools based on user instructions. Details of the system prompt can be found in Appendix A. Our training approach has evolved from the traditional *Thought-Action-Observation* framework to a streamlined *Thought-Observation* framework. In this updated method, the original *Action* component is integrated into *Thought* as tool invocation plans. Consequently, the LLMs are trained to generate new *Thoughts* by considering the user instruction alongside the history of *Thoughts* and *Observations*. The training loss function is defined as follows:

$$\mathcal{L}(\theta) = -\log \sum_{i=1}^{n} p_\theta(y^i | q, y^{[1:i-1]}, o^{[1:i-1]}), \tag{1}$$

where y represents the *Thought* generated by the LLMs, q is the user instruction, and o is the *Observation*. The *Thought* generated in the i-th round depends on the *Thought* and *Observation* from the previous round. In the final round, the focus of learning shifts from the *Thought* to the final answer generated by LLMs. Finally, we fine-tuned the LLM with the DTA-Tool, resulting in DTA-Llama, which can invoke tools in parallel.

3.3 Process/Thread-based Inference

To support the modified LLMs, we developed a new inference framework, as illustrated in Figure 2(b). This framework redefines the *Thought-Observation* cycle based on CoT/ReAct, executing tool invocation in the form of Process/Threads.

Process Originally, *Thought* could only design an invocation strategy for a single tool, limiting its perceptual scope. In contrast, *Process* enhances the LLMs' ability to ***divide*** tasks and plan multiple parallelizable tool invocation strategies. Specifically, during each round, LLMs first evaluate the task's status and progress based on the historical trajectories. Then, LLMs analyze what needs to be done in the current step and decompose the task based on the available tools. Finally, LLMs sequentially generate a series of complete tool names along with their corresponding input parameters, which can be executed in parallel. This multi-tool approach helps broaden the perspective of LLMs, increasing the informational richness of each *Thought* step. In our framework, *Process* directly integrates *Action* into *Thought*. After careful deliberation, the LLMs provide a formalized tool invocation plan that can be extracted using regular expressions, facilitating subsequent execution by *Threads*.

Threads *Threads* refers to the steps that faithfully execute the tool strategies presented in the *Process*. In previous frameworks, *Thought* provides only one tool invocation strategy. However, once *Thought* is capable of proposing multiple tool strategies in parallel, the execution component must also support concurrent processing. To this end, we introduce *Threads*. All the tool invocation strategies provided by the *Process* are distributed across multiple *Threads*, which then independently and concurrently execute each strategy. Importantly, most real-time tool APIs inherently support a moderate degree of concurrency. Furthermore, the tool invocation plans proposed by *Process* are typically lightweight. As a result, even when multiple concurrent invocations target the same API, the level of concurrency remains within a tolerable range and does not pose a risk of overloading the service.

Intermediate State Lock When tools are invoked using *Threads*, the information processing load on the inference framework increases proportionally. The original *observation* only needed to record the execution result of one tool. Now, it must systematically link multiple tools and their corresponding results in an orderly manner. Otherwise, a disorganized *observation* could hinder the LLM's subsequent decision-making. To achieve this, we have specifically implemented a thread-oriented intermediate state lock at the end of each *Threads* round. The lock is only released once all *Threads* have completed their execution and returned the results. During the complete invocation process, the intermediate state lock regularly maintains communication between *Threads* and *Process*. The execution results of *Threads* are ***aggregated*** and used as part of the input to interact with the LLM, initiating the next round of *Process*. This cycle repeats until the task is completed.

Statistic	
# Data scale	21,342
# Average tool invocation rounds per data	2.46
# Average APIs required per data	3.48
% Percentage of parallel tool invocation data	99.1%

Table 1: Several important characteristics of DTA-Tools are presented in the table.

4 Experimental Setup

Dataset We use StableToolBench (Guo et al., 2024) for evaluation. All test cases in StableToolBench are actually derived from the test portion of ToolBench (Touvron et al., 2023b). Concretely, ToolBench is divided into six evaluation subsets based on tool categories and scenarios. The tool categories are as follows: *Inst.* denotes unseen instructions for the same set of tools in the training data, *Tool* denotes unseen tools within the same (seen) category as those in the training data, and *Cat.* denotes unseen tools from a different (unseen) category. The scenarios are: *I1* for single-tool instructions, *I2* for intra-category multi-tool instructions, and *I3* for intra-collection multi-tool instructions. The difficulty level of the task escalates progressively from *I1-Inst.* to *I3-Inst.*. Compared to the original ToolBench, StableToolBench introduces an extra caching system and an API simulator to mitigate the instability issues associated with real-time APIs.

As for the DTA-Llama training, we adopt the training portion of ToolBench and transform it into DTA-Tool style using GPT-4-turbo (OpenAI, 2023) through the method described in Sec.3.2. ToolBench provides a corresponding API set for each data point, enabling us to focus on tool learning without having to pay attention to tool retrieval. A detailed overview of DTA-Tool is presented in Table 1, with an instance provided in Appendix C.

Baselines We use both GPT-series and other open-source LLMs as our baselines. For the GPT-series models, we use OpenAI's GPT-3.5-turbo and GPT-4-turbo, leveraging their function calling capabilities[2]. And we include Parallel as an additional baseline paradigm alongside ReAct and DFSDT.

For open-source models, we fine-tune ToolLLaMA (Qin et al., 2023) from Llama2-7B on ToolBench and compare it using the ReAct and DFSDT

---

[2] While the exact mechanisms remain unclear, OpenAI has enabled parallel tool invocation in these models.

Method	I1-Inst.		I1-Tool		I1-Cat.		I2-Inst.		I2-Cat.		I3-Inst.		Average	
	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR
GPT-series														
GPT-3.5 (ReAct)	53.0	–	53.0	–	51.2	–	37.6	–	43.9	–	48.6	–	47.9	-
GPT-3.5 (DFSDT)	63.8	58.9	73.9	65.8	65.8	60.1	57.1	72.6	69.8	68.5	69.9	67.2	66.7	65.5
GPT-3.5 (Parallel)	64.6	48.5	65.0	55.7	69.0	54.2	54.9	55.7	61.4	53.2	56.6	50.8	61.9	53.0
GPT-4 (ReAct)	54.4	53.4	44.1	60.1	48.8	52.9	50.6	69.8	48.9	62.1	42.6	54.1	48.2	58.7
GPT-4 (DFSDT)	**69.0**	57.1	**69.6**	**66.5**	68.1	61.4	70.8	73.6	68.0	62.9	**76.0**	63.9	**70.3**	64.2
GPT-4 (Parallel)	62.9	**66.3**	67.4	61.4	**70.9**	**62.7**	**73.4**	**85.8**	**70.8**	**77.4**	69.7	**70.5**	69.2	**70.7**
Open-source														
ToolLLaMA (ReAct)	42.7	36.2	35.4	36.1	38.6	34.6	39.9	49.1	40.9	38.7	29.8	41.0	37.9	39.3
—-ToolLLaMA† (ReAct)	26.7	22.1	25.0	27.2	31.7	29.4	23.1	32.1	24.5	28.2	20.5	24.6	25.3	27.3
ToolLLaMA (DFSDT)	56.6	39.9	55.5	46.8	56.5	41.8	49.7	53.8	53.4	49.2	53.6	50.8	54.2	47.1
—-ToolLLaMA† (DFSDT)	41.8	35.6	39.9	37.3	44.9	39.9	36.0	47.2	39.1	39.5	33.3	26.2	39.2	37.6
LLMCompiler	39.2	35.6	35.1	36.0	39.8	35.3	37.5	45.6	38.4	38.1	27.0	36.5	36.2	37.9
Qwen2.5 (Parallel)	**65.7**	**54.0**	58.8	51.0	63.5	52.4	60.2	55.6	61.3	61.3	**68.3**	57.6	63.0	55.3
Ours	63.5	52.1	**64.2**	**53.2**	**67.2**	**54.2**	**62.1**	**70.8**	**71.9**	**65.3**	67.5	**59.0**	**66.1**	**59.1**

Table 2: A comparison different baselines and method on StableToolBench. Considering that real-world APIs are time-sensitive, the results of baselines presented in the table are reproduced during the period `from September to October 2024` using their official implementation. We highlight the best performance of GPT-series models and open-source models with the red and blue, respectively.

methods. LLMCompiler (Kim et al., 2023) is a non-training-based parallel tool invocation method that relies on system design and prompt engineering. We also use it as a baseline for comparison with Llama2-7B. Additionally, recent open-source LLMs, such as Qwen2.5 (Qwen Team, 2024), have demonstrated strong capabilities, including function calls and parallel tool invocation. Consequently, we include Qwen2.5-7B-Instruct in our baseline for comparison. To ensure experimental consistency, our method is fine-tuned on Llama2-7B for a fair comparison.

Evaluation Metrics StableToolBench introduces two key metrics to assess the tool learning capabilities of LLMs: Solvable Pass Rate (SoPR) and Solvable Win Rate (SoWR). SoPR measures the success rate of LLMs in solving tasks, while SoWR compares the quality of results against the GPT-3.5 (ReAct) baseline. All the results are averaged from three tests to minimize variance.

Besides the SoPR and SoWR, we expanded the StableToolBench evaluation to further investigate the efficiency of LLMs. We assess the computational cost of LLMs from two dimensions: token consumption and inference time, to better analyze their efficiency in task-solving.

Implementation Details We use the Llama-series models (Touvron et al., 2023a,b; Dubey et al., 2024) as our backbone, and fine-tune them for our task. Since Llama2 has a context length limit of 4096 tokens, shorter contexts may not be sufficient for effective tool invocation. To overcome this limitation, we followed the approach in ToolLLaMA (Qin et al., 2023) and applied position interpolation (Chen et al., 2023) to extend the context length to 8192 tokens. More details about our training process can be found in Appendix D.

5 Experiments

In this section, we first evaluate the performance of our method in tool learning tasks through extensive experiments in § 5.1. Next, we analyze its computational costs compared to baselines in § 5.2, and extend our method to different models to evaluate its generalizability in § 5.3. We showcase the practical workflow of DTA-Llama through case examples in Appendix E.

5.1 Main Experiments

SoPR As shown in Table 2, our method surpasses all open-source baselines [3]. While GPT series models, particularly GPT-4, show superior performance when compared to earlier open-source models, our approach not only surpasses GPT-3.5 but also competes with GPT-4. These results suggest that our method has a notable impact on enhancing the tool invocation capabilities of LLMs.

Moreover, under the same GPT model conditions, DFSDT and Parallel demonstrate similar per-

---

[3] To study the effect of training data, for each sample in the DTA-Tool, we collect its raw data in ToolBench and merge them together (denoted as DTA-Tool*); then, we re-train Llama2-7B on DTA-Tool* using the ReAct and DFSDT methods, denoting this version with †. However, the performance of this version is even inferior to the original, suggesting that data filtering is not the primary reason for the contribution.

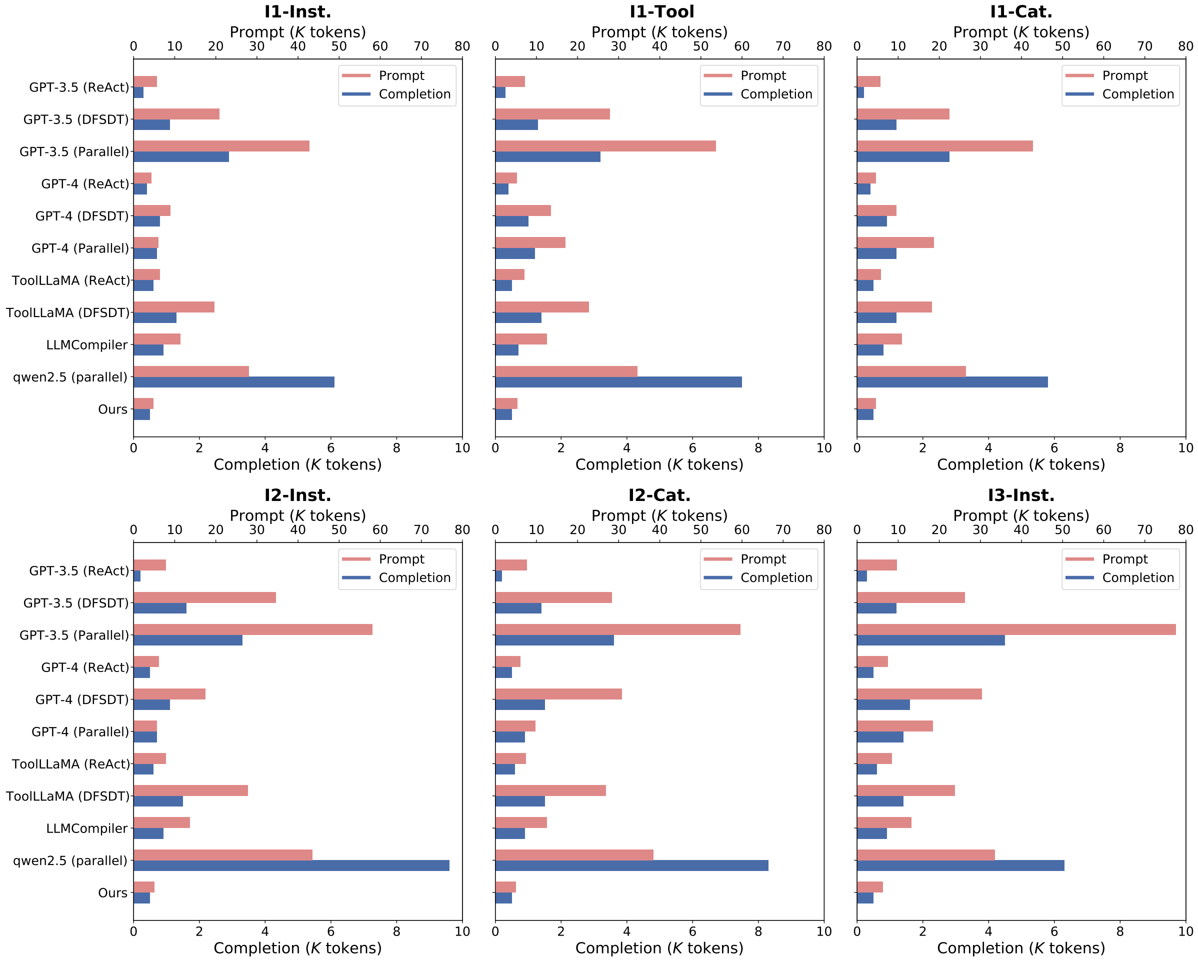


Figure 3: A comparison of all methods in terms of token consumption. The figure has two horizontal axes, representing *Prompt* and *Completion*, both measured in thousand tokens (K tokens)).

formances, suggesting that their parallel function call strategy does not notably enhance the tool invocation capabilities of GPT-based LLMs. In contrast, Qwen2.5 (Parallel) exhibited a clear improvement over DFSDT in open-source LLMs, with our method further advancing this performance. This demonstrates that our parallel mechanism is more effective than other parallel approaches, providing a greater boost to tool invocation capabilities.

SoWR The Solvable Pass Rate evaluates the quality of results against the GPT-3.5 (ReAct) baseline. As shown in Table 2, most open-source models achieve a SoWR score below 50, indicating their responses are of lower quality compared to GPT-3.5 (ReAct). In contrast, our method outperforms all open-source models, achieving a SoWR score of 70.8 on the I2-Inst dataset (27.32% relative improvement). Compared to GPT-3.5 DFSDT & Parallel, our method achieves nearly equivalent performance based solely on Llama2-7B, and even surpasses it in certain subsets.

These results, combined with SoPR, demonstrate that our method significantly enhances the ability of LLMs' tool utilization. We attribute this improvement to our specialized parallel tool invocation mechanism. This mechanism expands the decision-making scope of LLMs within one round, reducing the overall trajectories while enabling better decision-making. We provide example of the actual decision-making processes of different methods in the Appendix E to illustrate this.

5.2 Computational Cost

Token Consumption We count tokens for both *Completion* (output tokens generated by LLMs) and *Prompt* (input tokens provided to LLMs). Typically, *Completion* is more costly than *Prompt*. Figure 3 shows the token consumption for all methods. For each subset of StableToolBench, we calculate the average token count across all cases to assess the overall performance of the LLMs. The results in Figure 3 show that our approach is highly

Method	**I1-Inst.**	**I1-Tool**	**I1-Cat.**	**I2-Inst.**	**I2-Cat.**	**I3-Inst.**
GPT-series						
GPT-3.5 (ReAct)	4.28	4.75	4.48	5.16	5.05	5.31
GPT-3.5 (DFSDT)	11.60	13.36	11.77	16.60	14.06	12.54
GPT-3.5 (parallel)	25.33	28.06	26.12	31.79	31.04	38.10
GPT-4 (ReAct)	3.27	3.64	3.87	4.04	4.19	4.23
GPT-4 (DFSDT)	5.90	8.09	6.67	9.97	18.13	14.05
GPT-4 (parallel)	4.66	9.18	12.90	3.63	5.98	10.38
Open-source						
ToolLLaMA (ReAct)	3.42	3.47	3.50	3.67	3.63	3.64
ToolLLaMA (DFSDT)	8.09	8.51	8.10	10.20	9.93	9.23
LLMCompiler	5.48	5.56	6.07	5.36	5.68	5.62
Qwen2.5 (Parallel)	9.07	9.47	12.01	14.58	14.56	12.38
Ours	**2.41**	**2.41**	**2.51**	**2.32**	**2.34**	**2.48**

Table 3: A comparison of the number of inference steps across different methods.

competitive among open-source models. It significantly outperforms DFSDT while costing less than ReAct. Compared to GPT-based methods, our approach is particularly cost-effective, consuming fewer tokens. Given that DTA-Llama is a 7-billion-parameter model, its actual deployment costs are even lower. In addition, we also provide statistics on the maximum token consumption for each subset in Appendix F, to assess the performance of the LLMs in handling complex scenarios.

Furthermore, we analyze the inference steps of each baseline by calculating the average number of steps across all subsets. As demonstratd in Table 3, our method maintains its strong performance in terms of token consumption, requiring the fewest inference steps. In contrast, both the GPT series and Qwen2.5 models exhibit higher inference steps with their Parallel methods, which may be attributed to their limited task-planning capabilities. The LLMCompiler, a non-training approach, faces a bottleneck in processing complex tasks with 7-billion-parameter models. When considered alongside the results presented in Table 2, its performance falls short of expectations.

Inference Time Inference time is another key metric for assessing the computational cost of LLMs, which directly influences the deployment overhead of the service. Considering the intangibility of the GPT-series models and the impact of network latency on service requests, our inference time experiments focus solely on open-source LLMs. Similar to the setting of token consumption comparisons, we record the inference times for all cases within each subset and calculated the average. As shown in Table 4, our method demonstrates a clear advantage in inference time. Furthermore, combining the results from Table 2, we can conclude that our method delivers better performance at lower computational costs.

Dataset	ReAct	DFSDT	LLMCompiler	Qwen2.5	Ours
Inference latency (s)					
I1-Inst.	34	$76_{\uparrow 124\%}$	$58_{\uparrow 71\%}$	$104_{\uparrow 205\%}$	$\mathbf{29}_{\downarrow \mathbf{15}\%}$
I1-Tool	40	$103_{\uparrow 158\%}$	$67_{\uparrow 68\%}$	$111_{\uparrow 177\%}$	$\mathbf{33}_{\downarrow \mathbf{20}\%}$
I1-Cat.	35	$90_{\uparrow 157\%}$	$61_{\uparrow 74\%}$	$84_{\uparrow 140\%}$	$\mathbf{28}_{\downarrow \mathbf{20}\%}$
I2-Inst.	39	$110_{\uparrow 182\%}$	$83_{\uparrow 113\%}$	$153_{\uparrow 292\%}$	$\mathbf{29}_{\downarrow \mathbf{26}\%}$
I2-Cat.	38	$124_{\uparrow 226\%}$	$71_{\uparrow 87\%}$	$130_{\uparrow 241\%}$	$\mathbf{30}_{\downarrow \mathbf{21}\%}$
I3-Inst.	46	$120_{\uparrow 161\%}$	$69_{\uparrow 50\%}$	$135_{\uparrow 193\%}$	$\mathbf{40}_{\downarrow \mathbf{13}\%}$
Avg.	39	$104_{\uparrow 167\%}$	$68_{\uparrow 74\%}$	$119_{\uparrow 204\%}$	$\mathbf{31}_{\downarrow 21\%}$
Speed up (rate)					
I1-Inst.	1.00	×0.45	×0.59	×0.33	×**1.18**
I1-Tool	1.00	×0.39	×0.60	×0.36	×**1.20**
I1-Cat.	1.00	×0.39	×0.57	×0.42	×**1.20**
I2-Inst.	1.00	×0.31	×0.47	×0.26	×**1.27**
I2-Cat.	1.00	×0.35	×0.54	×0.29	×**1.35**
I3-Inst.	1.00	×0.38	×0.67	×0.34	×**1.15**
Avg.	1.00	×0.37	×0.57	×0.33	×**1.22**

Table 4: A comparison of the inference times of LLMs. Inference latency represents the average inference time across all cases in different subsets; Speed up indicates the factor of the inference speed being improved relative to ToolLLaMA (ReAct).

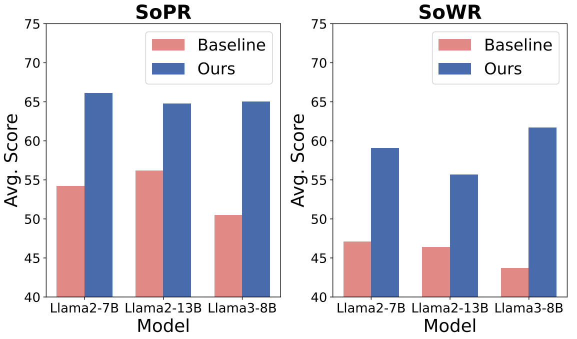


Figure 4: A comparison of different LLMs after fine-tuning with our method and the baseline method.

5.3 Generalizability

To validate whether our method can achieve similar improvement across a wide range of LLMs besides Llama2-7B, we select ToolLLaMA (DFSDT) as the baseline and conduct fine-tuning and testing on Llama2-13B and Llama3-8B. The average experimental results of six benchmark subsets are presented in Figure 4. The Llama2-7B results are derived from the *Averages* of ToolLLaMA (DFSDT) and DTA-Llama in Table 2. The results show that our method significantly outperforms the baselines across across all scales of LLMs, especially in the SoWR metric for Llama3-8B, where the improvement exceeds 40%. More detailed experimental results and analysis are provided in Appendix F.

6 Conclusion

In this paper, we introduce DTA-Llama, a novel tool learning approach based on the parallel invocation of tools through the iteratively Divide-Then-Aggregate paradigm. We construct the training data by transforming sequential data into a parallel DAG structure and use this data to train the model. Subsequently, we integrate a Process/Threads-based inference framework to enable LLMs to perform tool invocation in parallel. Extensive experimental results demonstrate that, compared to existing methods, DTA-Llama not only significantly improves performance but also substantially enhances the efficiency of tool learning in LLMs.

Limitations

This paper aims to advance the research of tool learning in LLMs, particularly in both industry and academia. However, due to limitations in human resources, computational power, and the current research conditions, there are certain constraints, as outlined below:

First, due to resource constraints, we were unable to conduct additional experiments on larger models to further validate the effectiveness of DTA-Llama. With the publication of this paper, we hope that more researchers in the field will attempt to build upon and extend our work.

Second, although our method shows improvements over existing tool learning approaches, LLMs still struggle to reliably and consistently address complex real-world problems through tool invocation. We hope to attract more researchers to the study of tool learning, as this area urgently requires more attention and resources.

References

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. 2023. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*.

Sijia Chen, Yibo Wang, Yi-Feng Wu, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2024. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. *arXiv preprint arXiv:2406.07115*.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Yanchu Guan, Dong Wang, Zhixuan Chu, Shiyu Wang, Feiyue Ni, Ruihua Song, and Chenyi Zhuang. 2024. Intelligent agents with llm-based process automation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5018–5027.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2024. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36.

Mengkang Hu, Yao Mu, Xinmiao Yu, Mingyu Ding, Shiguang Wu, Wenqi Shao, Qiguang Chen, Bin Wang, Yu Qiao, and Ping Luo. 2023. Tree-planner: Efficient close-loop task planning with large language models. *arXiv preprint arXiv:2310.08582*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, et al. 2023. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2023. An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. 2024. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2024. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36.

Zihan Ma, Minnan Luo, Hao Guo, Zhi Zeng, Yiran Hao, and Xiang Zhao. 2024. Event-radar: Event-driven multi-view learning for multimodal fake news detection. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5809–5821.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations*.

OpenAI. 2022. Openai: Introducing chatgpt.

OpenAI. 2023. Gpt-4 technical report.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824.

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*.

Cheng Qian, Chi Han, Yi Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6922–6939.

Cheng Qian, Chenyan Xiong, Zhenghao Liu, and Zhiyuan Liu. 2024. Toolink: Linking toolkit creation and using through chain-of-solving on open-source model. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 831–854.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. 2024a. Small llms are weak tool learners: A multi-llm agent. *arXiv preprint arXiv:2401.07324*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024b. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. 2024. Learning to use tools via cooperative and interactive agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10642–10657.

Zhengliang Shi, Shen Gao, Lingyong Yan, Yue Feng, Xiuyi Chen, Zhumin Chen, Dawei Yin, Suzan Verberne, and Zhaochun Ren. 2025. Tool learning in the wild: Empowering language models as automatic tool agents. In *Proceedings of the ACM on Web Conference 2025*, pages 2222–2237.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. 2006. *Operating System Concepts*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE.

Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. 2024. Lambda: A large model based data agent. *arXiv preprint arXiv:2407.17535*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, 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, et al. 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*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Shawn Ma, and Yitao Liang. 2024. Describe, explain, plan and select: interactive planning with llms enables open-world multi-task agents. *Advances in Neural Information Processing Systems*, 36.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R Fung, Hao Peng, and Heng Ji. 2023. Craft: Customizing llms by creating and retrieving from specialized toolsets. *arXiv preprint arXiv:2309.17428*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *arXiv preprint arXiv:2401.06201*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Dongsheng Zhu, Zhenyu Mao, Jinghui Lu, Rui Zhao, and Fei Tan. 2024a. Sda: Simple discrete augmentation for contrastive sentence representation learning. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 14459–14471.

Dongsheng Zhu, Daniel Tang, Weidong Han, Jinghui Lu, Yukun Zhao, Guoliang Xing, Junfeng Wang, and Dawei Yin. 2024b. Vislinginstruct: Elevating zero-shot learning in multi-modal language models with autonomous instruction optimization. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2122–2135.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A Rossi, Somdeb Sarkhel, and Chao Zhang. 2023. Toolchain*: Efficient action space navigation in large language models with a* search. *arXiv preprint arXiv:2310.13227*.

A Prompt

Data Transformation Prompt Figure 5 shows the prompts used during the data transformation process. We use this prompt with GPT-4-turbo to convert the serial data into a DAG structure. The prompt first evaluates the reasonableness of the conversation, discarding any data that doesn't meet the criteria. It then assesses whether the steps can be processed in parallel, generating a planning path within the DAG structure. The evaluations of reasonableness and parallelism follow distinct analytical processes, reflecting a form of internalized CoT prompt engineering. By applying these analyses to DAG generation, we achieve more accurate results.

System Prompt Figure 6 illustrates the system prompt used during both training and inference. Each user instruction is paired with a list of tool candidates, including both relevant and unrelated. The *tools* section contains the names and descriptions of all tools in the candidate set. Additionally, each tool is equipped with a set of APIs designed to handle various types of tasks. The *API* section is represented as a JSON list, providing detailed information about the names, descriptions, and parameters of the APIs associated with each tool.

B Data Filtering Rules

The rules for data filtering are divided into before and after structural transformation. The specific rules are listed in text form in Figure 7. These rules can be easily implemented in code to filter the data.

C DTA-Tool Instance

Figure 8 illustrates a data entry in DTA-Tool, stored in JSON format. The outer layer consists of two keys: "id" and "conversations". The "id" represents the user's instruction, while "conversations" details the task execution process by the LLMs. Within "conversations", there are four roles: "system", "user", "assistant", and "function". The "system" role, which represents the system prompt, is introduced in Appendix A and is omitted here; "user" is the same as "id" and reflects the user's instruction; "assistant" represents the LLMs' reasoning process (marked by *Thought*) and provides a specific tool invocation plan (marked by *Function Call*); "function" contains the result of the tool execution. When tool invocation are parallel, the results are concatenated sequentially.

D Training Details

We train the LLMs using multi-round conversations with the following hyperparameters: for Llama2-7B and Llama3-8B, the learning rate is 5×10^{-5}, warmup ratio is 4×10^{-2}, with 4 epochs, a batch size of 64, and a maximum sequence length of 8192. All other settings are default. Training is performed on 8 × A100 GPUs, while evaluation is done on one A100 GPU. For Llama2-13B, the hyperparameters are similar, with a learning rate of 5×10^{-5}, warmup ratio of 4×10^{-2}, 5 epochs, a batch size of 64, and a maximum sequence length of 8192. The model is trained on 8 × A100 GPUs, and evaluation is conducted on 4 × A100 GPUs.

E Case Study

Comparison of decision process Figure 9 illustrates the actual tool invocation decision-making processes of ReAct, DFSDT, and our method.

Complete DTA-Llama performance In Figure 10, we present several cases to showcase the performance of DTA-Llama on practical tasks. Each case consists of the user's Question (instruction), the LLM's Output, and the Tool Response. The LLM's Output represents the *Process*, which includes the LLMs' thought and tool invocation strategy. After the execution of *Threads*, the Tool Response presents the results of the tools' execution.

F Supplementary of the Experiments

More Details on Token Consumption In Table 5, we present the maximum token consumption for each method across various subsets. As shown, the ReAct method has the lowest maximum token count, with our method following closely behind. The ReAct method employs a simple and straightforward task-planning mechanism, leading to low resource consumption but a very low success rate, which makes it ill-suited for handling complex real-time tasks. In contrast, our method efficiently utilizes additional tokens to tackle more challenging tasks without significantly increasing token usage, demonstrating superior cost-effectiveness.

Detailed Generalization Experiments Table 6 presents the detailed results of the generalization experiments conducted on Llama2-13B and Llama3-8B. Using ToolLLaMA (DFSDT) as the baseline, our method demonstrates significant improvements over the baseline across all subsets.

```
You are AutoGPT, possessing powerful capabilities to optimize the API function call process.
In the following, I will provide the contents of "QUERY" and "CONVERSATIONS".
"QUERY" represents the user's request, which is the problem to be solved.
"CONVERSATIONS" records the entire process of solving "QUERY" through the invocation of related API functions, encompassing information from "system," "user," and "assistant." Here, "system" represents system configuration information, and "user" refers to information provided by the user. The "assistant" part documents the detailed steps of problem-solving, primarily divided into two main sections. The first part is "value," where "Thought" shows the thought process of the assistant during the function call, "Action" refers to the specific function name called, and "Action Input" is the parameters passed to the function. The second part is "function," which displays the actual results obtained after calling the API function.
Your task is to adjust "CONVERSATIONS" based on "QUERY" according to the following requirements:
1. First, you need to determine whether there are severe logical confusion issues in "CONVERSATIONS". If the confusion is so severe that it cannot be amended, deem "CONVERSATIONS" as unreasonable, and answer "No" in the "reasonable" field of your response. If there is no logical confusion or only minor issues, deem "CONVERSATIONS" as reasonable and answer "Yes". Note, if the answer is "No", disregard all subsequent requirements and return the response directly.
2. Then, analyze in "CONVERSATIONS" which steps can be called in parallel, meaning these steps do not have a dependency relationship. Whether or not there are steps that can be parallelized, provide convincing reasons in the "analysis" field of your response.
3. If there are steps that can be parallelized, answer "Yes" in the "parallelizable" field; otherwise, answer "No". If the answer is "No", disregard all subsequent requirements and return the response directly.
4. When the answer to "parallelizable" is "Yes", first provide an analysis process, followed by a complete function call process. The process should be represented as a Directed Acyclic Graph (DAG), using the sequence numbers of "steps" to build the DAG, and must include all serial and parallel steps end-to-end. Provide this in the "DAG" field of your response.

Please return your answer in JSON format, as shown in the example below:
{"reasonable": "Yes", "reasonable_analysis": "...", "parallelizable": "Yes", "parallelizable_analysis": "...", "DAG": "1->2,2->3,3->4,2->5,4->6,5->6"}

Now, please respond based on the "QUERY" and "CONVERSATIONS":
<QUERY>
{query}
</QUERY>

<CONVERSATIONS>
{conversations}
```

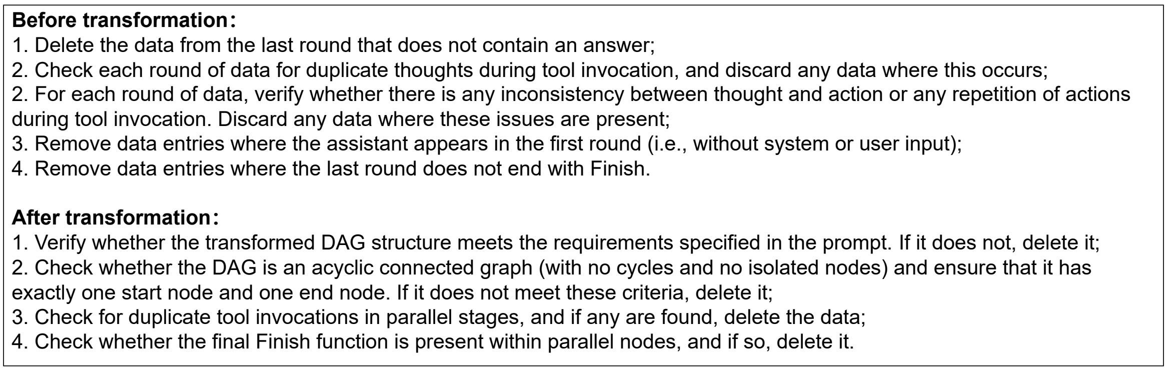
Figure 5: The prompt used for data transformation: in this context, *{query}* represents the instruction from the user, while *{conversations}* refers to the original conversation content in the ToolBench data.

```
You are AutoGPT, you can use many tools(functions) to do the following task.
First I will give you the task description, and your task start.
At each step, you need to give your thought to analyze the status now and what to do next, with one or more function calls to actually excute your step. Functions within the same step should not have dependencies on each other. Your output should follow this format:
Thought:
Function Call:

After the call, you will get the call result, and you are now in a new state.
Then you will analyze your status now, then decide what to do next...
After many (Thought-call) pairs, you finally perform the task, then you can give your finial answer.
Remember:
1.the state change is irreversible, you can't go back to one of the former state, if you want to restart the task, say "I give up and restart".
2.All the thought is short, at most in 5 sentence.
3.You can do more then one trys, so if your plan is to continusly try some conditions, you can do one of the conditions per try.
Let's Begin!
Task description: You should use functions to help handle the real time user querys. Remember:
1.ALWAYS call "Finish" function at the end of the task. And the final answer should contain enough information to show to the user,If you can't handle the task, or you find that function calls always fail(the function is not valid now), use function Finish->give_up_and_restart.
2.Do not use origin tool names, use only subfunctions' names.
You have access of the following tools:
{tools}

Specifically, you have access to the following APIs:
{APIs}
```

Figure 6: A unified system prompt is used during both training and inference. It must be input into the LLMs at the beginning of the conversation.



Before transformation:
1. Delete the data from the last round that does not contain an answer;
2. Check each round of data for duplicate thoughts during tool invocation, and discard any data where this occurs;
2. For each round of data, verify whether there is any inconsistency between thought and action or any repetition of actions during tool invocation. Discard any data where these issues are present;
3. Remove data entries where the assistant appears in the first round (i.e., without system or user input);
4. Remove data entries where the last round does not end with Finish.

After transformation:
1. Verify whether the transformed DAG structure meets the requirements specified in the prompt. If it does not, delete it;
2. Check whether the DAG is an acyclic connected graph (with no cycles and no isolated nodes) and ensure that it has exactly one start node and one end node. If it does not meet these criteria, delete it;
3. Check for duplicate tool invocations in parallel stages, and if any are found, delete the data;
4. Check whether the final Finish function is present within parallel nodes, and if so, delete it.

Figure 7: The image illustrates the detailed data filtering rules.

Method	**I1-Inst.**		**I1-Tool**		**I1-Cat.**		**I2-Inst.**		**I2-Cat.**		**I3-Inst.**		**Average**	
	Com.	Pro.	Com.	Pro.	Com.	Pro.	Com.	Pro.	Com.	Pro.	Com.	Pro.	Com.	Pro.
GPT-series														
GPT-3.5 (ReAct)	1.3	17.1	1.9	**42.8**	2.1	**14.7**	1.5	**16.1**	**1.5**	**17.6**	**0.9**	**23.7**	1.5	**22.0**
GPT-3.5 (DFSDT)	7.7	171.1	10.0	496.0	12.8	296.8	7.9	158.8	13.3	194.0	10.1	280.1	10.3	266.1
GPT-3.5 (Parallel)	20.1	321.9	30.5	578.6	15.7	252.6	16.0	261.2	19.5	297.1	17.3	322.8	19.9	325.3
GPT-4 (ReAct)	**1.2**	**15.8**	**1.2**	43.6	**0.9**	17.9	**1.3**	23.9	1.5	23.7	0.9	33.7	**1.2**	26.4
GPT-4 (DFSDT)	6.0	109.4	10.1	301.2	4.9	114.2	11.3	354.1	16.2	380.2	13.2	649.5	10.3	318.1
GPT-4 (Parallel)	5.0	88.5	19.4	482.2	21.6	440.8	4.0	46.2	4.6	118.8	13.9	231.0	11.4	234.6
Open-source														
ToolLLaMA (ReAct)	**1.5**	**21.2**	**1.5**	**41.3**	**1.6**	**17.7**	**1.3**	**14.8**	**1.4**	**18.4**	1.5	**18.7**	**1.5**	**22.0**
ToolLLaMA (DFSDT)	5.0	96.4	8.4	138.5	7.8	118.0	4.6	113.7	6.6	106.5	4.4	99.8	6.1	112.2
LLMCompiler	3.2	42.3	3.6	71.9	4.2	96.2	2.0	27.1	2.7	42.6	3.1	36.6	3.1	52.8
Qwen2.5 (Parallel)	42.9	197.1	39.2	215.5	39.5	176.8	41.0	182.0	41.6	195.6	33.7	162.2	39.7	188.2
Ours	1.8	23.4	4.2	59.8	4.4	60.9	1.5	29.5	1.4	22.2	**1.3**	25.3	2.4	36.9

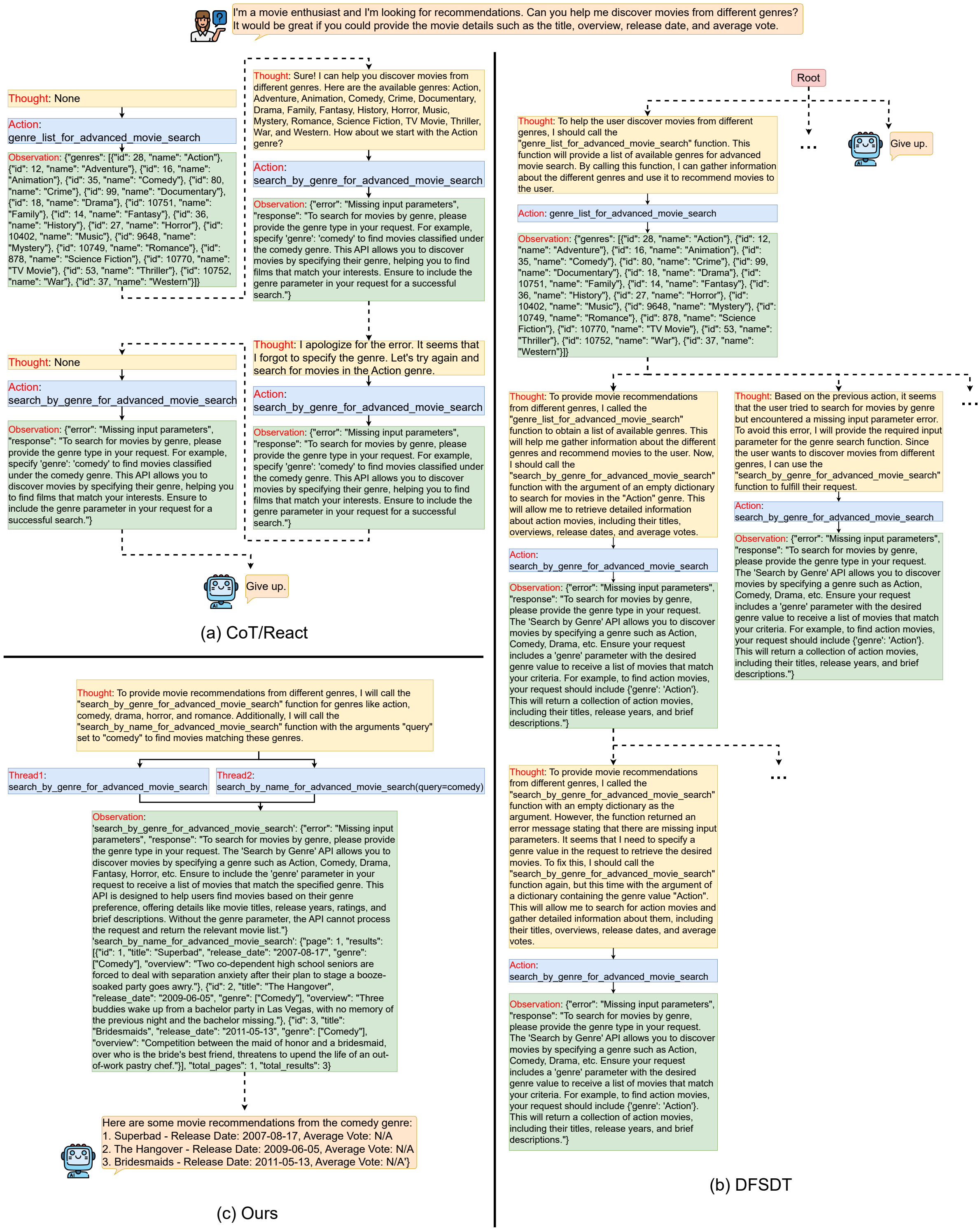
Table 5: A comparison of the maximum token consumption across all methods. All values in the table are given in thousands. On the horizontal axis, "Com." represents *Completion*, and "Pro." represents *Prompt*.

Method	**I1-Inst.**		**I1-Tool**		**I1-Cat.**		**I2-Inst.**		**I2-Cat.**		**I3-Inst.**		**Average**	
	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR	SoPR	SoWR
Llama2-13B (baseline)	57.6	40.4	60.9	41.2	52.3	41.8	55.0	48.4	59.7	55.7	51.4	50.8	56.2	46.4
Llama2-13B (Ours)	60.6	46.0	61.1	51.6	67.7	48.7	61.5	57.3	65.6	69.8	72.4	60.7	64.8	55.7
Llama3-8B (baseline)	51.5	38.7	54.6	40.5	51.1	39.9	48.3	46.0	43.1	52.8	54.1	44.3	50.5	43.7
Llama3-8B (Ours)	63.7	61.3	64.2	51.6	63.2	65.2	64.0	68.5	67.7	70.8	67.2	52.5	65.0	61.7

Table 6: A comparison of the generalization experiment results on StableToolBench between Llama2-13B and Llama3-8B.

```
{
   "id": "I recently bought an NFT with the contract address 0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D and
tokenId 4671. Can you provide me with the metadata, attributes, and enclosed media for this NFT? Additionally, I would
like to know the current circulation supply of ISLAMICOIN.",
   "conversations": [
     {
       "from": "system",
       "value": System prompt
     },
     {
       "from": "user",
       "value": "\nI recently bought an NFT with the contract address
0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D and tokenId 4671. Can you provide me with the metadata,
attributes, and enclosed media for this NFT? Additionally, I would like to know the current circulation supply of
ISLAMICOIN.\nBegin!\n"
     },
     {
       "from": "assistant",
       "value": "Thought: I am using the 'getnfts_metadata_for_nfts_by_address' function to retrieve metadata, attributes,
and media for the NFT with contract address 0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D and tokenId 4671.
This will help fulfill the user's request for information about their recently purchased NFT. Additionally, I will use the
'islamiblockchain_islamicoin_main_cs_txt_for_islamicoin' function to retrieve the current circulation supply of
ISLAMICOIN, enabling me to respond accurately to the user's query about the current supply. \nFunction Call:
[{\"name\": \"getnfts_metadata_for_nfts_by_address\", \"arguments\": {\"contractaddress\":
\"0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D\", \"tokenid\": 4671}}, {\"name\":
\"islamiblockchain_islamicoin_main_cs_txt_for_islamicoin\", \"arguments\": {}}]"
     },
     {
       "from": "function",
       "value": "{'name': 'getnfts_metadata_for_nfts_by_address', 'arguments': {'contractaddress':
 '0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D', 'tokenid': 4671}, 'result': {'error': '', 'response': ' {\"\\'contract\":
 {\"address\":\"0xbc4ca0eda7647a8ab7c2061c2e118a18a936f13d\"},\"id\": {\"tokenId\":\"4671\",\"tokenMetadata\":
{\"tokenType\":\"ERC721\"}},\"title\":\"\",\"description\":\"\",\"tokenUri\":{\"gateway\":\"https://alchemy.mypinata.cloud/ipfs/
QmeSjSinHpPnmXmspMjwiXyN6zS4E9zccariGR3jxcaWtq/4671\",\"raw\":\"ipfs://QmeSjSinHpPnmXmspMjwiXyN6zS4E
9zccariGR3jxcaWtq/4671\"},\"media\": [{\"gateway\":\"https://nft-cdn.alchemy.com/eth-mainnet/bf2831daf6b80b432ff616
16cd63c468\",\"thumbnail\":\"https://res.cloudinary.com/alchemyapi/image/upload/thumbnailv2/eth-mainnet/bf2831daf6
b80b432ff61616cd63c468\",\"raw\":\"ipfs://Qmci1pGaUmvb6StPxdGp1WqfK9QjPjdf43nCbY5LJ9y1MY\",\"format\":\"png
\",\"bytes\": 125061}],\"metadata\": {\"image\":\"ipfs://Qmci1pGaUmvb6StPxdGp1WqfK9QjPjdf43nCbY5LJ9y1MY\",
\"attributes\": [{\"value\":\"Black\",\"trait_type\":\"Fur\"}, {\"value\":\"Bored\",\"trait_type\":\"Mouth\"}, {\"value\":\"Party Hat
1\",\"trait_type\":\"Hat\"}, {\"value\":\"Gold Stud\",\"trait_type\":\"Earring\"}, {\"value\":\"Aquamarine\",\"trait_type\":
\\'Back...'}}"
     },
     {
       "from": "function",
       "value": "{'name': 'islamiblockchain_islamicoin_main_cs_txt_for_islamicoin', 'arguments': {}, 'result': {'error': '',
'response': '3,069,324,811\\n'}}"
     },
     {
       "from": "assistant",
       "value": "Thought: The user has gathered the NFT metadata and the ISLAMICOIN circulation supply. They will
format this information clearly and use the \"Finish\" function to provide a comprehensive answer to the user. \nFunction
 Call: [{\"name\": \"Finish\", \"arguments\": {\"return_type\": \"give_answer\", \"final_answer\": \"The metadata, attributes,
and enclosed media for the NFT with contract address 0xBC4CA0EdA7647A8aB7C2061c2E118A18a936f13D and
tokenId 4671 are as follows:\\n\\n- Title: \\n- Description: \\n- Token URI:
[ipfs://QmeSjSinHpPnmXmspMjwiXyN6zS4E9zccariGR3jxcaWtq/4671]
(https://alchemy.mypinata.cloud/ipfs/QmeSjSinHpPnmXmspMjwiXyN6zS4E9zccariGR3jxcaWtq/4671)\\n- Media:
[ipfs://Qmci1pGaUmvb6StPxdGp1WqfK9QjPjdf43nCbY5LJ9y1MY](https://nft-cdn.alchemy.com/eth-
mainnet/bf2831daf6b80b432ff61616cd63c468)\\n\\nAttributes:\\n- Fur: Black\\n- Mouth: Bored\\n- Hat: Party Hat 1\\n-
 Earring: Gold Stud\\n- Back: Aquamarine\\nThe current circulation supply of ISLAMICOIN is 3,069,324,811.\\n\\nIf you
 have any more questions or need further assistance, feel free to ask!\"}}]"
     }
```

Figure 8: An instance of a data entry in DTA-Tool.



I'm a movie enthusiast and I'm looking for recommendations. Can you help me discover movies from different genres? It would be great if you could provide the movie details such as the title, overview, release date, and average vote.

Thought: None

Action: genre_list_for_advanced_movie_search

Observation: {"genres": [{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 16, "name": "Animation"}, {"id": 35, "name": "Comedy"}, {"id": 80, "name": "Crime"}, {"id": 99, "name": "Documentary"}, {"id": 18, "name": "Drama"}, {"id": 10751, "name": "Family"}, {"id": 14, "name": "Fantasy"}, {"id": 36, "name": "History"}, {"id": 27, "name": "Horror"}, {"id": 10402, "name": "Music"}, {"id": 9648, "name": "Mystery"}, {"id": 10749, "name": "Romance"}, {"id": 878, "name": "Science Fiction"}, {"id": 10770, "name": "TV Movie"}, {"id": 53, "name": "Thriller"}, {"id": 10752, "name": "War"}, {"id": 37, "name": "Western"}]}

Thought: Sure! I can help you discover movies from different genres. Here are the available genres: Action, Adventure, Animation, Comedy, Crime, Documentary, Drama, Family, Fantasy, History, Horror, Music, Mystery, Romance, Science Fiction, TV Movie, Thriller, War, and Western. How about we start with the Action genre?

Action: search_by_genre_for_advanced_movie_search

Observation: {"error": "Missing input parameters", "response": "To search for movies by genre, please provide the genre type in your request. For example, specify 'genre': 'comedy' to find movies classified under the comedy genre. This API allows you to discover movies by specifying their genre, helping you to find films that match your interests. Ensure to include the genre parameter in your request for a successful search."}

Thought: I apologize for the error. It seems that I forgot to specify the genre. Let's try again and search for movies in the Action genre.

Action: search_by_genre_for_advanced_movie_search

Observation: {"error": "Missing input parameters", "response": "To search for movies by genre, please provide the genre type in your request. For example, specify 'genre': 'comedy' to find movies classified under the comedy genre. This API allows you to discover movies by specifying their genre, helping you to find films that match your interests. Ensure to include the genre parameter in your request for a successful search."}

Thought: None

Action: search_by_genre_for_advanced_movie_search

Observation: {"error": "Missing input parameters", "response": "To search for movies by genre, please provide the genre type in your request. For example, specify 'genre': 'comedy' to find movies classified under the comedy genre. This API allows you to discover movies by specifying their genre, helping you to find films that match your interests. Ensure to include the genre parameter in your request for a successful search."}

Give up.

(a) CoT/React

Thought: To provide movie recommendations from different genres, I will call the "search_by_genre_for_advanced_movie_search" function for genres like action, comedy, drama, horror, and romance. Additionally, I will call the "search_by_name_for_advanced_movie_search" function with the arguments "query" set to "comedy" to find movies matching these genres.

Thread1: search_by_genre_for_advanced_movie_search

Thread2: search_by_name_for_advanced_movie_search(query=comedy)

Observation: 'search_by_genre_for_advanced_movie_search': {"error": "Missing input parameters", "response": "To search for movies by genre, please provide the genre type in your request. The 'Search by Genre' API allows you to discover movies by specifying a genre such as Action, Comedy, Drama, Fantasy, Horror, etc. Ensure to include the 'genre' parameter in your request to receive a list of movies that match the specified genre. This API is designed to help users find movies based on their genre preference, offering details like movie titles, release years, ratings, and brief descriptions. Without the genre parameter, the API cannot process the request and return the relevant movie list."}
'search_by_name_for_advanced_movie_search': {"page": 1, "results": [{"id": 1, "title": "Superbad", "release_date": "2007-08-17", "genre": ["Comedy"], "overview": "Two co-dependent high school seniors are forced to deal with separation anxiety after their plan to stage a booze-soaked party goes awry."}, {"id": 2, "title": "The Hangover", "release_date": "2009-06-05", "genre": ["Comedy"], "overview": "Three buddies wake up from a bachelor party in Las Vegas, with no memory of the previous night and the bachelor missing."}, {"id": 3, "title": "Bridesmaids", "release_date": "2011-05-13", "genre": ["Comedy"], "overview": "Competition between the maid of honor and a bridesmaid, over who is the bride's best friend, threatens to upend the life of an out-of-work pastry chef."}], "total_pages": 1, "total_results": 3}

Here are some movie recommendations from the comedy genre:
1. Superbad - Release Date: 2007-08-17, Average Vote: N/A
2. The Hangover - Release Date: 2009-06-05, Average Vote: N/A
3. Bridesmaids - Release Date: 2011-05-13, Average Vote: N/A'}

(c) Ours

Thought: To help the user discover movies from different genres, I should call the "genre_list_for_advanced_movie_search" function. This function will provide a list of available genres for advanced movie search. By calling this function, I can gather information about the different genres and use it to recommend movies to the user.

Action: genre_list_for_advanced_movie_search

Observation: {"genres": [{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 16, "name": "Animation"}, {"id": 35, "name": "Comedy"}, {"id": 80, "name": "Crime"}, {"id": 99, "name": "Documentary"}, {"id": 18, "name": "Drama"}, {"id": 10751, "name": "Family"}, {"id": 14, "name": "Fantasy"}, {"id": 36, "name": "History"}, {"id": 27, "name": "Horror"}, {"id": 10402, "name": "Music"}, {"id": 9648, "name": "Mystery"}, {"id": 10749, "name": "Romance"}, {"id": 878, "name": "Science Fiction"}, {"id": 10770, "name": "TV Movie"}, {"id": 53, "name": "Thriller"}, {"id": 10752, "name": "War"}, {"id": 37, "name": "Western"}]}

Thought: To provide movie recommendations from different genres, I called the "genre_list_for_advanced_movie_search" function to obtain a list of available genres. This will help me gather information about the different genres and recommend movies to the user. Now, I should call the "search_by_genre_for_advanced_movie_search" function with the argument of an empty dictionary to search for movies in the "Action" genre. This will allow me to retrieve detailed information about action movies, including their titles, overviews, release dates, and average votes.

Action: search_by_genre_for_advanced_movie_search

Observation: {"error": "Missing input parameters", "response": "To search for movies by genre, please provide the genre type in your request. The 'Search by Genre' API allows you to discover movies by specifying a genre such as Action, Comedy, Drama, etc. Ensure your request includes a 'genre' parameter with the desired genre value to receive a list of movies that match your criteria. For example, to find action movies, your request should include {'genre': 'Action'}. This will return a collection of action movies, including their titles, release years, and brief descriptions."}

Thought: To provide movie recommendations from different genres, I called the "search_by_genre_for_advanced_movie_search" function with an empty dictionary as the argument. However, the function returned an error message stating that there are missing input parameters. It seems that I need to specify a genre value in the request to retrieve the desired movies. To fix this, I should call the "search_by_genre_for_advanced_movie_search" function again, but this time with the argument of a dictionary containing the genre value "Action". This will allow me to search for action movies and gather detailed information about them, including their titles, overviews, release dates, and average votes.

Action: search_by_genre_for_advanced_movie_search

Observation: {"error": "Missing input parameters", "response": "To search for movies by genre, please provide the genre type in your request. The 'Search by Genre' API allows you to discover movies by specifying a genre such as Action, Comedy, Drama, etc. Ensure your request includes a 'genre' parameter with the desired genre value to receive a list of movies that match your criteria. For example, to find action movies, your request should include {'genre': 'Action'}. This will return a collection of action movies, including their titles, release years, and brief descriptions."}

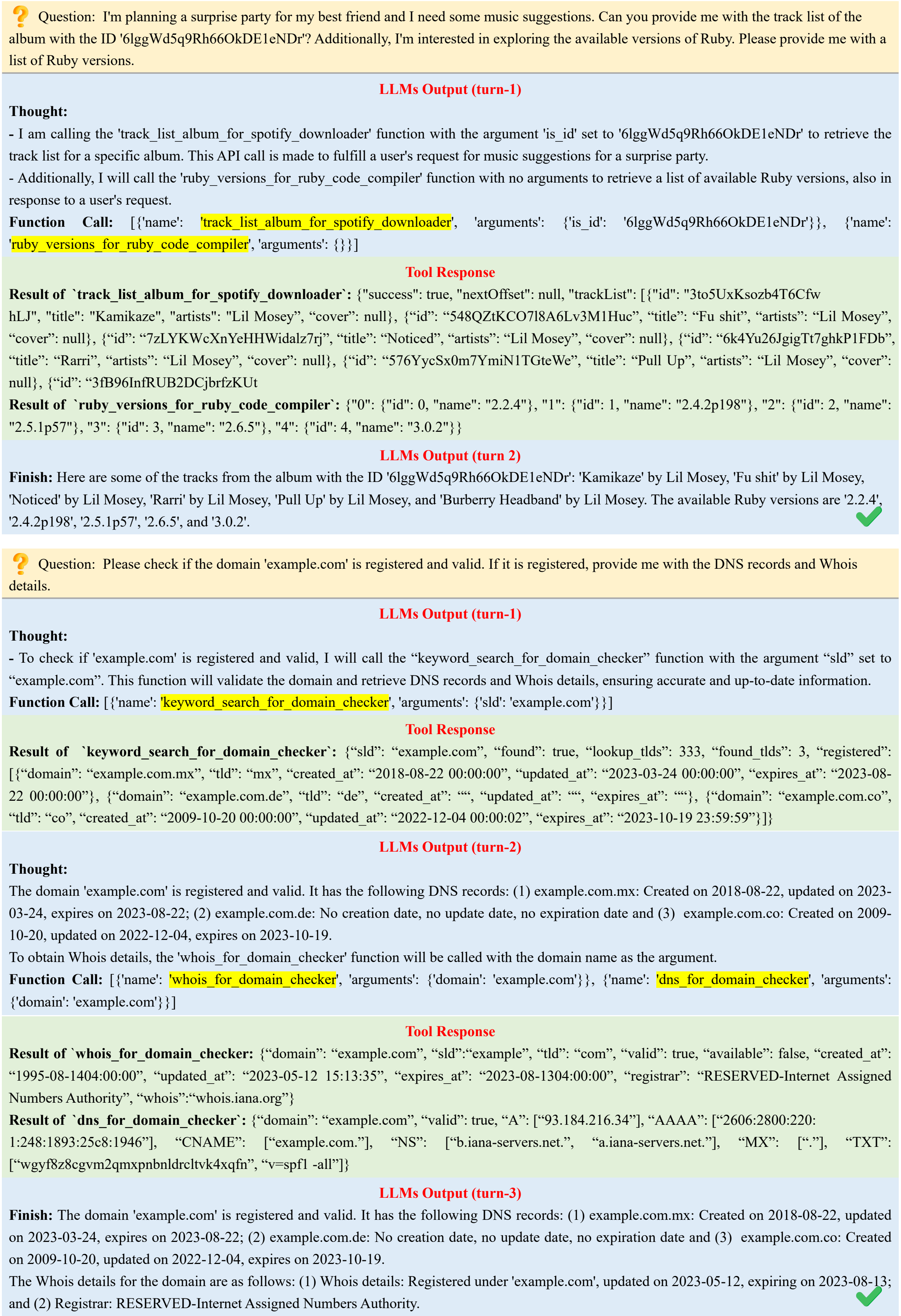
Thought: Based on the previous action, it seems that the user tried to search for movies by genre but encountered a missing input parameter error. To avoid this error, I will provide the required input parameter for the genre search function. Since the user wants to discover movies from different genres, I can use the "search_by_genre_for_advanced_movie_search" function to fulfill their request.

Action: search_by_genre_for_advanced_movie_search

Observation: {"error": "Missing input parameters", "response": "To search for movies by genre, please provide the genre type in your request. The 'Search by Genre' API allows you to discover movies by specifying a genre such as Action, Comedy, Drama, etc. Ensure your request includes a 'genre' parameter with the desired genre value to receive a list of movies that match your criteria. For example, to find action movies, your request should include {'genre': 'Action'}. This will return a collection of action movies, including their titles, release years, and brief descriptions."}

(b) DFSDT

Figure 9: A comparison of the real decision-making visualization processes between CoT/React, DFSDT, and our method: (a) The serial approach is straightforward but prone to getting stuck in local loops when errors occur, leading to task failure; (b) The tree-based decision-making approach attempts various possibilities through deep traversal and backtracking, but due to its narrow decision scope, it involves a large amount of redundant processes; (c) Our method maximizes the parallel use of available tools during the *Threads* phase, enhancing the perception of LLMs and resulting in an efficient and accurate final outcome.



❓ Question: I'm planning a surprise party for my best friend and I need some music suggestions. Can you provide me with the track list of the album with the ID '6lggWd5q9Rh66OkDE1eNDr'? Additionally, I'm interested in exploring the available versions of Ruby. Please provide me with a list of Ruby versions.

LLMs Output (turn-1)

Thought:

- I am calling the 'track_list_album_for_spotify_downloader' function with the argument 'is_id' set to '6lggWd5q9Rh66OkDE1eNDr' to retrieve the track list for a specific album. This API call is made to fulfill a user's request for music suggestions for a surprise party.
- Additionally, I will call the 'ruby_versions_for_ruby_code_compiler' function with no arguments to retrieve a list of available Ruby versions, also in response to a user's request.

Function Call: [{'name': 'track_list_album_for_spotify_downloader', 'arguments': {'is_id': '6lggWd5q9Rh66OkDE1eNDr'}}, {'name': 'ruby_versions_for_ruby_code_compiler', 'arguments': {}}]

Tool Response

Result of `track_list_album_for_spotify_downloader`: {"success": true, "nextOffset": null, "trackList": [{"id": "3to5UxKsozb4T6CfwhLJ", "title": "Kamikaze", "artists": "Lil Mosey", “cover”: null}, {“id”: “548QZtKCO7l8A6Lv3M1Huc”, “title”: “Fu shit”, “artists”: “Lil Mosey”, “cover”: null}, {“id”: “7zLYKWcXnYeHHWidalz7rj”, “title”: “Noticed”, “artists”: “Lil Mosey”, “cover”: null}, {“id”: “6k4Yu26JgigTt7ghkP1FDb”, “title”: “Rarri”, “artists”: “Lil Mosey”, “cover”: null}, {“id”: “576YycSx0m7YmiN1TGteWe”, “title”: “Pull Up”, “artists”: “Lil Mosey”, “cover”: null}, {“id”: “3fB96InfRUB2DCjbrfzKUt

Result of `ruby_versions_for_ruby_code_compiler`: {"0": {"id": 0, "name": "2.2.4"}, "1": {"id": 1, "name": "2.4.2p198"}, "2": {"id": 2, "name": "2.5.1p57"}, "3": {"id": 3, "name": "2.6.5"}, "4": {"id": 4, "name": "3.0.2"}}

LLMs Output (turn 2)

Finish: Here are some of the tracks from the album with the ID '6lggWd5q9Rh66OkDE1eNDr': 'Kamikaze' by Lil Mosey, 'Fu shit' by Lil Mosey, 'Noticed' by Lil Mosey, 'Rarri' by Lil Mosey, 'Pull Up' by Lil Mosey, and 'Burberry Headband' by Lil Mosey. The available Ruby versions are '2.2.4', '2.4.2p198', '2.5.1p57', '2.6.5', and '3.0.2'. ✓

❓ Question: Please check if the domain 'example.com' is registered and valid. If it is registered, provide me with the DNS records and Whois details.

LLMs Output (turn-1)

Thought:

- To check if 'example.com' is registered and valid, I will call the “keyword_search_for_domain_checker” function with the argument “sld” set to “example.com”. This function will validate the domain and retrieve DNS records and Whois details, ensuring accurate and up-to-date information.

Function Call: [{'name': 'keyword_search_for_domain_checker', 'arguments': {'sld': 'example.com'}}]

Tool Response

Result of `keyword_search_for_domain_checker`: {“sld”: “example.com”, “found”: true, “lookup_tlds”: 333, “found_tlds”: 3, “registered”: [{“domain”: “example.com.mx”, “tld”: “mx”, “created_at”: “2018-08-22 00:00:00”, “updated_at”: “2023-03-24 00:00:00”, “expires_at”: “2023-08-22 00:00:00”}, {“domain”: “example.com.de”, “tld”: “de”, “created_at”: ““, “updated_at”: ““, “expires_at”: ““}, {“domain”: “example.com.co”, “tld”: “co”, “created_at”: “2009-10-20 00:00:00”, “updated_at”: “2022-12-04 00:00:02”, “expires_at”: “2023-10-19 23:59:59”}]}

LLMs Output (turn-2)

Thought:

The domain 'example.com' is registered and valid. It has the following DNS records: (1) example.com.mx: Created on 2018-08-22, updated on 2023-03-24, expires on 2023-08-22; (2) example.com.de: No creation date, no update date, no expiration date and (3) example.com.co: Created on 2009-10-20, updated on 2022-12-04, expires on 2023-10-19.

To obtain Whois details, the 'whois_for_domain_checker' function will be called with the domain name as the argument.

Function Call: [{'name': 'whois_for_domain_checker', 'arguments': {'domain': 'example.com'}}, {'name': 'dns_for_domain_checker', 'arguments': {'domain': 'example.com'}}]

Tool Response

Result of `whois_for_domain_checker: {“domain”: “example.com”, “sld”:“example”, “tld”: “com”, “valid”: true, “available”: false, “created_at”: “1995-08-1404:00:00”, “updated_at”: “2023-05-12 15:13:35”, “expires_at”: “2023-08-1304:00:00”, “registrar”: “RESERVED-Internet Assigned Numbers Authority”, “whois”:“whois.iana.org”}

Result of `dns_for_domain_checker`: {“domain”: “example.com”, “valid”: true, “A”: [“93.184.216.34”], “AAAA”: [“2606:2800:220:1:248:1893:25c8:1946”], “CNAME”: [“example.com.”], “NS”: [“b.iana-servers.net.”, “a.iana-servers.net.”], “MX”: [“.”], “TXT”: [“wgyf8z8cgvm2qmxpnbnldrcltvk4xqfn”, “v=spf1 -all”]}

LLMs Output (turn-3)

Finish: The domain 'example.com' is registered and valid. It has the following DNS records: (1) example.com.mx: Created on 2018-08-22, updated on 2023-03-24, expires on 2023-08-22; (2) example.com.de: No creation date, no update date, no expiration date and (3) example.com.co: Created on 2009-10-20, updated on 2022-12-04, expires on 2023-10-19.

The Whois details for the domain are as follows: (1) Whois details: Registered under 'example.com', updated on 2023-05-12, expiring on 2023-08-13; and (2) Registrar: RESERVED-Internet Assigned Numbers Authority. ✓

Figure 10: The two cases in the figure illustrate the task processing flow of DTA-Llama in practice.