

ASYNCTool: EVALUATING THE ASYNCHRONOUS FUNCTION CALLING CAPABILITY UNDER MULTI-TASK SCENARIOS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) based agents have demonstrated strong proficiency in leveraging external tools to address complex problems. However, existing evaluations largely overlook the temporal dimension of tool invocation, particularly the practical impact of inherent tool response latency, and they are typically confined to single-task scenarios. In realistic applications, tasks often need to be executed in parallel, and overall efficiency critically depends on the ability to utilize idle time during tool response delays. We denote this capability as asynchronous tool calling. To address the lack of evaluation in this area, we propose ASYNCTool, which, to the best of our knowledge, is the first benchmark specifically aimed at assessing the asynchronous multitasking abilities of LLM-based agents within interactive tool-use contexts. ASYNCTool consists of composite tasks with intra-task step dependencies that must be executed concurrently while incorporating realistic tool response delays. Through a hybrid data evolution strategy, we construct a diverse and representative asynchronous multitasking dataset that covers multiple scenarios and exhibits a wide range of tool use patterns. We further assess performance from three levels, namely Step Level, Sub-Task Level, and Task Level, covering perspectives from fine-grained to coarse-grained. Extensive experiments on ASYNCTool show that even state of the art models experience notable performance degradation when confronted with complex asynchronous workflows. Our analysis identifies the main failure modes of current tool agents and provides practical guidelines for designing future systems with stronger temporal reasoning and coordination capabilities.

1 INTRODUCTION

With recent advancements in large language models (LLMs) in their ability to follow instructions and understand contextual information, LLM-based agents have become increasingly proficient in tool use (OpenAI, 2025b; Comanici et al., 2025; Anthropic, 2025; Yang et al., 2025; Kimi et al., 2025; Zeng et al., 2025; Chen et al., 2025; Wang et al., 2025; Chen et al., 2024b). This capability enables them to handle more sophisticated, multi-step tasks and to achieve strong performance across diverse tool-use scenarios (Liu et al., 2023).

However, real-world environments are often more complex, frequently requiring the concurrent execution of multiple tasks that may involve different tools. In practical settings, function calls usually incur latency, and executing tasks sequentially in a synchronous manner fails to fully utilize idle time, thereby reducing overall efficiency. To better evaluate and enhance the agent’s performance under such conditions, we introduce the concept of **Asynchronous Tool Call** into the interaction between the agent and the environment, where the agent should utilize these idle intervals to advance other tasks. Motivated by these gaps, we identify three critical observations: **(i) Inadequate evaluation of the agent’s capability to complete multiple tasks in asynchronous scenarios.** Existing studies are typically restricted to single-task scenarios in which tools operate in an immediate response manner (Zhuang et al., 2023; Ruan et al., 2023; Xu et al., 2023; Guo et al., 2024; Qin et al., 2023; Ye et al., 2024), overlooking the evaluation for multiple tasks in asynchronous scenarios. **(ii) Lack of alignment with real-world conditions in interactive environments involving real-time tool calls.** Existing asynchronous planning benchmarks do not operate within interactive environments, which is inconsistent with real-world scenarios involving real-time tool calls (Lin et al., 2024). **(iii)**

Table 1: Comparison between ASYNCTOOL and existing benchmarks.

Benchmark	Async Executor	Function Call	Multi Task	Multi Step	Cross Scenario
τ -bench (Yao et al., 2024)	✗	✓	✗	✗	✗
BFCL v3 (Yan et al., 2024)	✗	✓	✗	✓	✗
NestFul (Basu et al., 2024)	✗	✓	✗	✓	✗
TimeArena (Zhang et al., 2024a)	✓	✗	✓	✓	✗
C ³ -Bench (Yu et al., 2025a)	✗	✓	✓	✓	✗
Robotouille (Gonzalez-Pumariega et al., 2025)	✓	✗	✗	✓	✗
ASYNCTOOL (Ours)	✓	✓	✓	✓	✓

Insufficient metrics and standardized protocols specific to concurrent tasks with delayed and out-of-order tool feedback. Traditional benchmarks involving time delays do not cover tool-using tasks and cannot be transferred to agentic tasks (Zhang et al., 2024a; Gonzalez-Pumariega et al., 2025).

Building upon these insights, we propose ASYNCTOOL, the first benchmark specifically designed to evaluate the ability of LLM-based agents to perform asynchronous tool calling in multi-task scenarios. (i) Our benchmark comprises combinations of multiple tasks, each containing intra-task step dependencies, while the tasks themselves run in parallel. This design allows a model to advance other tasks concurrently during the waiting periods for tool responses. (ii) To more accurately reflect real-world conditions, we assign realistic response times to each tool, integrate multiple tasks, and require the agent to progress them concurrently through asynchronous function calls. This framework provides a practical and realistic environment for assessing asynchronous task execution capabilities. Table 1 presents a comparison between ASYNCTOOL and existing benchmarks on tool calling and asynchronous execution. (iii) To better evaluate the asynchronous capabilities of models, we assess performance from three levels: Step Level, Sub-Task Level, and Task Level, thereby enabling our benchmark to provide a comprehensive evaluation ranging from fine-grained to coarse-grained perspectives. Through extensive experiments, we observe that while some models demonstrate effective asynchronous behavior, most LLMs still encounter considerable challenges in handling asynchronous function-call tasks. Our detailed analysis of the experimental results offers further insights into the observed failure modes, providing valuable guidance for future research.

The main contributions of our work are summarized as follows:

- We propose ASYNCTOOL, the first framework to evaluate an agent’s ability to perform multiple tasks through asynchronous tool calling, by explicitly simulating real-world tool response delays.
- We construct a high-quality asynchronous multitasking dataset, derived from single-task data using a hybrid strategy for data evolution. These composite tasks vary in quantity and encompass various scenarios, with each task comprising multi-step tool-call trajectories and clear function dependencies. This design guarantees complexity and challenge while maintaining the validity of the tasks.
- Through extensive experiments, we provide a detailed characterization of the challenges faced by tool-calling agents in asynchronous multitasking, examine the key difficulty factors affecting their performance, and conduct quantitative analyses that offer actionable insights for future research.

2 ASYNCTOOL

Building upon the motivation introduced in Section 1, ASYNCTOOL is designed to realistically simulate tool response latency and evaluate an agent’s asynchronous tool calling capabilities. This section first formalizes the interaction paradigm used in ASYNCTOOL, which models how agents manage multiple tasks concurrently under delayed tool feedback. We then describe the construction of the high-quality asynchronous multitasking dataset and present the evaluation protocol.

108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161

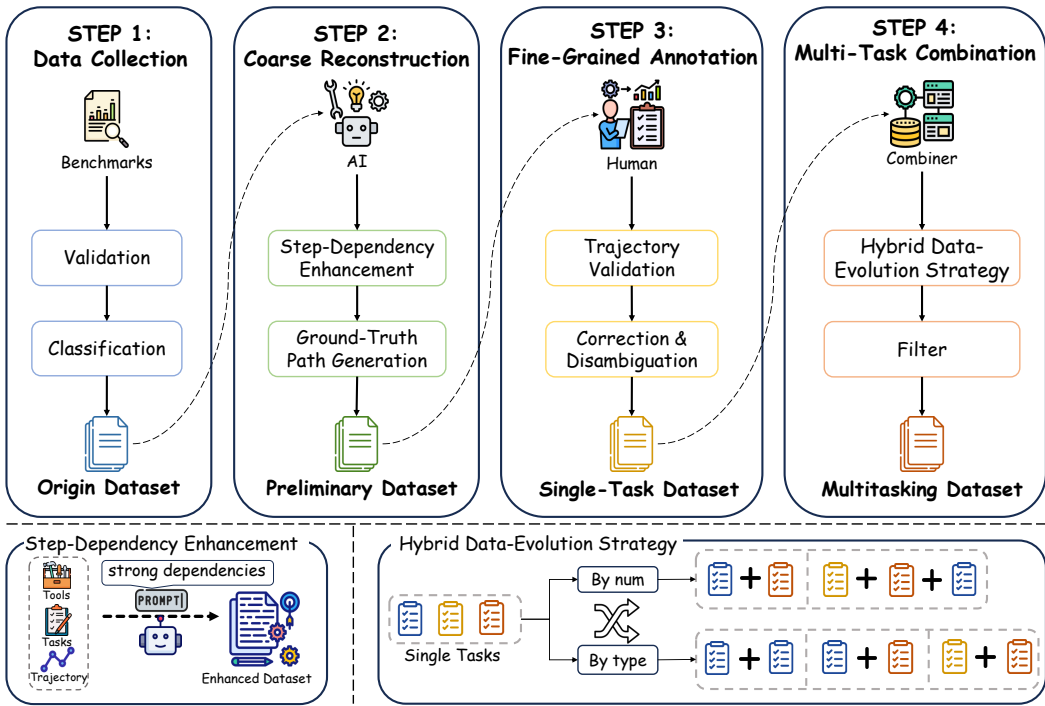


Figure 1: **Overview of the dataset construction process.** The pipeline starts by collecting raw data from tool-use benchmarks, categorizing it by scenario. Task-step dependencies are reinforced, and execution trajectories are reconstructed using Gemini 2.5 Pro, followed by manual verification for accuracy and determinism. Finally, data evolution occurs through hybrid strategies, with filtering producing the final multitasking dataset.

2.1 AGENT AS CONCURRENT TOOL-USING SYSTEM

While agents’ ability to solve problems through tool use is well established, their practical effectiveness is often hindered by the non-negligible response latencies of tool calls in real-world scenarios. This raises the research question of whether agents can exploit such idle periods to make progress on other tasks. ASYNCTOOL is designed to address this challenge by simulating real-world scenarios to evaluate an agent’s ability to asynchronously utilize tools and complete tasks. The framework explicitly models two key characteristics of real-world tool use: the tool response latency and the concurrent submission of multiple tasks. In ASYNCTOOL, the interaction pattern between the agent and tools differs fundamentally from that in conventional tool use settings: the results of all tool calls are not returned immediately but are intentionally delayed by a predefined waiting period. This design compels the model to advance other tasks during the idle time created by tool response latency. Consequently, delayed feedback and multitasking are essential components of ASYNCTOOL, distinguishing it from existing tool-use benchmarks.

For example, consider a scenario in which the agent receives two tasks, denoted as $task_1$ and $task_2$, requiring the function calls $\langle f_1, f_3, f_5 \rangle$ and $\langle f_2, f_4 \rangle$, respectively. The two tasks are independent of each other, while the function calls within each list must be executed sequentially due to dependency constraints. In this setting, the agent acts as the *Assistant*, while the execution system functions as the *Environment*. The *Assistant* first attempts to solve $task_1$ by calling f_1 . After producing the formatted output for this call, the *Environment* immediately reports that f_1 has not yet returned a result, as tool executions are inherently non-instantaneous. At this point, the *Assistant* switches to $task_2$ and issues the call f_2 , which similarly incurs its own waiting time. By the time the *Assistant* completes formatting the call for f_2 , the *Environment* provides the return value for f_1 , enabling the *Assistant* to resume execution of $task_1$. This alternating process continues until all tasks are completed.

The above example can be concretely realized within our framework, as illustrated in Figure 2. Conceptually, the framework models the large language model as a **concurrent system**, allowing it

162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215

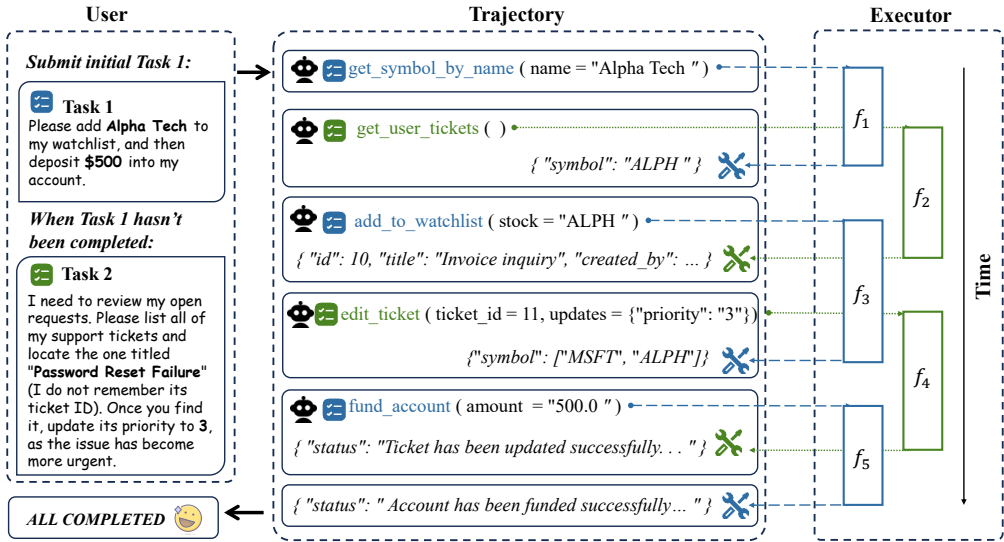


Figure 2: An example of multi-task completion through asynchronous function calls. In this scenario, the model receives tasks concurrently, and the executor simulates real-world latency by delaying outputs, which forces the model to interleave calls across tasks.

to interact with tools asynchronously and execute multiple tasks in parallel. The proposed benchmark subsequently evaluates the model’s capability to schedule and coordinate these tasks effectively.

2.2 DATA CONSTRUCTION

The construction of ASYNCTOOL requires a high-quality multitasking dataset. To this end, we propose a data construction pipeline consisting of four main steps: **Data Collection** (§ 2.2.1), **Coarse Reconstruction** (§ 2.2.2), **Fine-Grained Annotation** (§ 2.2.3), and **Multi-Task Composition** (§ 2.2.4). The overview of the dataset construction process is illustrated in Figure 1.

2.2.1 DATA COLLECTION

We note that existing benchmarks have already compiled tool APIs derived from real-world scenarios and provide well-developed tool executors, task descriptions, and execution paths. To avoid reinventing the wheel, we leverage these resources as suitable sources of high-quality single-task data. In our work, we select two representative benchmarks, NESTFUL (Basu et al., 2024) and BFCLv3 (Yan et al., 2024). After programmatic verification, we categorize and organize their tools and tasks, resulting in each task being uniquely associated with a specific tool category. Through this process, we extract a total of 12 tools and 358 tasks with corresponding tool-call paths as **Origin Dataset**.

2.2.2 COARSE RECONSTRUCTION

To ensure both consistency and accuracy of evaluation, we expect the data to follow a fixed tool-call trajectory for each task, which eliminates uncertainties in the calling sequence (Figure 4). Following this principle, we reinforce the dependencies between task steps by refining the task descriptions. Furthermore, to reduce the cost of manual annotation, we leverage Gemini 2.5 Pro (Comanici et al., 2025) to reconstruct the original task descriptions. To reduce the cost of manual annotation, we leverage Gemini 2.5 Pro (Comanici et al., 2025) to reconstruct the original task descriptions. Specifically, we provide the model with the original task description, multi-step execution trajectory, and tool set (Hsieh et al., 2023) as essential inputs, and sample instances in batches. Using carefully designed few-shot prompts, the model is guided to reconstruct the task descriptions and to generate strictly ordered function-call trajectories that are consistent with them. The specific prompt can be found in the Appendix C.5. Owing to the strong comprehension and reasoning capabilities of Gemini 2.5 Pro, the model produces reconstructed data that largely satisfies our requirements. Nonetheless, a

small number of issues remain, such as incorrect function arguments or mismatches between the task description and execution order, which require further manual annotation.

2.2.3 FINE-GRAINED ANNOTATION

After refining the data with Gemini 2.5 Pro, we obtain a **Preliminary Dataset** that still contains several potential issues, some of which are severe enough to invalidate the entire tool-call trajectory. To address these issues, we design a fine-grained human annotation pipeline aimed at eliminating data errors introduced during model-based generation.

Trajectory Validation. We first ensure that every function call in a trajectory is valid. To this end, we manually verify the sequential execution results for each task trajectory. Through this process, we identified three recurring error patterns and applied targeted corrections: (1) misinterpretation of the initial task conditions, leading to errors in the first call, such as repeatedly executing `cd` to enter the current directory in file system tasks; (2) violation of dependency relations, i.e., failing to invoke prerequisite functions, for example skipping a preceding call that is closely tied to the current one; and (3) misunderstanding of tool functionalities, often manifested as filling arguments in unsupported formats.

Correction and Disambiguation. Once the validation step confirms that all trajectories are free of execution errors, our focus shifts to aligning tasks with their corresponding trajectories and eliminating ambiguities. First, we verify the consistency between each task and its trajectory, removing task descriptions or partial trajectories that cannot be matched. Second, we strictly enforce the order of function calls within each trajectory, correcting any incorrect sequences. Finally, we replace ambiguous descriptions with precise expressions wherever possible, ensuring that essential details (e.g., location, time, and other key arguments) are explicitly included in the task description.

Following this pipeline, we conduct multiple rounds of verification on 358 tasks until no errors remain, ultimately producing a high-quality **Single-Task Dataset** comprising 358 validated instances, as summarized in Table 4.

2.2.4 MULTI-TASK COMPOSITION

In real-world settings, multitasking often spans multiple scenarios and involves complex tool combinations. To more realistically evaluate an agent’s capability for asynchronous function calls in multi-task scenarios, we focus on two factors: task quantity and task type. We introduce a hybrid data-evolution strategy, enabling the creation of diverse multitask combinations. Specifically, for task quantity, we define two categories: dual-task and tri-task. For task type, we define two categories: within-class and cross-class. By combining these factors, we obtain four distinct multitask configurations, which are applied to our single-task dataset to generate a rich multitask dataset.

However, applying the hybrid strategy results in an excessively large number of task combinations. To manage this, we adopt a weighted random sampling method to select a fixed-size subset of multitask data. The final **Multitasking Dataset** comprises 712 instances, effectively capturing the diversity and complexity of real-world multitasking scenarios.

2.3 EVALUATION

In **ASYNCTOOL**, each task is defined as a set of n subtasks $\{S_1, S_2, \dots, S_n\}$. Each subtask S_i is represented as a tuple (I, Q, T, E) , where I is a unique identifier, Q denotes the task query, T specifies the list of available APIs, and E denotes the hidden state corresponding to the subtask, which is not directly exposed to the assistant. The model’s response must explicitly include I to indicate which subtask is being executed. For each subtask, we extract its execution trajectory \mathcal{T}_i , defined as an ordered sequence of tool calls: $\mathcal{T}_i = \langle a_1, a_2, \dots, a_k \rangle$, where each action a_j is represented as a tuple $(tool, args)$. Once all subtasks are completed, we obtain the set $\{\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_n\}$, which is then used to evaluate whether the model has successfully accomplished the overall task.

In asynchronous multi-task execution, the interactions between the assistant and external tools can become highly complex. To provide a more comprehensive evaluation of the assistant’s performance under such conditions, we assess the results along three dimensions: Step Level, Sub-Task Level, and Task Level.

Table 2: **Main Results of ASYNCTOOL.** Func. and Param. mean matching the model’s results with the ground truth to calculate the F1 score. Char. means path matching while Env. means matching multiple execution result environments. **Bold** indicates best overall performance, while underline denotes the best within the same group.

Models	Step Level		Sub-Task Level			Task Level		Overall
	Func.	Param.	Char.	Env.	Acc.	Char.	Env.	
<i>Closed-Sourced Models</i>								
Qwen-Max	86.22	73.62	53.45	72.98	52.44	26.97	50.14	25.56
Kimi-K2	96.14	80.46	59.61	76.06	56.79	28.23	51.69	24.44
Gemini 2.5 Pro	89.08	78.27	63.16	76.43	62.05	34.13	54.35	32.44
GPT-5	92.21	80.11	62.13	78.34	60.67	33.15	58.43	31.32
GPT-4o	93.92	82.26	62.90	80.04	61.41	33.71	60.53	31.74
GPT-4.1	96.22	84.08	67.89	82.91	67.14	39.04	64.89	38.06
<i>Open-Sourced Models < 20B</i>								
LLaMA-3.1-8B-Ins.	78.29	43.69	12.95	41.08	12.47	1.26	14.61	1.26
GLM-4-9B-chat	74.72	59.70	29.67	55.94	28.93	7.30	25.56	6.88
Qwen2.5-7B-Ins.	82.40	65.01	28.66	55.15	26.38	6.74	25.84	6.04
Qwen2.5-14B-Ins.	81.32	70.22	50.21	65.29	46.28	<u>22.61</u>	38.20	<u>18.82</u>
Qwen3-8B	63.05	53.61	29.88	53.08	29.30	11.10	28.65	10.67
Qwen3-14B	<u>85.02</u>	<u>72.67</u>	49.62	<u>69.27</u>	<u>47.19</u>	20.51	<u>44.66</u>	<u>18.82</u>
<i>Open-Sourced Models > 20B</i>								
LLaMA-3.1-70B-Ins.	89.60	47.10	18.47	43.26	17.83	3.37	16.43	2.81
LLaMA-3.3-70B-Ins.	73.00	40.32	21.92	44.37	20.54	6.18	18.26	5.34
GLM-4-32B	60.59	51.41	35.46	56.21	33.97	16.57	29.78	15.17
Qwen3-32B	79.95	70.37	48.14	67.46	46.71	20.22	41.43	19.10
Qwen2.5-32B-Ins.	94.24	81.73	61.41	<u>74.42</u>	56.48	30.62	49.72	24.86
Qwen3-30B-A3B-Ins.	<u>94.29</u>	80.03	54.14	<u>72.72</u>	53.03	22.33	47.47	21.49
DeepSeek-V3.1-Terminus	86.10	75.32	59.39	72.45	56.21	<u>33.43</u>	49.30	<u>28.93</u>

Step Level. Following the fine-grained evaluation methodology of Patil et al. (2023), we assess the agent’s fundamental tool-calling capability, focusing on aspects such as call format, tool verification, and parameter verification. To quantify these results, we adopt the approach of Basu et al. (2024) and compute the F1 score separately for tool accuracy and parameter accuracy.

Sub-task Level. At this level, we define accuracy-based metrics to evaluate the agent’s performance on individual subtasks. For each subtask, we compare the assistant’s predicted trajectory $\mathcal{T}_i^{\text{pred}}$ with the ground-truth trajectory $\mathcal{T}_i^{\text{gt}}$ to determine whether the subtask is successfully completed, yielding the **trajectory-completion metric**. In addition, we assess the subtask’s hidden state by comparing E_i^{pred} and E_i^{gt} to measure consistency, yielding the **environment-matching metric**. Finally, these two metrics are combined into an **overall subtask accuracy**, which evaluates the agent’s final performance on subtask completion. Detailed calculation procedures are provided in Appendix A.1.

Task Level. As the final evaluation, the trajectory-completion and environment-consistency metrics at this level are counted only when all corresponding subtask-level metrics within a task are satisfied. These measurements provide an overall assessment of the agent’s ability to complete the entire task. The final overall task accuracy is defined as the proportion of tasks for which both conditions are simultaneously met.

3 EXPERIMENT

3.1 EXPERIMENTAL SETUP

We evaluate 19 models on ASYNCTOOL, aiming to provide a comprehensive benchmark for assessing their capability of asynchronous tool calling under multi-task scenarios. Specifically, for closed-source models, we select four prominent models: Qwen-max (Team, 2024c) created by the Qwen Team,

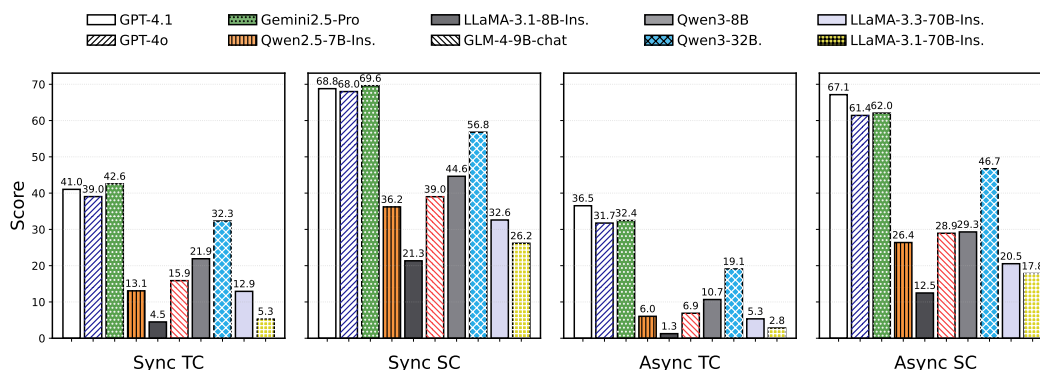


Figure 3: Scores of some models. TC is overall, and SC is Sub-Task according. As the model size reduces, the model score correspondingly declines. A higher subtask completion rate does not invariably result in a higher overall score.

Kimi k2 (Kimi et al., 2025) by Kimi Team, Gemini 2.5 Pro (Comanici et al., 2025) developed by Google, alongside GPT-4.1 (Achiam et al., 2023), GPT-4o (Hurst et al., 2024), and GPT-5 (OpenAI, 2025a) by OpenAI. For open-source LLMs, we evaluate numerous models including LLaMA3.1 (AI@Meta, 2024), LLaMA3.3, Qwen2.5 (Team, 2024a;b), Qwen3 (Yang et al., 2025), GLM4 (GLM et al., 2024), DeepSeek (Liu et al., 2024).

3.2 RESULTS ON ASYNCTOOL

We conducted a comprehensive evaluation of the current mainstream models. Based on these findings, our analysis is structured around three key questions.

Q1: Which Model is Better in Completing Multiple Tasks Asynchronously?

As shown in Table 10, GPT-4.1 demonstrates the strongest performance in asynchronous capability under asynchronous multitasking evaluation, achieving a score of 38.06. Close behind, the large open-source model DeepSeek-V3.1-Terminus achieves performance comparable to closed-source models, highlighting its competitive strength.

In the step-level evaluation, closed-source models consistently achieve high scores, while open-source models exhibit notable discrepancies. This highlights the differences in the asynchronous capabilities of these models. In the sub-task evaluation, the models’ scores are nearly double those of the overload models.

Q2: What Challenges do LLMs Encounter in ASYNCTOOL?

In the ASYNCTOOL evaluation, we carefully examine the models’ tool-call trajectories and conduct a detailed analysis of different numbers and classes of tasks. Models with lower performance often exhibit a lack of temporal reasoning ability, executing the next function call immediately without waiting for the tool’s response to the current task. This behavior is particularly problematic when there is a strong dependency between the two calls, frequently resulting in unforeseen errors. In contrast, higher-performing models can interleave the execution of different tasks, enabling them to advance multiple tasks in parallel successfully and thereby creating a substantial performance gap.

We also observe that agents occasionally fail to complete certain tasks. Specifically, some models tend to execute the most recently presented task first, neglecting earlier tasks. This issue is more common in smaller models and is almost absent in larger 70B-scale models. Moreover, it occurs more frequently in tri-task combinations than in dual-task combinations, which is consistent with intuition. Another notable error is tool misidentification, in which models misjudge the tool use across tasks, for example, invoking a flight-booking tool when they should continue a data-processing task. This type of error typically leads to cascading failures, as the model often struggles to self-correct once the confusion occurs. Common errors, by contrast, include non-compliant instructions, erroneous tool-call trajectories, and parameter errors, all of which directly prevent tasks from being scored successfully. Such errors still constitute a significant proportion of failures in smaller models within ASYNCTOOL, but their frequency decreases substantially as model size increases.

378
379
380
381
382
383
384
385
386
387
388
389
390

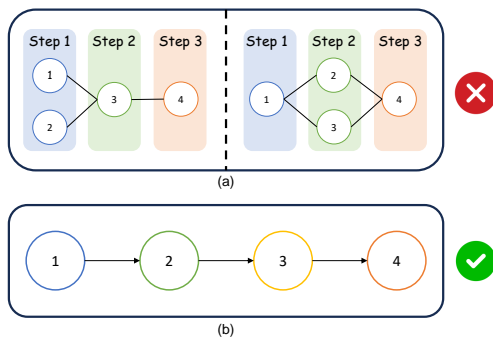


Figure 4: (a) shows the Non-deterministic trajectory. (b) shows the Deterministic trajectory.

391
392
393
394
395

Q3: What Factors Make Multi-Tasking Hard in Asynchronous Tool Calls?

396
397
398
399
400
401

As shown in Figure 3, the results show that asynchronous multitasking causes varying degrees of performance degradation across different models compared with regular multitasking, accompanied by a simultaneous decline in the SC index. In terms of the SC metric, Gemini 2.5 Pro exhibits the largest drop among closed-source models, while Qwen3-8B experiences the greatest decrease among open-source models.

402
403
404
405
406
407
408
409
410
411

The fundamental difference between asynchronous and regular multitasking lies in whether the agent can obtain tool responses immediately. In our evaluation, each function call in asynchronous multitasking incurs a one-round response delay, meaning that after invoking a function, the agent cannot immediately acquire the necessary information to proceed with the subsequent step. Such information may include confirmation signals from file operations, query results, or essential state data. Generally, higher-performing models can effectively shift their attention to other tasks by invoking different tools to make progress while waiting. By contrast, lower-performing models often lack temporal awareness and inappropriately proceed to the next step of the current task. In doing so, they sometimes fabricate parameters for the current call based on imagined results from the previous call—a typical hallucination phenomenon observed in large language models.

412
413
414
415
416
417
418
419

Another factor influencing difficulty is **task quantity**. An intuitive hypothesis is that as the number of tasks increases, the overall difficulty also increases. When the number of tasks reaches four, the difficulty of each task increases significantly, and the impact of task quantity becomes more pronounced. Consequently, we do not consider a larger number of tasks in this analysis. Based on ASYNCTOOL, we find that when the number of tasks goes from two to three, the closed-source model Gemini 2.5 Pro decreases the most, from 42.56 to 32.44 while the open-source model Qwen3-8B decreases the most, from 27.91 to 10.67 while the closed-source model GPT-4.1 and the open-source model LLaMA-3.1-70B-Ins decrease less. Full results are listed in Appendix C.1

420
421
422

3.3 DISCUSSION

423
424
425
426
427
428

In comparing ASYNCTOOL with an established tool-use benchmark, we observe a broadly similar trend in model performance as shown in Figure 5. Models that perform better in traditional tool-use settings also tend to achieve higher scores in asynchronous multi-task scenarios. This correlation may be explained by the fact that both types of tasks require similar underlying competencies, particularly the ability to determine intra-task step dependencies and retain parameter information across interactions, alongside robust context management and coherent integration of outputs.

429
430
431

Unlike conventional single-task, immediate-response tool-use benchmarks, ASYNCTOOL evaluates agents from a complementary perspective. It not only measures how well a model can invoke tools, but also assesses its capacity to maintain intermediate parameters in memory while resolving intra-task step dependencies and managing inter-task parallelism. This design imposes additional

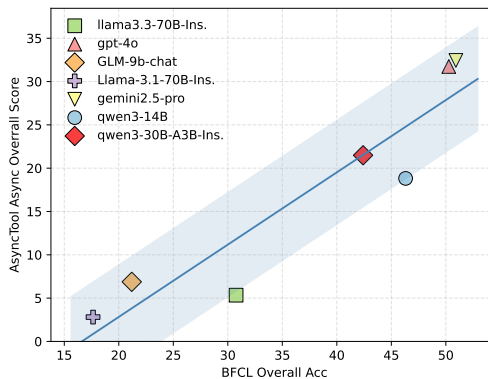


Figure 5: Comparison between BFCL Overall Accuracy and ASYNCTOOL Overall Scores across several models.

reasoning demands compared with traditional evaluations, as the model must track and coordinate multiple, partially interdependent processes under realistic latency constraints.

These differences make ASYNCTOOL a valuable supplement to existing benchmarks. It provides insight into dimensions of tool-augmented reasoning that are often overlooked in standard settings, such as memory retention under long-horizon tasks and the ability to plan actions based on dependency structures. We hypothesize that architectural or training strategies aimed at improving asynchronous multitasking, for example, by enhancing execution planning mechanisms or introducing latency-aware scheduling policies, might not only enhance performance in ASYNCTOOL but could also yield measurable improvements in conventional tool-use benchmarks. If this conjecture holds, it would underscore the strategic importance of emphasizing asynchronous mechanisms in the design of next-generation AI agents.

4 RELATED WORK

Multi-Step Tool Call. As LLM-based agents continue to improve in their ability to leverage external tools, related benchmarks have higher complexity and richer dependencies. Some tasks now require multiple tool calls, executed in the correct order, to be completed, which presents significant challenges for the models in terms of comprehension and planning. Currently, two primary approaches are used to evaluate this capability. The first approach asks the model to provide, in a single step, the full sequence of tool calls needed to solve the task, focusing on the model’s ability to plan tool usage (Yu et al., 2025b; Huang et al., 2024; Zhang et al., 2024b; Chen et al., 2024a; Wu et al., 2024). The second approach allows the model to invoke tools step by step, receiving responses at each stage. It emphasizes evaluating the multi-step tool calling process until task completion (Yan et al., 2024; Basu et al., 2024; Zhong et al., 2025).

The order of tool calls is often a critical aspect to evaluate. (Yan et al., 2024) constrains the tool call sequence using traditional rule-based restrictions, while NESTFUL (Basu et al., 2024) enforces a fixed call order through parameter dependencies. A key limitation of existing benchmarks is their failure to account for tool invocation delays frequently encountered in real-world applications. ASYNCTOOL addresses this gap, enabling systematic evaluation of a model’s capacity for asynchronous tool calling.

Asynchronous Multitasking Scenarios. The ability to perform multiple tasks asynchronously is an inevitable trend in the development of large LLMs. (Chu et al., 2023; Tan et al., 2023; Wang & Zhao, 2023; Ginart et al., 2024) introduce time changes in model evaluation and provide related evaluation tasks. (Zhang et al., 2024a) introduces the concept of time in multitasking, aiming to assess the efficiency with which models manage multiple tasks concurrently. (Gonzalez-Pumariega et al., 2025) evaluates models’ asynchronous planning, focusing on failure modes and challenges in integrating long-term information. While important for understanding LLMs in asynchronous tasks, these studies are limited to simulations and don’t assess performance on tasks requiring real-world tool calls. Moreover, (Gim et al., 2024) enhances the operational efficiency of LLMs by enabling them to generate and execute function calls concurrently, representing a novel approach. Unlike prior benchmarks, ASYNCTOOL is the first to systematically evaluate models’ capabilities for asynchronous tool calling in realistic multi-task scenarios, explicitly incorporating tool response latency into the assessment.

5 CONCLUSION

In this paper, we present ASYNCTOOL, the first benchmark to explicitly incorporate the notion of time into the evaluation of agent tool use. ASYNCTOOL captures not only the intra-task step dependencies within a task but also the associated tool response latency—an essential factor that drives agents to perform multiple tasks efficiently under an asynchronous framework in realistic scenarios. Leveraging this asynchronous tool calling paradigm, ASYNCTOOL systematically evaluates LLM-based agents in terms of their asynchronous multitasking capabilities and their ability to perform tool calling under realistic latency constraints. Experimental results show that as the number of tasks increases and their compositions become more complex, model performance degrades significantly, with many failing to complete the tasks successfully. These findings suggest that tool-using agents still have considerable

486 room for improvement in handling multiple tasks asynchronously, providing valuable guidance for
487 the advancement of future agent systems.
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539

REPRODUCIBILITY STATEMENT

Our work is fully reproducible. We guarantee the reproducibility of all results and the reliability of the models used. All data and code will be released within one month after the article is accepted.

ETHICS STATEMENT

Our paper proposes a new LLM benchmark that contains frequently updated questions from new information sources, scores answers based on objective ground truth, and covers a wide variety of tasks. We believe our work does not have any inherent negative impact on society. We hope that our work will have a positive impact on practitioners and researchers: by providing a new benchmark with frequently updated questions, our work has the potential to both accelerate future research and enable more comprehensive and rigorous evaluations of existing and future models. Furthermore, we hope that the overall framework of our benchmark—containing frequently updated questions from new information sources—will become popular, thereby mitigating the negative impact of contamination in future LLM assessments and making LLM benchmarks more “future-proof.”

REFERENCES

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.
- Anthropic. Introducing claude 4, 2025. URL <https://www.anthropic.com/news/claude-4>.
- Kinjal Basu, Ibrahim Abdelaziz, Kelsey Bradford, Maxwell Crouse, Kiran Kate, Sadhana Kumaravel, Saurabh Goyal, Asim Munawar, Yara Rizk, Xin Wang, et al. Nestful: A benchmark for evaluating llms on nested sequences of api calls. *arXiv preprint arXiv:2409.03797*, 2024.
- Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.
- Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. T-eval: Evaluating the tool utilization capability of large language models step by step. In *ACL (1)*, pp. 9510–9529, 2024a. URL <https://aclanthology.org/2024.acl-long.515>.
- Zehui Chen, Kuikun Liu, Qiuchen Wang, Jiangning Liu, Wenwei Zhang, Kai Chen, and Feng Zhao. Mindsearch: Mimicking human minds elicits deep ai searcher. *arXiv preprint arXiv:2407.20183*, 2024b.
- Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Haotian Wang, Ming Liu, and Bing Qin. Timebench: A comprehensive evaluation of temporal reasoning abilities in large language models. *arXiv preprint arXiv:2311.17667*, 2023.
- Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.
- In Gim, Seung-seob Lee, and Lin Zhong. Asynchronous llm function calling. *arXiv preprint arXiv:2412.07017*, 2024.
- Antonio A Ginart, Naveen Kodali, Jason Lee, Caiming Xiong, Silvio Savarese, and John Emmons. Asynchronous tool usage for real-time agents. *arXiv preprint arXiv:2410.21620*, 2024.

- 594 Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas,
595 Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to
596 glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.
- 597
598 Gonzalo Gonzalez-Pumariaga, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille:
599 An asynchronous planning benchmark for llm agents. *arXiv preprint arXiv:2502.05227*, 2025.
- 600
601 Zishan Guo, Yufei Huang, and Deyi Xiong. CToolEval: A Chinese benchmark for LLM-powered
602 agent evaluation in real-world API interactions. pp. 15711–15724, 2024.
- 603
604 Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee,
605 Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large
606 language models. *arXiv preprint arXiv:2308.00675*, 2023.
- 607
608 Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan
609 Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. Planning, creation,
610 usage: Benchmarking LLMs for comprehensive tool utilization in real-world complex scenarios.
611 In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for
612 Computational Linguistics: ACL 2024*, pp. 4363–4400. Association for Computational Linguistics,
2024. doi: 10.18653/v1/2024.findings-acl.259. URL [https://aclanthology.org/2024.
findings-acl.259/](https://aclanthology.org/2024.findings-acl.259/).
- 613
614 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-
615 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint
arXiv:2410.21276*, 2024.
- 616
617 Team Kimi, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru
618 Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint
arXiv:2507.20534*, 2025.
- 620
621 Fangru Lin, Emanuele La Malfa, Valentin Hofmann, Elle Michelle Yang, Anthony G. Cohn, and
622 Janet B. Pierrehumbert. Graph-enhanced large language models in asynchronous plan reasoning.
623 In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*, 2024.
- 624
625 Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao,
626 Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint
arXiv:2412.19437*, 2024.
- 627
628 Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding,
629 Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint
arXiv:2308.03688*, 2023.
- 630
631 OpenAI. Gpt-5. <https://openai.com/zh-Hans-CN/index/introducing-gpt-5/>,
632 2025a. Accessed: 2025-08-07.
- 633
634 OpenAI. Introducing o3 and o4-mini, 2025b. URL [https://openai.com/index/
introducing-o3-and-o4-mini/](https://openai.com/index/introducing-o3-and-o4-mini/).
- 635
636 Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model
637 connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.
- 638
639 Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru
640 Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world
641 apis. *arXiv preprint arXiv:2307.16789*, 2023.
- 642
643 Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi,
644 Hangyu Mao, Xingyu Zeng, and Rui Zhao. Tptu: Task planning and tool usage of large language
model-based ai agents. *arXiv preprint arXiv:2308.03427*, 2023.
- 645
646 Qingyu Tan, Hwee Tou Ng, and Lidong Bing. Towards benchmarking and improving the temporal
647 reasoning capability of large language models. *arXiv preprint arXiv:2306.08952*, 2023.
- Qwen Team. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

- 648 Qwen Team. Qwen2.5: A party of foundation models, September 2024b. URL [https://qwenlm.](https://qwenlm.github.io/blog/qwen2.5/)
649 [github.io/blog/qwen2.5/](https://qwenlm.github.io/blog/qwen2.5/).
650
- 651 Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024c.
- 652 Qiuchen Wang, Ruixue Ding, Yu Zeng, Zehui Chen, Lin Chen, Shihang Wang, Pengjun Xie, Fei
653 Huang, and Feng Zhao. Vrag-rl: Empower vision-perception-based rag for visually rich in-
654 formation understanding via iterative reasoning with reinforcement learning. *arXiv preprint*
655 *arXiv:2505.22019*, 2025.
- 656 Yuqing Wang and Yun Zhao. Tram: Benchmarking temporal reasoning for large language models.
657 *arXiv preprint arXiv:2310.00835*, 2023.
658
- 659 Mingsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. Seal-tools:
660 Self-instruct tool learning dataset for agent tuning and detailed benchmark. In *Natural*
661 *Language Processing and Chinese Computing: 13th National CCF Conference, NLPCC 2024,*
662 *Hangzhou, China, November 1–3, 2024, Proceedings, Part II*, pp. 372–384, 2024. ISBN 978-
663 981-97-9433-1. doi: 10.1007/978-981-97-9434-8_29. URL [https://doi.org/10.1007/
664 978-981-97-9434-8_29](https://doi.org/10.1007/978-981-97-9434-8_29).
- 665 Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool
666 manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*,
667 2023.
- 668 Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and
669 Joseph E. Gonzalez. Berkeley function calling leaderboard. 2024.
670
- 671 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang
672 Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*,
673 2025.
- 674 Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A benchmark for
675 tool-agent-user interaction in real-world domains, 2024. URL [https://arxiv.org/abs/
676 2406.12045](https://arxiv.org/abs/2406.12045).
- 677 Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan
678 Dou, Qi Zhang, Tao Gui, et al. Tooleyes: Fine-grained evaluation for tool learning capabilities of
679 large language models in real-world scenarios. *arXiv preprint arXiv:2401.00741*, 2024.
680
- 681 Peijie Yu, Yifan Yang, Jinjian Li, Zelong Zhang, Haorui Wang, Xiao Feng, and Feng Zhang. \mathcal{E} 3-bench:
682 The things real disturbing llm based agent in multi-tasking. *arXiv preprint arXiv:2505.18746*,
683 2025a.
- 684 Peijie Yu, Yifan Yang, Jinjian Li, Zelong Zhang, Haorui Wang, Xiao Feng, and Feng Zhang. Multi-
685 mission tool bench: Assessing the robustness of llm based agents through related and dynamic
686 missions, 2025b. URL <https://arxiv.org/abs/2504.02623>.
687
- 688 Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang,
689 Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation
690 models. *arXiv preprint arXiv:2508.06471*, 2025.
- 691 Yikai Zhang, Siyu Yuan, Caiyu Hu, Kyle Richardson, Yanghua Xiao, and Jiangjie Chen. Timearena:
692 Shaping efficient multitasking language agents in a time-aware simulation. *arXiv preprint*
693 *arXiv:2402.05733*, 2024a.
694
- 695 Yinger Zhang, Hui Cai, Xierui Song, Yicheng Chen, Rui Sun, and Jing Zheng. Reverse chain: A
696 generic-rule for LLMs to master multi-API planning. In Kevin Duh, Helena Gomez, and Steven
697 Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 302–
698 325. Association for Computational Linguistics, 2024b. doi: 10.18653/v1/2024.findings-naacl.22.
699 URL <https://aclanthology.org/2024.findings-naacl.22/>.
- 700 Lucen Zhong, Zhengxiao Du, Xiaohan Zhang, Haiyi Hu, and Jie Tang. Complexfuncbench: ex-
701 ploring multi-step and constrained function calling under long-context scenario. *arXiv preprint*
arXiv:2501.10132, 2025.

702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755

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

A EVALUATION

A.1 METRICS

Character matching. We extract the function name and parameters and match them against the golden truth. We also consider the possibility that the model might execute some incorrect function calls due to the use of subset validation. Specifically, given a tool list T and a query q , the model generates a series

$$\mathcal{T}^{\text{pred}} = \langle a_1^{\text{pred}}, a_2^{\text{pred}}, \dots, a_n^{\text{pred}} \rangle,$$

where a_i^{pred} is the predicted LLM action at turn i . The golden truth is

$$\mathcal{T}^{\text{gt}} = \langle a_1^{\text{gt}}, a_2^{\text{gt}}, \dots, a_n^{\text{gt}} \rangle.$$

We verify whether $A^{\text{gt}} \subseteq A^{\text{pred}}$.

F1 function & Parameters. Referring to the implementation of NESTFUL (Basu et al. (2024)), we use the above metrics to evaluate the model’s responses. In our experiments, we find that frequent task switching by the model interferes with its ability to recognize different tasks accurately. Specifically, the model often invokes incorrect tools for the tasks, which leads to a decline in performance metrics.

Trajectory completion. After we extract the function, we will throw \mathcal{T}^{gt} and $\mathcal{T}^{\text{pred}}$ to the executor for execution. The executor will return the corresponding results R^{gt} and R^{pred} , and we will compare the results using the subset judgment method the same as above.

Environment matching. Referring to the implementation of BFCL (Yan et al. (2024)), we use the executor class instance for comparison. In our data, some tool calls induce alterations to the environment, while the majority of tool calls do not result in any environmental changes, such as those associated with queries. Nonetheless, we incorporate this metric into the evaluation to ensure comprehensiveness.

Table 3: Abbreviation explanation of data categories.

Full Name	Abbreviation
Data Management	DM
Filesystem	FS
Data generation	DG
MessageAPI	MA
Number operations	NO
SocialConnect	SC
String Manipulation	SM
TicketPurchase	TP
TradingBot	TB
TravelPlanning	TP*
DataFormat	DF
Machine Operation	MO

A.2 MODEL VERSION

The version for GPT-4o is gpt-4o-2024-11-20, for GPT-4.1 is gpt-4.1-2025-04-14, and for Gemini 2.5 Pro is gemini-2.5-pro-preview-05-06. The version for GLM-4-32B is GLM-4-32B-0414. The version for Qwen3-30B-A3B-Ins. is Qwen3-30B-A3B-Instruct-2507.

B DATA

B.1 DATA DETAILS

As shown in Table 3, the content inside introduces the meaning of our categories. Besides, Table 5 shows the data distribution of ASYNCTOOL.

Table 4: Task Counts and Average Trajectory Lengths.

Type	DM	FS	DG	MA	NO	SC	SM	TP	TB	TP*	DF	MO
#Tasks	59	53	29	23	24	18	34	21	15	20	44	18
Avg. len.	2.14	3.58	2.14	3.48	2.12	4.11	2.09	2.81	3.27	3.10	2.09	3.00

B.2 DATA LENGTH

We counted the input and output lengths of each task to ensure that they were within a reasonable range. These data are shown in Figure 6.

B.3 SUPPLEMENTARY DATA

For the four-task test, considering the large number of combination types and tasks, we randomly screened the data after each combination and finally selected 300 SIMILAR and 300 CROSS four-task data for comparison. The results are only used for ablation experiments.

Table 5: **Data composition of ASYNCTOOL.** # denotes number of, and % denotes proportion of. SIMILAR means similar task combinations and CROSS means cross task combinations.

Method	#Tasks	#Samples	%Samples
SIMILAR	2	120	16.85%
CROSS	2	132	18.54%
SIMILAR	3	240	33.71%
CROSS	3	220	30.90%

C RESULTS

In this section, we present the comprehensive results, categorized by different types, along with the test outcomes for varying numbers of tasks on the open-source model.

C.1 DETAILED RESULTS

As shown in Figure 6, we compile and present the detailed results by task type.

C.2 OTHER RESULTS

As shown in Figure 7, we conduct a preliminary experiment on the open-source model to demonstrate the impact of the number of tasks on the score.

C.3 ANALYSIS OF ERROR CASES

We provide numerous examples of common model errors in the end of appendix. A frequent issue, likely due to insufficient prior training on multiple tasks, is the model forgetting about ongoing tasks. This results in a high rate of incompleteness failures, despite the model’s high accuracy in completing subtasks. In open-source models, frequent task switching presents an even greater challenge to the model’s memory and localization abilities, causing confusion in function calls, which is a primary source of errors.

C.4 ANALYSIS OF MODEL PERFORMANCE

We must acknowledge that even the most advanced models perform suboptimally on our benchmark. However, we need to analyze the reasons behind this from two perspectives. First, we have conducted ablation experiments on mainstream large models, revealing that as the number of tasks increases, the

864 Table 6: **Detailed Results of ASYNCTOOL.** We approach the analysis from the perspective of
 865 dataset classification to calculate the results. We find that the scores across different categories exhibit
 866 significant disparities. **Bold** indicates best overall performance, while underline denotes the best
 867 within the same group.

Models	SIMILAR#2	SIMILAR#3	CROSS#2	CROSS#3	Overall
<i>Closed-Sourced Models</i>					
Qwen-max	32.50	15.42	41.67	23.18	25.56
Kimi-K2	33.33	20.83	30.30	20.00	24.44
Gemini 2.5 Pro	44.17	27.5	40.15	26.82	32.44
GPT-5	41.67	21.67	43.94	28.64	31.32
GPT-4o	42.50	29.17	41.67	22.73	31.74
GPT-4.1	50.00	36.67	47.73	27.27	38.06
<i>Open-Sourced Models < 20B</i>					
LLaMA-3.1-8B-Ins.	1.67	0.42	4.55	0	1.26
GLM-4-9B-chat	12.50	7.08	8.33	2.73	6.88
Qwen2.5-7B-Ins.	10.00	4.17	12.12	2.27	6.04
Qwen2.5-14B-Ins.	<u>30.00</u>	<u>17.92</u>	18.94	<u>13.64</u>	<u>18.82</u>
Qwen3-8B	25.83	5.42	15.91	5.00	10.67
Qwen3-14B	27.50	14.58	<u>28.03</u>	13.18	<u>18.82</u>
<i>Open-Sourced Models >20B</i>					
LLaMA-3.1-70B-Ins.	5.83	1.67	6.06	0.45	2.81
LLaMA-3.3-70B-Ins.	10.00	3.75	8.33	2.73	5.34
GLM-4-32B	21.67	13.75	21.97	9.09	15.17
Qwen3-32B	31.67	13.33	26.52	14.09	19.10
Qwen2.5-32B-Ins.	<u>34.17</u>	20.83	31.82	20.00	24.86
Qwen3-30B-A3B-Ins.	30.00	15.42	32.58	16.82	21.49
DeepSeek-V3.1-Terminus	32.50	<u>24.58</u>	<u>35.61</u>	<u>27.73</u>	<u>28.93</u>

891 Table 7: **Experimental results with different numbers of tasks on SYNC setting.**
 892 In #number,number means the number of tasks. # 3 is compared with # 2 and # 4 is compared with #
 893 3. The data source of # 4 is shown in Appendix B.1

Models	# 2			# 3				# 4			
	S#2	C#2	Overall	S#3	C#3	Overall	Drop	S#4	C#4	Overall	Drop
Qwen2.5-7B-Ins.	20.00	14.39	17.06	11.67	10.00	10.87	36.28%	8.51	3.44	6.33	41.77%
GLM-4-9b-chat	20.83	21.97	21.43	14.58	10.91	12.83	40.13%	12.73	3.65	8.17	36.32%
Qwen2.5-72B-Ins.	53.33	47.73	50.79	35.42	31.82	33.70	33.65%	20.33	23.67	22.00	34.72%
LLaMA3.3-70B-Ins.	25.00	16.67	20.63	10.42	6.82	8.70	57.83%	3.67	3.67	3.67	57.81%

902
 903 difficulty grows nonlinearly. To encourage future models to achieve greater capabilities, we designed
 904 the benchmark such that tasks involving three or more objectives account for over 60

905
 906 C.5 ABLATION RESULTS

907
 908 D USE OF LARGE LANGUAGE MODELS

909
 910 We use a large language model for translation and language polishing, but its role in the paper is
 911 limited. The core research and content are completed by us.

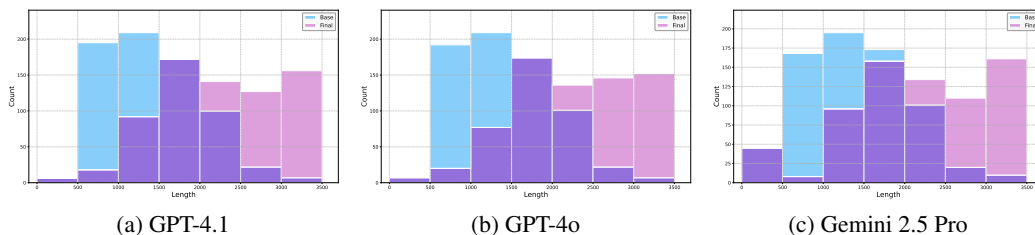
912
 913
 914
 915
 916
 917

Table 8: **Supplementary Results of ASYNCTOOL. All experiments are conducted with a delay of two turns. Bold indicates best overall performance, while underline denotes the best within the same group.**

Models	Step Level		Sub-Task Level			Task Level		Overall
	Func.	Param.	Char.	Env.	Acc.	Char.	Env.	
<i>Closed-Sourced Models</i>								
Qwen-Max	95.81	82.70	64.7	77.18	60.18	35.11	53.51	30.76
Gemini 2.5 Pro	91.94	80.05	57.48	75.90	56.48	26.97	53.65	25.70
GPT-4o	91.41	79.79	57.70	75.42	56.79	27.67	53.37	26.54
GPT-4.1	91.32	79.97	57.96	75.90	6.90	27.67	54.49	26.40
<i>Open-Sourced Models < 20B</i>								
LLaMA-3.1-8B-Ins.	83.16	44.42	13.38	40.76	12.42	1.54	14.75	1.40
Qwen2.5-7B-Ins.	<u>87.67</u>	66.97	32.27	58.28	29.99	6.88	26.40	5.48
Qwen2.5-14B-Ins.	86.75	<u>72.18</u>	<u>50.21</u>	<u>63.59</u>	<u>45.06</u>	<u>19.94</u>	<u>35.11</u>	<u>15.17</u>
Qwen3-8B	59.06	49.4	24.52	48.99	23.99	6.6	21.77	6.6
<i>Open-Sourced Models > 20B</i>								
LLaMA-3.1-70B-Ins.	90.73	45.45	17.04	40.02	16.14	1.83	12.36	1.4
Qwen3-32B	70.67	60.99	34.34	56.95	33.7	11.52	30.76	10.96
Qwen3-30B-A3B-Ins.	<u>95.57</u>	<u>79.72</u>	<u>55.41</u>	<u>71.71</u>	<u>52.81</u>	<u>23.17</u>	<u>44.66</u>	<u>20.65</u>
DeepSeek-V3.1-Terminus	81.69	70.52	43.84	65.02	42.94	15.73	36.94	14.61

Table 9: **Supplementary Results of ASYNCTOOL. All experiments are conducted with a delay of zero to one turn randomized. Bold indicates best overall performance, while underline denotes the best within the same group.**

Models	Step Level		Sub-Task Level			Task Level		Overall
	Func.	Param.	Char.	Env.	Acc.	Char.	Env.	
<i>Closed-Sourced Models</i>								
Qwen-Max	96.65	84.21	66.30	80.57	64.65	36.80	58.43	34.41
Gemini 2.5Pro	<u>95.57</u>	83.85	65.45	81.69	64.92	<u>35.81</u>	62.22	35.25
GPT-4o	96.00	84.03	66.08	81.74	65.61	35.96	62.08	35.53
GPT-4.1	95.45	83.58	65.76	81.90	65.18	35.67	62.92	34.55
<i>Open-Sourced Models < 20B</i>								
LLaMA-3.1-8B-Ins.	83.93	47.44	16.24	43.84	15.13	1.97	16.71	1.69
Qwen2.5-7B-Ins.	83.73	67.22	32.11	59.02	31.00	8.71	29.07	8.43
Qwen2.5-14B-Ins.	<u>85.00</u>	<u>73.32</u>	<u>48.30</u>	<u>67.46</u>	<u>46.18</u>	<u>19.24</u>	<u>39.47</u>	<u>17.98</u>
Qwen3-8B	69.38	60.22	38.80	61.09	38.32	14.47	35.11	14.04
<i>Open-Sourced Models > 20B</i>								
LLaMA-3.1-70B-Ins.	90.49	49.28	20.12	45.59	19.75	3.51	18.12	3.37
Qwen3-32B	76.21	67.91	47.45	66.88	47.03	20.79	43.54	20.08
Qwen3-30B-A3B-Ins.	<u>95.10</u>	<u>82.17</u>	<u>59.39</u>	<u>76.70</u>	<u>58.86</u>	26.69	<u>52.67</u>	<u>26.26</u>
DeepSeek-V3.1-Terminus	90.04	78.29	56.32	75.96	55.73	<u>26.97</u>	52.39	<u>26.26</u>



(a) GPT-4.1

(b) GPT-4o

(c) Gemini 2.5 Pro

Figure 6: Length distribution for base and final conversation, measured by the number of tokens.

Table 10: **Supplementary Results of ASYNCTOOL. All experiments are conducted with a delay of one to two turns randomized. Bold indicates best overall performance, while underline denotes the best within the same group.**

Models	Step Level		Sub-Task Level			Task Level		Overall
	Func.	Param.	Char.	Env.	Acc.	Char.	Env.	
<i>Closed-Sourced Models</i>								
Qwen-Max	96.87	83.56	64.49	78.50	61.94	33.85	54.63	30.20
Gemini 2.5 Pro	93.98	81.68	61.84	78.34	61.04	30.62	57.58	29.35
GPT-4o	93.08	81.05	60.51	78.29	59.66	30.06	57.44	28.79
GPT-4.1	91.97	80.29	58.76	76.17	58.07	30.34	55.90	29.49
<i>Open-Sourced Models < 20B</i>								
LLaMA-3.1-8B-Ins.	83.77	45.06	12.74	41.14	11.68	1.54	15.03	1.54
Qwen2.5-7B-Ins.	86.02	66.81	29.09	56.48	27.55	6.88	26.26	6.04
Qwen2.5-14B-Ins.	<u>87.22</u>	<u>73.30</u>	<u>49.10</u>	<u>65.82</u>	<u>45.70</u>	<u>19.52</u>	<u>36.94</u>	<u>16.29</u>
Qwen3-8B	60.72	50.76	25.85	51.38	25.42	7.30	25.14	7.16
<i>Open-Sourced Models > 20B</i>								
LLaMA-3.1-70B-Ins.	90.76	46.37	16.67	40.98	16.03	2.25	13.90	1.83
Qwen3-32B	71.41	61.92	37.10	59.55	36.20	12.08	33.71	11.24
Qwen3-30B-A3B-Ins.	<u>95.39</u>	<u>79.86</u>	<u>54.09</u>	<u>72.13</u>	<u>52.87</u>	<u>22.75</u>	<u>46.49</u>	<u>20.93</u>
DeepSeek-V3.1-Terminus	84.52	73.42	47.35	68.31	46.07	17.98	39.89	16.57

Table 11: **Results of ASYNCTOOL on fewshot settings. Specifically, we provided a successful trajectory in the prompt as a reference.**

Models	Step Level		Sub-Task Level			Task Level		Overall
	Func.	Param.	Char.	Env.	Acc.	Char.	Env.	
<i>Open-Sourced Models < 20B</i>								
Qwen2.5-7B-Ins.	82.40	65.01	28.66	55.15	26.38	6.74	25.84	6.04
+fewshot	78.89	61.72	30.04	54.41	28.13	8.99	26.40	8.29
LLaMA-3.1-8B-Ins.	78.29	43.69	12.95	41.08	12.47	1.26	14.61	1.26
+fewshot	90.27	63.35	29.67	56.37	28.03	7.16	27.67	6.74
Qwen3-8B	63.05	53.61	29.88	53.08	29.30	11.10	28.65	10.67
+fewshot	72.04	60.67	34.82	34.82	34.34	11.52	32.30	11.24
Qwen2.5-14B-Ins.	81.32	70.22	50.21	65.29	46.28	22.61	38.20	18.32
+fewshot	94.28	80.40	54.19	75.32	52.87	22.89	50.14	21.91
Qwen2.5-72B-Ins.	95.69	82.78	65.07	79.78	62.00	34.41	58.01	31.04
+fewshot	89.34	83.61	66.40	81.16	64.65	36.94	59.69	34.55

1026
 1027
 1028
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079

Table 12: **Supplementary Results of ASYNCTOOL. Specifically, we analyzed the average number of rounds of interaction in successfully completed task models.**

Models	# 2	# 3
<i>Closed-Sourced Models</i>		
Qwen-max	6.91	9.38
Gemini 2.5 Pro	6.85	10.50
GPT-4o	6.95	10.52
GPT-4.1	6.70	10.57
<i>Open-Sourced Models < 20B</i>		
LLaMA-3.1-8B-Ins.	9.75	9.00
Qwen2.5-7B-Ins.	8.50	12.35
Qwen2.5-14B-Ins.	7.94	10.95
Qwen3-8B	6.64	9.19
<i>Open-Sourced Models >20B</i>		
LLaMA-3.1-70B-Ins.	8.41	16.28
LLaMA-3.3-70B-Ins.	10.62	16.23
Qwen2.5-32B-Ins.	9.20	13.68

1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

Standardization of Test Data

System Prompt:
You are an expert in handling multiple tasks using functions. Your goal is to call one function in each round, gradually completing all tasks. When a function call is not yet complete, you can switch to another task. If you decide to invoke a function for a task, you **MUST** format it as follows:

```
{"id": "id of task", "func_name": "func_name", "params": {"params_name1": "params_value1", "params_name2": "params_value2"}}
```

You must invoke **only one** function in your response. If you believe all tasks have been completed, please use the following format to reply:

```
{"content": "ALL COMPLETED"}
```

Here is a list of functions in JSON format that you can invoke.

```
[API List]
```

User: Below are all tasks you need to finish:

```
{
  id:'file_13',
  question:'Compare 'academic_venture/goals.txt' with
'academic_venture/reference_goals.txt' to verify if they are identical, then
rename 'goals.txt' to 'master_goals.txt' in the same directory
tory.'
}
{
  id:'SM_11',
  question:have a list of integers [1, 2, 3, 4, 5]. I want to convert this list
into a string representing a space-separated sequence of those integers. Then, I
want to normalize this string by replacing all non-alphanumeric characters with
underscores and converting to lowercase.'
}
{
  id:'posting_11',
  question:'Now that I'm following 'AI_enthusiast', I'd like to know more about
them
. Please retrieve their user statistics. After that, fetch all of their tweets.
I remember seeing one about 'ethics'; find that specific tweet and add the
comment 'A very important point to consider.' to it.'
}
```

Ground True:

```
[{"task_id": "file_13",
  "ground_truth": [
    "cd(folder='academic_venture')",
    "diff(file_name1='goals.txt', file_name2='reference_goals.txt')",
    "mv(source='goals.txt', destination='master_goals.txt')"
  ]
},
{"task_id": "SM_11",
  "ground_truth": [
    "integer_list_to_string(integer_list=[1, 2, 3, 4, 5])",
    "normalize_string(s='1 2 3 4 5')"
  ]
},
{"task_id": "posting_11",
  "ground_truth": [
    "get_user_stats(username='AI_enthusiast')",
    "get_user_tweets(username='AI_enthusiast')",
    "comment(tweet_id=18, comment_content='A very important point to
consider.')"
  ]
}]
```

Figure 7: An example of standardized test data.

1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187

Prompt for Reconstruction

Prompt:
You are an AI assistant with API-calling capabilities. Your task is to refine query descriptions based on given contexts and initial configurations, then provide explicit solution paths. You must meet these requirements:

1. Ensure problems are solvable within the given initial configuration, avoiding ambiguous or unsolvable cases. Clarify any ambiguities in the original problem by deriving explicit specifications from the initial configuration. Including but not limited to: file names, folder paths, numerical counts, and similar parameters.
2. Define solution paths as strict API call sequences (not less than 2 steps) with strong dependencies—later steps must require outputs from earlier ones. Expand the problem if necessary to meet this length requirement.
3. Preserve existing explicit information (*e.g.*, filenames, data). Extend problems only by deriving new details from the initial config.
4. Always provide the most efficient path—no unnecessary API calls. Consecutive duplicate function calls are prohibited in the execution sequence.
5. Never explicitly name APIs in the problem description. Carefully review the function description. You must ensure all function calls in the provided ground truth strictly comply with the specifications. For example, certain filesystem operations are restricted to the current working directory and cannot process multi-level paths. Only functions explicitly defined in the function description may appear in the ground truth.
6. For every input data given, you must return the same number of corresponding processed data.

6. You have permission to edit the question and ground_truth fields, and your response must explicitly return modified versions. The output format should follow the usage examples in **JSON format** instead of the origin data.

You can use the following APIs:

[API Document]

Here is one usage examples that you must follow:

```
[
  {
    "id": "number_0",
    "question": "Hey there, I need to set up a directory titled 'Reports' in my current workspace. Once that's in place, could you help me locate a file called 'summary.doc' in this vicinity and transfer it to the new 'Reports' folder? After that, I'd appreciate if you could find and open up 'data.txt', then scan through it to identify lines that mention 'Q4 financials'. Also, could you let me know the total number of lines in 'data.txt'?",
    "ground_truth": ["mkdir(dir_name='Reports')", "mv(source='summary.doc', destination='Reports')", "cat(file_name='data.txt')", "grep(pattern='Q4 financials', file_name='data.txt')", "wc(file_name='data.txt', mode='l')"],
    "thought": "your thought"
  },
  {
    "id": "number_1",
    "question": "Let's bring some order to the project documents. I want to human readable log the storage usage of the entire current directory to usage.txt file. The content of the file should be the number followed by the word bytes and nothing else.",
    "ground_truth": ["du(human_readable=True)", "touch(file_name='usage.txt')", "echo(content='205 bytes', file_name='usage.txt')"],
    "thought": "your thought"
  }
]
```

where 'id' is the id of task, 'question' is the user query, 'ground_truth' is the shortest function calls sequences used to solve the query, and 'thought' is your ideas for refining this query.

Figure 8: prompt for reconstruction.

1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241

Reconstructed Data

Task:
Locate a file with “ideas” in its name within the ‘documents’ folder, display its contents, and create a backup copy named ‘ideas_backup.txt’ in the same directory.

Config:

```
{
  "root": {
    "simona": {
      "type": "directory",
      "contents": {
        "documents": {
          "type": "directory",
          "contents": {
            "ideas.txt": {
              "type": "file",
              "content": "Collaboration leads to success. Innovation ignites growth."
            },
            "Archived": {
              "type": "directory",
              "contents": {}
            },
            "past_projects": {
              "type": "directory",
              "contents": {}
            }
          }
        }
      }
    }
  }
}
```

Trajectory:

```
[
  "cd(folder='documents')",
  "find(name='ideas')",
  "cat(file_name='ideas.txt')",
  "cp(source='ideas.txt', destination='ideas_backup.txt')"
]
```

Figure 9: An example of reconstructed data: GorillaFileSystem.

1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295

API Document

```

{
  "name": "mv",
  "description": "This tool belongs to the Gorilla file system. It is a simple
file system that allows users to perform basic file operations such as
navigating directories, creating files and directories, reading and writing to
files, etc. Tool description: Move a file or directory from one location to
another. so",
  "parameters": {
    "type": "dict",
    "properties": {
      "source": {
        "type": "string",
        "description": "Source name of the file or directory to move. Source
must be local to the current directory."
      },
      "destination": {
        "type": "string",
        "description": "The destination name to move the file or directory
to. Destination must be local to the current directory and cannot be
a path. If destination is not an existing directory like when
renaming something, destination is the new file name. "
      }
    }
  },
  "required": [
    "source",
    "destination"
  ],
  "response": {
    "type": "dict",
    "properties": {
      "result": {
        "type": "string",
        "description": "The result of the move operation."
      }
    }
  }
}

```

Figure 10: An example of a API document: TravelAPI.

1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349

Correct Trajectory

User:
Below is all tasks you need to finish:

```
{id:'trading_0',question:'Can you get me the ticker for 'Alpha Tech', see its current price, and then execute a purchase of 20 shares at a price of $1320.5?'}
{id:'file_11',question:'Hey, can you set up a new directory named 'Projects' right in workspace folder?'}
```

Assistant:

```
{
  "id": "trading_0",
  "func_name": "get_symbol_by_name",
  "params": {"name": "Alpha Tech"}
}
```

Environment:

```
{"role": "tool", "content": "{\"wait\": \"The current function is being executed.\"}"}
```

Assistant:

```
{
  "id": "file_11",
  "func_name": "cd",
  "params": {"folder": "workspace"}
}
```

Environment:

```
{"role": "tool", "content": "{\"wait\": \"The current function is being executed.\"}"},
{"role": "tool", "id": "trading_0", "function name": "get_symbol_by_name(name='Alpha Tech)", "response": "{\"symbol\": \"ALPH\"}"}
```

Assistant:

```
{
  "id": "trading_0",
  "func_name": "get_stock_info",
  "params": {"symbol": "ALPH"}
}
```

Environment:

```
{"role": "tool", "content": "{\"wait\": \"The current function is being executed.\"}"},
{"role": "tool", "id": "file_11", "function name": "cd(folder='workspace)", "response": "{\"current_working_directory\": \"workspace\"}"}
```

Assistant:

```
{
  "id": "file_11",
  "func_name": "mkdir",
  "params": {"dir_name": "Projects"}
}
```

Environment:

```
{"role": "tool", "content": "{\"wait\": \"The current function is being executed.\"}"},
{"role": "tool", "id": "trading_0", "function name": "get_stock_info(symbol='ALPH)", "response": "{\"price\": 1320.45, \"percent_change\": -0.08, \"volume\": 1.567, \"MA(5)\": 1321.12, \"MA(20)\": 1325.78}"}
```

Assistant:

```
{
  "id": "trading_0",
  "func_name": "place_order",
  "params": {"order_type": "Buy", "symbol": "ALPH", "price": 1320.5, "amount": 20}
}
```

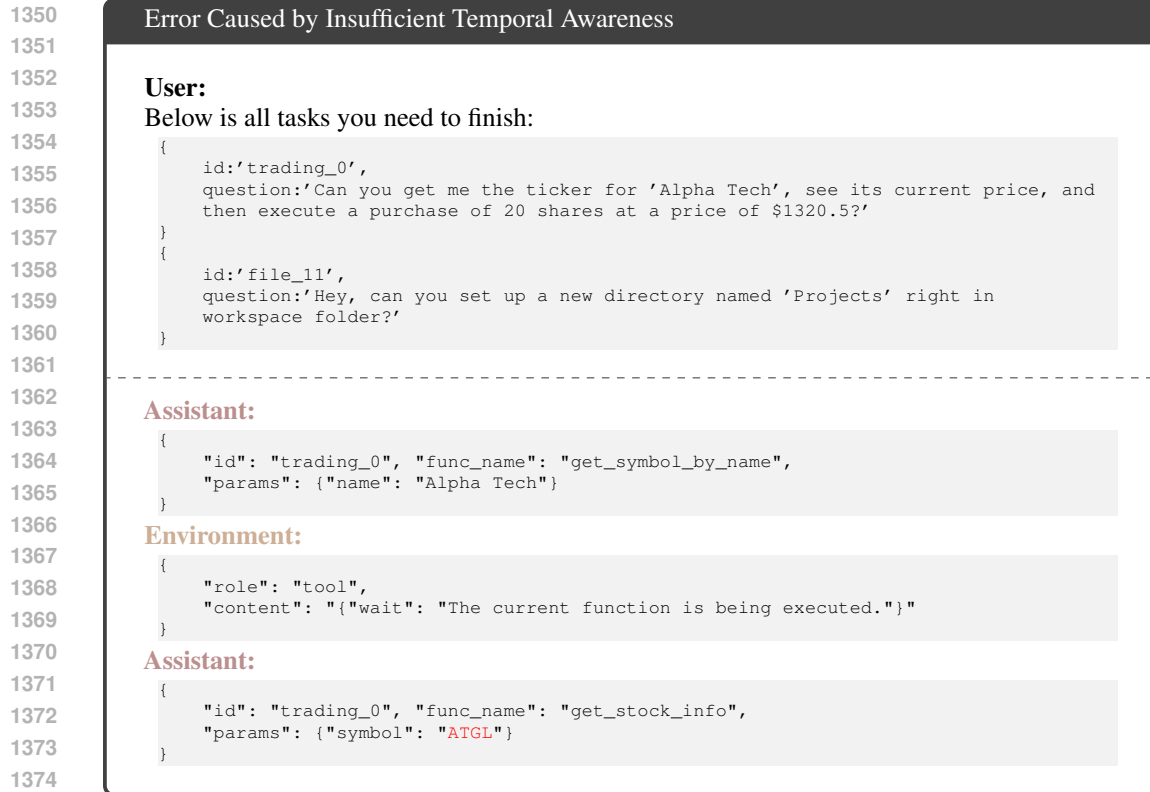
Environment:

```
{"role": "tool", "content": "{\"wait\": \"The current function is being executed.\"}"},
{"role": "tool", "id": "file_11", "function name": "mkdir(dir_name='Projects)", "response": "None"}
```

Assistant:

```
{"content": "ALL COMPLETED"}
```

Figure 11: An example of correct trajectory.



1375
1376
1377
1378
1379

Figure 12: An example of an error caused by insufficient temporal awareness. In this case, the agent prematurely assumed the symbol of Alpha Tech to be “ATGL” before receiving the actual call result.



Figure 13: An example of an error caused by tool confusion. In this case, the agent mistakenly applied the tool intended for task “file_11” to task “trading_0”.

1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457

Error Caused by Neglecting a Certain Task

User:
Below is all tasks you need to finish:

```

{
  id:'file_13',
  question:'Compare 'academic_venture/goals.txt' with
'academic_venture/reference_goals.txt' to verify if they are identical, then
rename 'goals.txt' to 'master_goals.txt' in the same directory.'
}
{
  id:'SM_11',
  question:'I have a list of integers [1, 2, 3, 4, 5]. I want to convert this list
into a string representing a space-separated sequence of those integers. Then, I
want to normalize this string by replacing all non-alphanumeric characters with
underscores and converting to lowercase.'
}
{
  id:'posting_11',
  question:'Now that I'm following 'AI_enthusiast', I'd like to know more about
them. Please retrieve their user statistics. After that, fetch all of their
tweets. I remember seeing one about 'ethics'; find that specific tweet and add
the comment 'A very important point to consider.' to it.'
}

```

Assistant:
{ "id": "file_13", "func_name": "cd", "params": { "folder": "academic_venture" } }

Environment:
{ "role": "tool", "content": "{ \"wait\": \"The current function is being executed.\" }" }

Assistant:
{ "id": "SM_11", "func_name": "integer_list_to_string", "params": { "integer_list": [1, 2, 3, 4, 5] } }

Environment:
{ "role": "tool", "content": "{ \"wait\": \"The current function is being executed.\" }", { "role": "tool", "id": "file_13", "function name": "cd(folder=\"academic_venture\")", "response": "{ \"current_working_directory\": \"academic_venture\" }" }

Assistant:
{ "id": "file_13", "func_name": "diff", "params": { "file_name1": "goals.txt", "file_name2": "reference_goals.txt" } }

Environment:
{ "role": "tool", "content": "{ \"wait\": \"The current function is being executed.\" }", { "role": "tool", "id": "SM_11", "function name": "integer_list_to_string(integer_list=[1, 2, 3, 4, 5])", "response": "1 2 3 4 5" } }

Assistant:
{ "id": "SM_11", "func_name": "normalize_string", "params": { "s": "1 2 3 4 5" } }

Environment:
{ "role": "tool", "content": "{ \"wait\": \"The current function is being executed.\" }", { "role": "tool", "id": "file_13", "function name": "diff(file_name1=\"goals.txt\", file_name2=\"reference_goals.txt\")", "response": "{ \"diff_lines\": \"- Research topic selection\\\\\\\\n+ Data analysis\\\\\\\\n- Literature review\\\\\\\\n+ Data collection\\\\\\\\n- Data collection\\\\\\\\n+ Draft writing\\\\\\\\n- Data analysis\\\\\\\\n+ Final submission\\\\\\\\n- Draft writing\\\\\\\\n+ Literature review\\\\\\\\n- Final submission\\\\\\\\n+ Research topic selection\" }" }

Assistant:
{ "id": "file_13", "func_name": "mv", "params": { "source": "goals.txt", "destination": "master_goals.txt" } }

Environment:
{ "role": "tool", "content": "{ \"wait\": \"The current function is being executed.\" }", { "role": "tool", "id": "SM_11", "function name": "normalize_string(s=\"1 2 3 4 5\")", "response": "1_2_3_4_5" } }

Assistant:
{ "content": "ALL COMPLETED" }

Figure 14: An example of error caused by neglecting a certain task. In this case, the task “file_13” was ignored by the agent