GraphToolBench: Benchmarking LLMs for Sequential Graph Comprehension and Conflict Identification in Tool Learning

Anonymous ACL submission

Abstract

Large language models (LLMs) have advanced rapidly but still exhibit outdated knowledge, unreliable reasoning, and limited real-world interaction. Tool learning addresses these gaps by enabling models to call external tools across four stages: task planning, tool selection, tool calling, and response generation. Current benchmarks mainly focus on final answer accuracy and do not assess intermediate capabilities such as sequential graph comprehension or conflict detection between tool outputs and model knowledge; they are also susceptible to network failures and usage costs. We introduce *GraphToolBench*, a comprehensive benchmark that covers all four tool learning stages and reduces practical constraints by providing an offline *Model Context Protocol (MCP) Function* library of executable Python functions derived from online tools. We develop a sampling procedure, named *Conflict Potential Random Sampling (CPRS)* to produce tool sets with controllable levels of disagreement between tool results and model knowledge. In view of these tool sets, we integrate the advanced LLM with human expertise to generate data that align with the characteristics of the four phases of tool learning. We further present *GraphToolEval*, a multi-dimensional evaluation suite that measures sequential graph understanding and conflict identification. Empirical results demonstrate deeper and more granular insights than prior benchmarks. Code and data are available at `https://anonymous.4open.science/r/GraphToolBench`.

1 Introduction

Recent years have witnessed the impressive capabilities of large language models (LLMs) across multiple domains (Song et al., 2024; Šmíd et al., 2025; Song et al., 2025), marking a significant breakthrough in the development of artificial intelligence. However, their practical applications are still constrained by notable limitations, including outdated knowledge (Qu et al., 2025b), insufficient reliability in logical reasoning (Mallen et al., 2023), and a lack of interaction with the real world (Inaba et al., 2023). To address these constraints, tool learning has emerged, which aims to enable LLMs to invoke external tools (e.g., APIs for weather query and Google news search (Nokia, 2025)), integrating their general cognitive abilities with the real-time performance and professional functions of tools to facilitate the realization of more advanced intelligent applications (Ye et al., 2025).

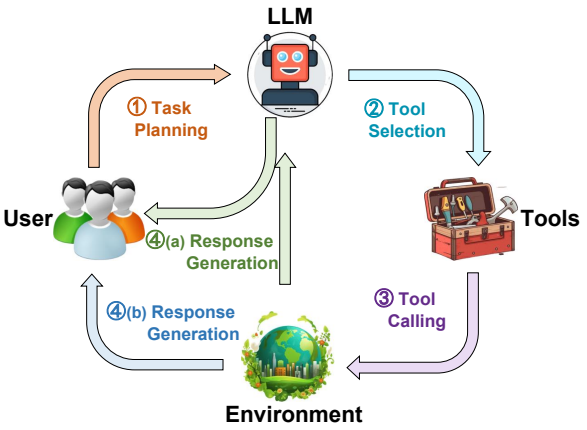

Figure 1: Workflow of tool learning. LLMs first perform ①*Task Planning* to decompose user queries into fine-grained subtasks, then match corresponding tools from the toolset via ②*Tool Selection*, followed by ③*Tool Calling*. The tool execution results from the environment are processed through two distinct ④*Response Generation*. (a) Indirect Mode: The LLM synthesizes tool outputs into a single, natural-language response, validating results and adapting them to the user's context to improve accuracy and clarity. It represents a major focus of current academic research; and (b) Direct Mode: The tool outputs are returned to the user unchanged, making invocation validity and result correctness difficult to verify and reducing interpretability.

Tool learning generally comprises four stages: task planning, tool selection, tool calling, and response generation (Qu et al., 2025b), as illustrated in Figure 1. To systematically evaluate and enhance

Aspect	GraphToolBench (Ours)	ShortcutsBench (SHEN et al., 2025)	ToolEyes (Ye et al., 2025)	SoAyBench (Wang et al., 2025a)	TOOLSANDBOX (Lu et al., 2025)	StableToolBench (Guo et al., 2024)	TaskBench (Shen et al., 2024b)	ToolBench (Qin et al., 2024)	ToolLens (Qu et al., 2024)	APIBench (Patil et al., 2024)
Sequential Graph Comprehension ?	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Conflict Identification ?	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Offline Evaluation ?	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
One-time Multi-step Reasoning ?	✓	✗	✗	✓	✗	✗	✓	✓	✗	✗
Number of Tools	671	1414	568	7	34	16464	103	16464	464	1645
Number of Queries	15376	7627	382	792	1032	12657	17331	12657	18770	17002
Stages involved	①②③④	②③	①②③④	②③	③	③④	①②③	①②③④	①②	②③

Table 1: Comparison of our GraphToolBench with existing benchmarks. Symbols ①, ②, ③, and ④ represent the four stages in Figure 1: task planning, tool selection, tool calling, and response generation, respectively.

the performance of LLMs across different stages of tool learning, several benchmark tests have been proposed, such as ToolBench (Qin et al., 2024), ShortcutsBench (SHEN et al., 2025), and ToolSandBox (Lu et al., 2025), detailed in Appendix A for **Related Work**. However, most existing benchmarks only cover limited stages, mainly testing models' performance by expanding the number of tools and scenario types, while lacking in-depth analysis of factors influencing the performance of LLMs' tool learning, as depicted in Table 1. Specifically, there are three shortcomings as follows:

(1) Overlooking the critical role of sequential graph comprehension in tool learning. The tool learning pipeline fundamentally involves task graph generation and the recognition and retrieval of tool graphs (Shen et al., 2024b; Wu et al., 2024), whose graph structures are commonly encoded as textual sequential diagrams. As reported by Wu et al. (2024), even a high-performance LLM like GPT-4-turbo produces hallucinations in sequential graph comprehension at a rate of nearly 30%, which demonstrates that LLMs struggle to accurately interpret sequence graphs. Moreover, any misinterpretation of these graphs by LLMs can cause the entire tool learning process to fail, highlighting the essential need to assess LLMs' capabilities in understanding sequential graphs.

(2) Lacking assessment of LLMs' ability to identify conflicts between tool execution results and their internal knowledge. Current benchmarks primarily focus on verifying the consistency between LLMs' final outputs and predefined ground truths (Ye et al., 2025). However, this consistency alone is insufficient for comprehensively evaluating tool learning performance. When the LLM receives results from tools, it must determine whether they conflict with its internal knowledge. If tool execution outputs are objectively incorrect and LLMs fail to detect them, it ultimately can lead to erroneous outputs of LLMs or even the propagation of improper conclusions (Suzgun et al., 2025). This conflict identification capability is fundamental for enabling autonomous error correction, trustworthy decision-making, and continuous model evolution (Xu et al., 2024b). It is critical for the safe deployment of LLMs in high-stakes domains such as healthcare, finance, and law (Singhal et al., 2025; Suzgun et al., 2025), underscoring the importance of evaluating this ability in LLMs.

(3) Suffering from network anomalies and financial constraints. Existing benchmarks often rely on online tools, leading to reproducibility failures due to network or authentication issues. StableToolBench (Guo et al., 2024) mitigates this via a virtual API server that simulates responses on failure, improving stability but not guaranteeing correctness. Furthermore, these online APIs are commercialized, and their large-scale invocation incurs substantial financial costs, which hinders in-depth research on tool learning. Therefore, deploying online tools offline can mitigate the drawbacks of remote execution, but ensuring stable and correct tool invocation in an offline environment remains a significant challenge.

Given these limitations, it is essential to construct a novel and offline benchmark for sequential graph comprehension and conflict identification in tool-augmented LLMs. However, two exclusive challenges arise. First, constructing a high-quality data covering four stages of tool learning is challenging, since these stages possess distinct focuses on sequential graph comprehension and conflict identification, resulting in various data requirements for each stage. It is non-trivial to ensure that such data are offline and free of charge, sufficiently aligned with real-world user scenarios, and consistent across the four stages of each scenario. Second, designing effective evaluation methods for each stage is particularly difficult, since the stages target different objectives and therefore require tailored metrics for respective characteristics.

To address the above challenges, we introduce **GraphToolBench**[1], a systematic evaluation bench-

[1]The name **GraphToolBench** originates from its primary purpose of evaluating LLMs' sequential **graph** comprehension in **tool** learning, as a dedicated **bench**mark.

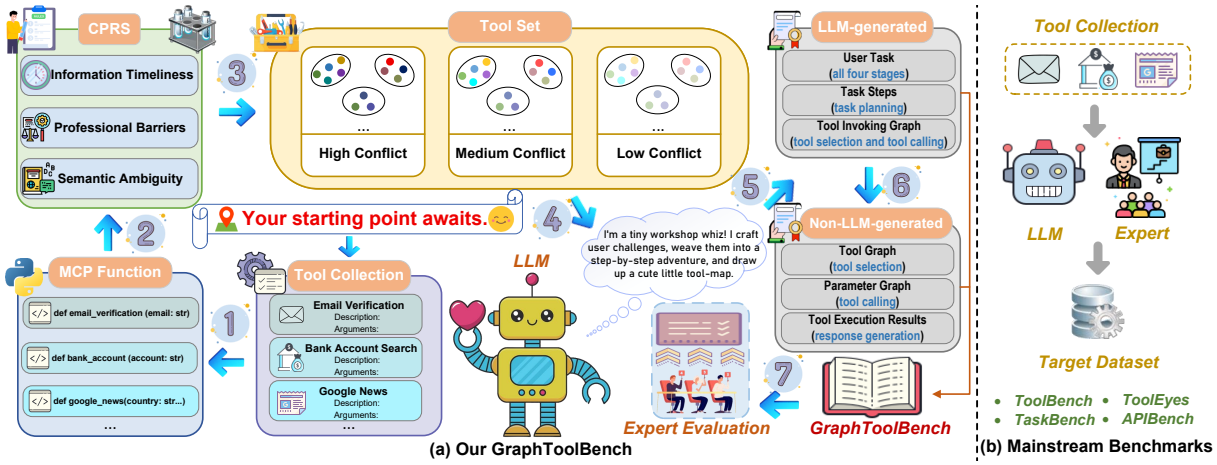

Figure 2: (a) Construction of the GraphToolBench with seven steps: ① On Tool Collection, we convert the tools into offline MCP Functions. ② Based on the proposed CPRS, we evaluate the conflict potential of each tool. ③ We uniformly at random sample multiple tool sets and categorize them into three types including high, medium, and low conflict. ④ We input the relevant descriptions of all tools in each tool set into the LLM. ⑤ Then, we prompt the LLM to generate realistic data including *User Task*, *Task Steps*, and *Tool Invoking Graph*, where blue texts indicate the stage to which the data apply (the same below). ⑥ Based on the three aforementioned elements, we manually generate the *Tool Graph* and *Parameter Graph*, and automatically invoke tools to produce *Tool Execution Results*. ⑦ We invite human experts to conduct quality evaluation of the data. (b) The development of mainstream benchmarks including ToolBench (Qin et al., 2024), TaskBench (Shen et al., 2024b), ToolEyes (Ye et al., 2025), and APIBench (Patil et al., 2024) relies on directly feeding collected tool information to the LLM or human experts.

mark encompassing four stages with the following three designs: (i) We construct a **Model Context Protocol (MCP) Function** library, which includes abundant offline executable Python programs converted by online tools collected to mitigate financial constraints and network anomalies; (ii) we propose **Conflict Potential Random Sampling (CPRS)** to assess the potential of various tools to trigger conflict identification in LLMs, based on which we derive tool sets with varying conflict levels through sampling. In light of these tool sets, we integrate the advanced LLM with human expertise to generate data that align with the characteristics of the four phases and conduct an advanced expert-based quality evaluation for ensuring the construction of diverse and realistic application scenarios; (iii) we design **GraphToolEval**, a multi-dimensional benchmark that rigorously evaluates LLMs' comprehension of sequential graphs across task planning, tool selection, tool invocation, and conflict identification during response generation. Table 1 highlights its innovations in *Sequential Graph Comprehension*, *Conflict Identification*, *Offline Evaluation*, and *One-time Multi-step Reasoning*.

Through extensive experiments and analysis across multiple LLMs, our GraphToolBench effectively captures LLMs' performance and variation for sequential graph comprehension and conflict identification in four stages of tool learning, demonstrating a significant advance over existing ones in depth and dimensionality, with novel findings such as: (1) *current mainstream LLMs face significant performance bottlenecks in sequential graph comprehension and conflict identification*; and (2) *future efforts should prioritize model architecture design and specialized capability training, as scaling model parameters alone fails to resolve these bottlenecks fundamentally*, which pave the way for future enhancements in LLMs' tool learning.

2 GraphToolBench

This section describes the construction of GraphToolBench and its pipeline, shown in Figure 2(a); further details appear in the subsections below. In contrast, mainstream benchmarks in Figure 2(b) generate target data by uniformly providing collected tools to either LLMs or experts. This method overlooks the diverse data needs of the four tool learning stages and the distinct generation strategies they require–precisely the gap that GraphToolBench is designed to address.

2.1 Tool Collection

Drawing on StableToolBench (Guo et al., 2024), we source tools from ToolBench (Qin et al., 2024), an open-source dataset containing 16,464 real-world APIs. Each API includes detailed information, as shown in *Tool Documentation* in Figure 13. Given the substantial number of functionally

similar APIs in ToolBench (e.g., distinct-named APIs for email verification), which would introduce redundancy to subsequent data generation, we implement a two-step fully automated filtering pipeline: (1) We match the pre-defined functional category keywords in ToolBench (e.g., *Email Verification*) against the target API name (e.g., *Quick Email Verification*) to identify its most relevant category based on vector similarity. (2) Following LightRAG (Guo et al., 2025), GPT-4o-mini (OpenAI, 2024b) is employed to conduct in-depth tool documentation analysis and determine whether the target API is functionally redundant with existing APIs in that category, thereby eliminating duplicate APIs, yielding a set of tools with unique functionalities. In total, we collect 671 tools.

2.2 MCP Function Conversion

Since APIs collected require online requests, which are subject to network instability and can become costly when called at scale owing to their commercial nature, we address these limitations by creating offline executable Python functions that simulate APIs' functionalities, as shown in *Step* ① of Figure 2(a). Specifically, based on each API's description and parameter information, corresponding Python functions are implemented manually by expert coders and wrapped into MCP[1] components supporting offline local deployment and invocation, using the @mcp.tool decorator from FastMCP[2]. Furthermore, we verify their executability and correctness by passing different example parameters to these functions offline and obtaining respective outputs successfully. A complete example of MCP function conversion is provided in Figure 13.

2.3 Conflict Potential Random Sampling

To effectively evaluate LLMs' capability in identifying conflicts between tool execution results and their internal knowledge, we propose *Conflict Potential Random Sampling (CPRS)*, which assesses the potential of various tools to provoke conflict identification in LLMs, and samples tool sets of different conflict levels based on these potential values. Specific details are outlined below:

Indicator Formulation. To facilitate the assessment of tool conflict potential, we have established the following three indicators based on relevant literature (Deng et al., 2023; Bi et al., 2025; Prandi et al., 2025): (1) *Information Timeliness*, which reveals how temporal validity of tool execution results affects the triggering of conflicts. Real-time data (e.g., stocks, news) are more likely to contradict LLMs' pre-trained knowledge than static data (e.g., historical events). This indicator is divided into three sub-criteria (low to high): *Static Data (+1)*, *Near-real-time Data (+2)*, *Real-time Data (+3)*, where *1*, *2*, *3* denote corresponding conflict potential scores using the three-point scale from Yang et al. (2021) (same below); (2) *Professional Barriers*, which reflects how the tool's specialized knowledge in certain high-threshold fields increases the likelihood of model errors due to insufficient or biased training data, further subdivided into three types: *Data-Rich & General-Purpose Fields (+1)*, *Variable-Data & Context-Sensitive Fields (+2)*, and *High-Stakes & Expertise-Dependent Fields (+3)*; (3) *Semantic Ambiguity*, which evaluates the comprehensibility of tool descriptions and parameter specifications, significantly affecting tool-task matching accuracy, partitioned into *Precise Understanding (+1)*, *Acceptable Understanding (+2)*, and *Ambiguous Understanding (+3)*. Poor matching may lead to erroneous tool execution results, thereby triggering conflicts. Detailed specifications are provided in Appendix C. These defined metrics enable effective quantification of each tool's conflict potential.

Conflict Potential Assessment. Upon completing formulation of the above three indicators, we proceed to evaluate the conflict potential of each tool based on these established metrics. Specifically, we define the conflict potential of a target tool (t) under the *Information Timeliness*, *Professional Barriers*, and *Semantic Ambiguity* indicators as $I(t)$, $P(t)$, and $S(t)$, respectively, with its cumulative conflict potential score denoted as $C(t)$. Thus, the formula is derived as:

$$C(t) = I(t) + P(t) + S(t) \quad (1)$$

For example, if we already find that the target tool's functionality belongs to *Real-time Data*, operates within *Data-Rich & General-Purpose Fields*, and exhibits *Ambiguous Understanding* of documentation, its $C(t)$ would be automatically calculated as 3 + 1 + 3 = 7. However, a critical problem arises: **how to recognize which subcategory of the three indicators a target tool falls into**. In-

[1]The Model Context Protocol (MCP) is an open standard introduced by Anthropic, designed to connect LLMs to different data sources and tools (Anthropic, 2024).

[2]FastMCP is a lightweight Python framework that simplifies the development of MCP servers, providing high-level interfaces and decorators (e.g., @mcp.tool, @mcp.resource), abstracting low-level protocol details and allowing developers to quickly wrap functions into MCP components with minimal boilerplate code (Prefect, 2024).

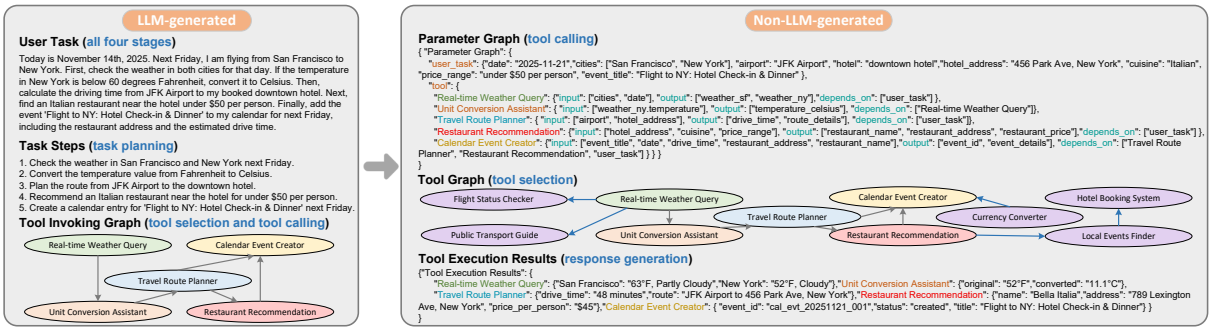

Figure 3: A complete data example from our GraphToolBench, aligned with *Step* ⑥ in Figure 2(a).

spired by GraphRAG (Edge et al., 2024) and LightRAG (Guo et al., 2025), which utilize GPT models like GPT-4o-mini (OpenAI, 2024b) as evaluators, we recognize that LLMs can deliver objective, comprehensive, and automated evaluation capabilities, thus reducing substantial human efforts. Similarly, Ye et al. (2025) finds that the agreement level between GPT-4 (OpenAI et al., 2023) and human evaluations exceed 83.50% consistently across all dimensions, which confirms the validity and reliability of LLM-based assessment. Among contemporary models, GPT-5 (OpenAI, 2025) stands out as the authoritative, high-performing, and recently developed option. Consequently, we employ GPT-5 to classify each tool according to the established indicators, with the final conflict potential score derived through cumulative assignment of corresponding values. The prompt is detailed in Figure 14 of Appendix F.

Sampling and Categorization. After obtaining the conflict potential score $C(t)$ for each tool, we generate multiple tool sets through random sampling and calculate their average conflict potential score, defined as $C_{\text{avg}}(t)$ as follows:

$$C_{\text{avg}}(t) = \frac{1}{n}\sum_{i=1}^{n} C(t_i) \tag{2}$$

where n and t_i denote the total number of tools in the tool set and different tools, respectively. These tool sets are then classified into three conflict levels based on $C_{\text{avg}}(t)$: high conflict ($7 < C_{\text{avg}}(t) \leq 9$), medium conflict ($5 < C_{\text{avg}}(t) \leq 7$), and low conflict ($3 \leq C_{\text{avg}}(t) \leq 5$), as illustrated in *Steps* ② and ③ of Figure 2(a). To maintain experimental consistency and mitigate performance degradation in smaller models which are sensitive to context length and may fail to generate valid output when overloaded with tool descriptions, we restrict each tool set to 1–10 tools. §3.1 empirically validates the soundness of this design choice.

2.4 Data Generation and Quality Evaluation

As depicted in *Steps* ④, ⑤, ⑥, and ⑦ of Figure 2(a), following the methodology of TaskBench (Shen et al., 2024b) and ToolBench (Qin et al., 2024) for LLM-based data production, we employ the authoritative LLM (GPT-5) to facilitate dataset generation based on the aforementioned tool sets. As an example shown in Figure 3, for a tool set containing five tools (i.e., *Real-time Weather Query*, *Unit Conversion Assistant*, *Travel Route Planner*, *Restaurant Recommendation*, and *Calendar Event Creator*), we prompt the LLM to generate the corresponding *User Task*, *Task Steps*, and *Tool Invoking Graph*. Subsequently, we augment *Tool Invoking Graph* with additional tools (e.g., *Flight Status Checker*, *Currency Converter*, *Public Transport Guide*, *Hotel Booking System*, and *Local Events Finder*) to form expanded *Tool Graph*, extract *Parameter Graph* based on *User Task* & *Tool Invoking Graph* , and automatically invoke tools from MCP functions to obtain *Tool Execution Results*. Detailed descriptions of these distinct graphs are provided in Appendix D. In total, we obtain 15,376 user tasks or queries. For the data generated by the LLM (i.e. *User Task*, *Task Steps*, and *Tool Invoking Graph*), we conduct quality evaluation and comparison against various baselines detailed in Appendix E. The thorough experimental results and analysis presented in §3 also substantiate the viability and superior quality of our datasets. The prompt template is provided in Figure 15 of Appendix F.

3 GraphToolEval

To comprehensively evaluate LLMs' capabilities in sequential graph comprehension and conflict identification, we propose GraphToolEval, a systematic evaluation framework based on GraphToolBench covering four distinct stages: task planning, tool selection, tool calling, and response generation. As illustrated in Figure 4, the framework

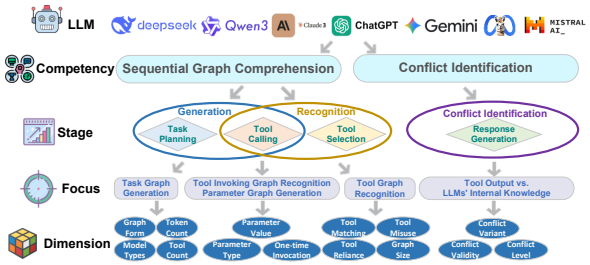

Figure 4: The framework of GraphToolEval.

assesses sequential graph comprehension through both generation (task planning) and recognition (tool selection) capabilities, where tool calling involves both aspects. The response generation stage specifically evaluates conflict identification. Each stage employs dedicated evaluation metrics and focuses on different dimensions. Notably, each phase is evaluated independently without inter-phase dependency, whose isolation prevents error propagation between phases and ensures accurate assessment of individual capabilities. Our evaluation covers major open-source small LLM families (DeepSeek-R1 (DeepSeek-AI, 2025), Llama2 (Touvron et al., 2023), Mistral (Mistral-AI, 2025), Qwen3 (Yang et al., 2025)) across various parameter scales, alongside proprietary LLMs (GPT (OpenAI, 2024a), Gemini (Google, 2025), Claude (Anthropic, 2025)), providing comprehensive coverage of contemporary mainstream architectures. Given the large scale of GraphToolBench (containing 15,376 user tasks), full-scale evaluation would be inefficient. Thus, for each experiment we run five rounds; each round uses a uniform random sample of 1,000 tasks, and the reported result is the mean across rounds. This sample size far exceeds the 382 entries of ToolEyes (Ye et al., 2025) and 792 entries of SoAyBench (Wang et al., 2025a), highlighting the rationality of our evaluation design.

3.1 Sequential Graph Comprehension

Task Planning entails analyzing user tasks and decomposing them into fine-grained subtask graphs with dependency relations (Qu et al., 2025b). Accurate decomposition is essential, since errors propagate through later stages and undermine tool learning. To assess LLMs' ability to interpret user tasks and generate sequential task graphs, we instruct LLMs to produce graphs in three formats: **(1) Knowledge Graph (KG)**: representing subtask graphs through triplets formatted as <previous subtask, relation, subsequent subtask>; **(2) Decision Tree (DT)**: each subtask requires explicit parent node specification, with "none" indicating root nodes; **(3) Ordered Steps (OS)**: subtasks described sequentially in natural language, where order denotes execution sequence rather than explicit dependencies. A visual comparison of these formats appears in Figure 16 (Appendix G). Furthermore, we adopt *Task Steps* and *Tool Invoking Graph* from GraphToolBench (Figure 3) as the ground truth, which adequately characterize the decomposed subtasks and their dependencies.

Figure 5 summarizes our findings: **(1)** Small LLMs can decompose tasks effectively (high OS accuracy) but struggle to generate sequential graphs (low KG accuracy); **(2)** increasing model size alone does not reliably improve sequential graph comprehension or generation, as observed in the Mistral and Llama2 series; **(3)** among open-source models, Qwen3 achieves the best overall performance, particularly in sequential task-graph generation. Further analysis of tool quantity and token-count effects is provided in Appendix H.2.

Tool Selection denotes the post-planning process in which LLMs interpret the tool graph and choose the appropriate tool invoking graph to satisfy subtask requirements (Shen et al., 2024b). Let the correct *Tool Invoking Graph* (ground truth) in GraphToolBench be α, the graph recognized by an LLM from the *Tool Graph* based on *User Task* and accurate *Task Steps* during tool selection be β, the set of tools in a graph be *Tool*, and the graph topology be *Structure*. We evaluate selection with three metrics: **(1) Tool Matching**: all tools in α appear in β, i.e., $Tool(\alpha) \in Tool(\beta)$; **(2) Tool Misuse**: β contains tools not present in α, i.e., $\exists\, \gamma \in Tool(\beta)$ while $\gamma \notin Tool(\alpha)$; **(3) Tool Reliance**: β contains a substructure that is topologically isomorphic to α, with the root nodes of this substructure being identical to those of α, i.e., $Structure(\alpha) \in Structure(\beta)$. These metrics yield fine-grained diagnostics that separate tool recall, selection precision, and comprehension of logical relationships.

As described in § 2.4, we augment *Tool Invoking Graph* with additional tools to form expanded *Tool Graph*. We define the number of additional tools as $\boldsymbol{X}$ and the *Tool Selection* in Table 2 presents results with $\boldsymbol{X}$=5, which highlight following insights: **(1)** Small LLMs exhibit markedly weak ability to recognize sequential tool graphs, as reflected by low *Tool Matching* scores that indicate frequent omission of critical tools during selection. **(2)** The similar performance on *Tool Reliance* and *Tool Misuse* suggests that, although these models retain basic

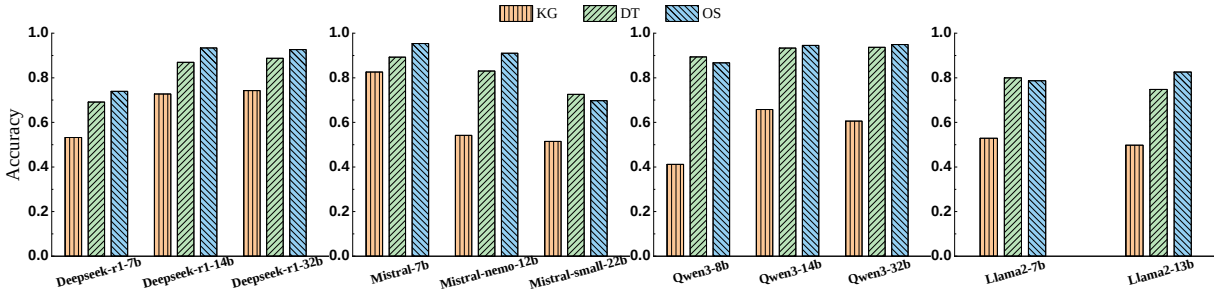

Figure 5: Performance of different open-source LLMs in the task planning stage across three distinct graph formats: KG, DT, and OS. **Experimental results for proprietary LLMs in this phase are available in Appendix H.1**.

Metric	Deepseek-R1			Mistral			Qwen3			Llama2	
	7b	14b	32b	7b	12b	22b	8b	14b	32b	7b	13b
Tool Selection											
Tool Matching↑	0.188	0.272	**0.302**	0.182	0.233	0.159	0.234	0.228	0.257	0.070	0.075
Tool Misuse↓	0.549	0.264	0.244	0.676	0.363	0.388	0.242	0.230	**0.187**	0.854	0.919
Tool Reliance↑	0.401	0.584	0.658	0.459	0.457	0.465	0.648	0.674	**0.712**	0.391	0.316
Tool Calling											
Parameter Value↑	0.506	0.698	0.706	0.593	0.659	**0.796**	0.689	0.544	0.721	0.414	0.318
Parameter Type↑	0.760	0.932	0.912	0.829	0.907	**0.943**	0.868	0.789	0.915	0.680	0.564
One-time Invocation↑	0.135	0.323	0.447	0.241	0.143	0.187	0.209	0.397	**0.642**	0.034	0.103
Response Generation											
R-Time↑	0.132	0.197	0.163	0.214	0.288	0.096	0.293	0.327	0.385	0.115	**0.419**
R-Prof↑	0.573	0.716	0.816	0.548	0.732	0.788	0.633	0.752	**0.883**	0.437	0.563
R-Ambig↑	0.207	0.418	0.443	0.657	0.483	0.294	0.816	0.873	**0.764**	0.078	0.124

Table 2: Accuracy of different open-source LLMs under various metrics in the tool selection, tool calling, and response generation phases, where the best results under each metric are highlighted in bold, and the poorest results are underlined. **Experimental results for proprietary LLMs in these phases are available in Appendix H.1**.

structural understanding, they are prone to tool confusion and lack a robust, global grasp of tool-graph relationships. Analysis across other X values (various tool-graph sizes) is provided in Appendix H.3.

Tool Calling requires LLMs to extract parameters from user queries according to the previously selected tools (i.e., *Tool Invoking Graph*) and to supply these parameters to the tools (Qu et al., 2025b). Because outputs of earlier tools frequently serve as inputs to later ones, parameters form a directed *Parameter Graph*. We evaluate LLMs against the ground-truth parameter graphs generated for each user task in §2.4 along three dimensions. **(1) Parameter Value**: accuracy of extracted parameter values for tool inputs; **(2) Parameter Type**: correct identification and representation of input/output parameter types for each tool in the invoking graph; and **(3) One-time Invocation**: successful single-pass execution of all tools following the parameter-graph dependencies, indicating correct recognition and formatting of parameter information. As all tools are converted to offline executable MCP functions (§2.2), the evaluation is not affected by network latency or interruptions.

Analysis of the *Tool Calling* results in Table 2 shows that small LLMs attain high accuracy in extracting tool parameters–particularly *Parameter Type*–indicating competent recognition of tool-invoking graphs. However, their low *One-time Invocation* performance indicates an inability to generate valid Parameter Graphs, reflecting insufficient understanding of parameter dependencies and incorrect structural formatting.

3.2 Conflict Identification

Response Generation denotes the stage where LLMs combine tool execution outputs with internal knowledge to produce the final user response (Qu et al., 2025b). To evaluate conflict identification in this phase, we precompute tool execution results $\boldsymbol{R}$ by executing parameter graphs (§2.4) and derive three $\boldsymbol{R}$ variants based on CPRS indicators, i.e., *Information Timeliness*, *Professional Barriers*, and *Semantic Ambiguity* (*Indicator Formulation* in §2.3): **(1) R-Time:** for time-sensitive tools, we alter results using data updated after each model's release date (e.g., "*the current US President is Biden*" → "*the current US President is Trump*"); **(2) R-Prof:** for specialized domains with potential professional barriers (e.g., finance, law), we replace outputs with standardized terminology from TermOnline[3] (e.g., "*heart beating irregularly*" →

[3]https://www.termonline.cn

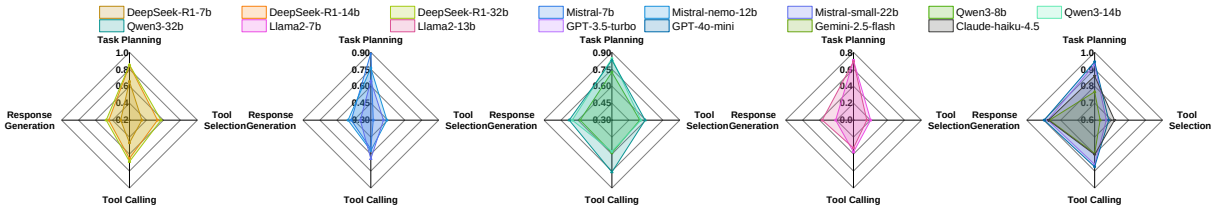

Figure 6: Average performance of various LLMs for each stage.

"*cardiac arrhythmia*"); and **(3) R-Ambig:** for high semantic ambiguity, we substitute outputs with external-tool responses to simulate incorrect tool matching. For example, if the user task is "*Query the weather in London*" with the correct tool output "*Sunny*", the output is replaced with "*The capital of the United Kingdom is London*". These variants enable fine-grained analysis of factors affecting conflict recognition; we measure performance using ***Conflict Validity***. E.g., when given the tool result "*the current US President is Trump*", Qwen3-8B replies: "*As of 2023, the US President is Joe Biden, not Donald Trump. For more recent updates (e.g., post-2024), please consult official sources*", and such responses that acknowledge knowledge limits and express skepticism toward tool outputs are counted as valid conflict identification.

The *Response Generation* results in Table 2 show: (1) R-Time scores are the lowest across all dimensions, indicating that small LLMs struggle most with detecting conflicts in time-sensitive tool outputs and often accept post-cutoff information even when it is incorrect; (2) the Qwen3 series achieves the strongest overall performance, particularly on R-Prof and R-Ambig, whereas the Llama2 series performs worst. These differences suggest that deeper investigation into Qwen3 and Llama2 architectures could inform improvements in conflict identification. Further analysis of conflict levels is provided in Appendix H.4.

We also categorized and analyzed the error types observed across the evaluation stages; detailed examples and analyses are provided in Appendix H.5.

4 Experimental Summary

To summarize LLM performance in sequential graph comprehension and conflict identification at the individual stage level, we compute average scores across the relevant metrics for each phase. Since *Tool Misuse* is lower-is-better, we transform it via *(1-Tool Misuse)*, thus all metrics share the same directional interpretation for averaging. Figure 6 reports these stage-wise averages for both open-source and proprietary LLMs, from which we derive the following summaries.

(1) Significant bottlenecks exist in sequential graph comprehension: Both open-source and proprietary LLMs underperform on stages that require sequential-graph understanding–namely task planning, tool selection, and tool calling. Although models can recognize basic graph structure, they struggle to generate, interpret, and reason about complex sequential relationships such as task dependencies, tool-invocation chains, and parameter dependencies. Increasing model size alone may not mitigate these limitations. *(2) Conflict identification capability remains weak*: During the response-generation phase, all LLMs–especially smaller ones–perform worst at detecting conflicts involving time-sensitive information, exposing a substantial weakness in assessing information timeliness and result plausibility. *(3) Model architecture and specialized training are crucial*: The Qwen3 series' stronger performance across tasks, compared with the weaker Llama2 series, suggests that architectural design and capability-oriented training–rather than parameter count alone–are critical for improving these core competencies.

A discussion of **future directions** is provided in Appendix H.6. The Pearson correlation analysis across stages is presented in Appendix H.7.

5 Conclusion

To address limitations of offline evaluation for LLMs' sequential graph comprehension and conflict identification in tool learning, we propose *GraphToolBench*, a systematic benchmark covering the four key stages of tool learning–task planning, tool selection, tool calling, and response generation. Our designs include: (1) an offline *MCP Function* library to mitigate financial and network constraints; (2) *Conflict Potential Random Sampling* for sampling tool sets with varying conflict levels and in light of these tool sets, high-quality, diverse user queries and tool data generated via the state-of-the-art LLM; and (3) *GraphToolEval*, a multi-dimensional evaluation framework to rigorously assess LLMs' abilities. Our benchmark offers meaningful insights for LLMs' tool learning advancements in the future.

Limitations

While GraphToolBench demonstrates valuable contributions to tool learning, two limitations still remain. First, constrained by budget, we only evaluate four proprietary LLMs from the GPT, Claude, and Gemini series, leaving coverage limited. Second, we have not yet utilized the benchmark to train novel LLMs specifically focused on sequential graph comprehension and conflict identification to further advance tool learning performance. In future, we plan to expand evaluation scope through academic API grants or industry collaborations to include more proprietary LLMs. Additionally, a task-oriented fine-tuning framework could be developed based on the benchmark to train specialized models with enhanced capabilities in sequential graph understanding and conflict detection, thereby improving the accuracy and generalization of tool learning systems.

References

Anthropic. 2024. Model context protocol. Accessed: 2025-11-19.

Anthropic. 2025. Claude. Accessed 2025-11-24.

Baolong Bi, Shenghua Liu, Yiwei Wang, Yilong Xu, Junfeng Fang, Lingrui Mei, and Xueqi Cheng. 2025. Parameters vs. Context: Fine-Grained Control of Knowledge Reliance in Language Models. *arXiv e-prints*, arXiv:2503.15888.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Yinlin Deng, Chunqiu Steven Xia, Haoran Peng, Chenyuan Yang, and Lingming Zhang. 2023. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2023, page 423–435, New York, NY, USA. Association for Computing Machinery.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv e-prints*, arXiv:2404.16130.

Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. 2024. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18030–18038.

Google. 2025. Gemini. Accessed 2025-11-24.

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. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156, Bangkok, Thailand. Association for Computational Linguistics.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2025. LightRAG: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 10746–10761, Suzhou, China. Association for Computational Linguistics.

Tatsuro Inaba, Hirokazu Kiyomaru, Fei Cheng, and Sadao Kurohashi. 2023. MultiTool-CoT: GPT-3 can use multiple external tools with chain of thought prompting. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1522–1532, Toronto, Canada. Association for Computational Linguistics.

Yilun Kong, Jingqing Ruan, YiHong Chen, Bin Zhang, Tianpeng Bao, Shi Shiwei, du Guo Qing, Xiaoru Hu, Hangyu Mao, Ziyue Li, Xingyu Zeng, Rui Zhao, and Xueqian Wang. 2024. TPTU-v2: Boosting task planning and tool usage of large language model-based agents in real-world industry systems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 371–385, Miami, Florida, US. Association for Computational Linguistics.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183, Albuquerque, New Mexico. Association for Computational Linguistics.

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

Mistral-AI. 2025. Models. Accessed 2025-11-24.

Lorenzo Molfetta, Giacomo Frisoni, Nicolò Monaldini, and Gianluca Moro. 2025. PORTS: Preference-optimized retrievers for tool selection with large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 10018–10041, Suzhou, China. Association for Computational Linguistics.

Nokia. 2025. Rapidapi. Accessed 2025-11-24.

OpenAI. 2024a. ChatGPT. Accessed 2025-11-24.

OpenAI. 2024b. GPT-4o-mini. Accessed: 2024-07-18.

OpenAI. 2025. GPT-5. Accessed: 2025-08-07.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 181 others. 2023. GPT-4 Technical Report. *arXiv e-prints*, arXiv:2303.08774.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565. Curran Associates, Inc.

Matteo Prandi, Vincenzo Suriani, Federico Pierucci, Marcello Galisai, Daniele Nardi, and Piercosma Bisconti. 2025. Bench-2-CoP: Can We Trust Benchmarking for EU AI Compliance? *arXiv e-prints*, arXiv:2508.05464.

Prefect. 2024. Fastmcp. Accessed: 2025-11-19.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Towards completeness-oriented tool retrieval for large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 1930–1940, New York, NY, USA. Association for Computing Machinery.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025a. From exploration to mastery: Enabling LLMs to master tools via self-driven interactions. In *The Thirteenth International Conference on Learning Representations*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-rong Wen. 2025b. Tool learning with large language models: a survey. *Frontiers of Computer Science*, 19(8):198343.

Haiyang SHEN, Yue Li, Desong Meng, Dongqi Cai, Sheng Qi, Li Zhang, Mengwei Xu, and Yun Ma. 2025. Shortcutsbench: A large-scale real-world benchmark for API-based agents. In *The Thirteenth International Conference on Learning Representations*.

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. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16658–16680, Miami, Florida, USA. Association for Computational Linguistics.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2024b. Taskbench: Benchmarking large language models for task automation. In *Advances in Neural Information Processing Systems*, volume 37, pages 4540–4574. Curran Associates, Inc.

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*, WWW '25, page 2222–2237, New York, NY, USA. Association for Computing Machinery.

Karan Singhal, Tao Tu, Juraj Gottweis, Rory Sayres, Ellery Wulczyn, Mohamed Amin, Le Hou, Kevin Clark, Stephen R. Pfohl, Heather Cole-Lewis, Darlene Neal, Qazi Mamunur Rashid, Mike Schaekermann, Amy Wang, Dev Dash, Jonathan H. Chen, Nigam H. Shah, Sami Lachgar, Philip Andrew Mansfield, and 16 others. 2025. Toward expert-level medical question answering with large language models. *Nature Medicine*, 31(3):943–950.

Jakub Šmíd, Pavel Priban, and Pavel Kral. 2025. LACA: Improving cross-lingual aspect-based sentiment analysis with LLM data augmentation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 839–853, Vienna, Austria. Association for Computational Linguistics.

Hwanjun Song, Hang Su, Igor Shalyminov, Jason Cai, and Saab Mansour. 2024. FineSurE: Fine-grained summarization evaluation using LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 906–922, Bangkok, Thailand. Association for Computational Linguistics.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li.

2023. RestGPT: Connecting Large Language Models with Real-World RESTful APIs. *arXiv e-prints*, arXiv:2306.06624.

Yuncheng Song, Liang Ding, Changtong Zan, and Shujian Huang. 2025. Self-evolution knowledge distillation for LLM-based machine translation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 10298–10308, Abu Dhabi, UAE. Association for Computational Linguistics.

Mirac Suzgun, Tayfun Gur, Federico Bianchi, Daniel E. Ho, Thomas Icard, Dan Jurafsky, and James Zou. 2025. Language models cannot reliably distinguish belief from knowledge and fact. *Nature Machine Intelligence*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv e-prints*, arXiv:2307.09288.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. LLMs in the imaginarium: Tool learning through simulated trial and error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10583–10604, Bangkok, Thailand. Association for Computational Linguistics.

Yuanchun Wang, Jifan Yu, Zijun Yao, Jing Zhang, Yuyang Xie, Shangqing Tu, Yiyang Fu, Youhe Feng, Jinkai Zhang, Jingyao Zhang, Bowen Huang, Yuanyao Li, Huihui Yuan, Lei Hou, Juanzi Li, and Jie Tang. 2025a. Soay: A solution-based llm api-using methodology for academic information seeking. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, KDD '25, page 2660–2671, New York, NY, USA. Association for Computing Machinery.

Zezhong Wang, Xingshan Zeng, Weiwen Liu, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. 2025b. ToolFlow: Boosting LLM tool-calling through natural and coherent dialogue synthesis. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4246–4263, Albuquerque, New Mexico. Association for Computational Linguistics.

Xixi Wu, Yifei Shen, Caihua Shan, Kaitao Song, Siwei Wang, Bohang Zhang, Jiarui Feng, Hong Cheng, Wei Chen, Yun Xiong, and Dongsheng Li. 2024. Can graph learning improve planning in LLM-based agents? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024a. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*.

Rongwu Xu, Zehan Qi, Zhijiang Guo, Cunxiang Wang, Hongru Wang, Yue Zhang, and Wei Xu. 2024b. Knowledge conflicts for LLMs: A survey. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8541–8565, Miami, Florida, USA. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Christine Yang, Duanchen Liu, Qingyun Yang, Zoey Liu, and Emily Prud'hommeaux. 2021. Predicting pragmatic discourse features in the language of adults with autism spectrum disorder. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 284–291, Online. Association for Computational Linguistics.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. In *Advances in Neural Information Processing Systems*, volume 36, pages 71995–72007. Curran Associates, Inc.

Junjie Ye, Guanyu Li, SongYang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Tao Ji, Qi Zhang, Tao Gui, and Xuanjing Huang. 2025. ToolEyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 156–187, Abu Dhabi, UAE. Association for Computational Linguistics.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. 2025. EASYTOOL: Enhancing LLM-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 951–972, Albuquerque, New Mexico. Association for Computational Linguistics.

Zijing Zhang, Zhanpeng Chen, He Zhu, Ziyang Chen, Nan Du, and Xiaolong Li. 2025. ToolExpNet: Optimizing multi-tool selection in LLMs with similarity and dependency-aware experience networks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 15706–15722, Vienna, Austria. Association for Computational Linguistics.

Methods	Authenticity		Naturalness		Logicality		Alignment		Overall
	TC	SP	UN	EC	SC	TD	TSC	STA	
Human Gold Standard	4.81	4.75	4.92	4.78	4.87	4.63	4.86	4.93	4.82
GPT-3.5 Generated	4.23	3.93	4.28	4.09	3.98	3.89	4.15	4.30	4.11
GPT-4 Generated	4.34	4.17	4.46	4.32	4.18	4.10	4.29	4.26	4.27
Ours (Ablation: w/o Complex Tasks)	3.83	4.34	**4.77**	4.58	4.35	**4.45**	**4.68**	**4.61**	4.45
Ours	**4.68**	**4.46**	4.74	**4.62**	**4.54**	4.37	4.53	4.46	**4.55**

Table 3: Human evaluation (rating from 1 to 5) on samples constructed by different methods. Average score rating from five human experts, where the best results are highlighted in bold.

A Related Work

Tool Learning. LLMs face limitations including outdated knowledge, unreliable logical reasoning, and limited real-world interaction capabilities (Mallen et al., 2023; Qu et al., 2025b). To address these challenges, tool learning has emerged as a promising direction, enabling LLMs to utilize external tools for solving complex problems. Existing research generally follows the four-stage paradigm of tool learning: task planning (Shen et al., 2024a; Kong et al., 2024; Qu et al., 2025a; Shi et al., 2025), tool selection (Gao et al., 2024; Molfetta et al., 2025; Zhang et al., 2025), tool calling (Yang et al., 2023; Wang et al., 2024; Yuan et al., 2025; Wang et al., 2025b), and response generation (Song et al., 2023; Xu et al., 2024a). These approaches can be broadly categorized into tuning-free methods (leveraging in-context learning capabilities through strategic prompting) and tuning-based methods (directly fine-tuning model parameters for tool usage mastery). For comprehensive details, we recommend readers refer to this survey (Qu et al., 2025b).

Benchmarks for Tool Learning. The advancement of tool learning research has spurred the development of multiple benchmarks to evaluate and enhance LLM performance across different stages (Guo et al., 2024; SHEN et al., 2025; Patil et al., 2025; Ye et al., 2025) . ToolBench (Qin et al., 2024) incorporates 16,464 real-world APIs and diverse usage scenarios to improve tool utilization capabilities. APIBench (Patil et al., 2024) assesses model adaptability to data variations during tool usage. TaskBench (Shen et al., 2024b) provides a comprehensive framework for evaluating task automation proficiency. TOOLSANDBOX (Lu et al., 2025) establishes a stateful, interactive benchmark for conversational tool invocation. ToolEyes (Ye et al., 2025) meticulously examines tool learning capabilities across seven real-world scenarios through multi-dimensional evaluation. Unlike these existing benchmarks, our GraphToolBench innovatively performs in-depth assessments of LLMs' sequential graph comprehension and conflict identification capabilities across four tool learning stages.

B Complete Example Demonstration of MCP Function Conversion

Figure 13 presents an example of MCP Function conversion, demonstrating the transformation of an online tool into an *Offline Executable Program* based on the provided *Tool Documentation*.

C Specific Indicator Information

Table 5 provides a detailed breakdown of each indicator, including its definition and evaluation criteria. The criteria encompass distinct categories with their corresponding cumulative conflict scores.

D Specific Descriptions for Various Graphs in GraphToolBench

Task Steps refer to the fine-grained subtasks obtained through the hierarchical decomposition of a *User Task*. These steps exhibit inherent temporal or logical precedence relationships. During the task planning phase, this representation corresponds to the Ordered Steps (OS) formalism of a *Task Graph*, as defined in §3.1 , and serves as the foundational schema for subsequent tool selection.

Tool Invoking Graph is an abstract workflow representation generated during the tool selection phase. It is constructed by deterministically mapping each subtask defined in the *Task Steps* to a suitable tool from the available *Tool Graph*, while rigorously enforcing the logical dependencies among the selected tools. This graph subsequently serves as the execution blueprint in the tool calling phase, orchestrating parameter extraction and sequential tool invocation.

Tool Graph is a graph structure comprising a set of tools interconnected by logical relationships, within which the *Tool Invoking Graph* constitutes a specific subgraph. For a given *User Task*, the

ground truth *Tool Invoking Graph* is uniquely determined. In addition, the tools present in the *Tool Graph* but not included in this tool invoking subgraph are regarded as interference factors that can impede task resolution. Consequently, *Tool Graph* is utilized during the tool selection phase to assess the accuracy with which LLMs extract the correct *Tool Invoking Graph* from it based on *User Task* and *Task Steps*, whereas the *Tool Invoking Graph* from our GraphToolBench is regarded as the ground truth.

Parameter Graph is an execution-specific dataflow graph instantiated from the *Tool Invoking Graph*. It is constructed by resolving the input parameters for each tool from the *User Task* description and establishing dataflow edges to represent functional composition, where outputs of certain tools become inputs to others. As this graph orchestrates the one-time execution of all tools in the calling phase, it must ensure not only the semantic accuracy of parameter values and the logical correctness of dependencies, but also compliance with required formatting specifications.

E Evaluation of the Dataset Quality

To demonstrate the quality of the GraphToolBench dataset, we conduct a comprehensive human evaluation. We engage five domain experts, including professors, PhD candidates from academia, and senior engineers from industry, to participate in the assessment. Following TaskBench (Shen et al., 2024b), we randomly sample 50 queries (user tasks) from GraphToolBench as evaluation samples. To guarantee an impartial and objective evaluation process, all samples are anonymized. Calibration samples are provided to the experts to standardize assessment criteria and ensure consistent rating practices throughout the annotation procedure.

Evaluation Metrics. The evaluation employs eight metrics across the following four dimensions:

- **Authenticity:** This measures how well the generated data reflect real-world scenarios, including *Task Complexity (TC)* and *Scenario Plausibility (SP)*.
- **Naturalness:** This assesses the fluency and clarity of the language, using metrics like *Utterance Naturalness (UN)* and *Expression Clarity (EC)*.
- **Logicality:** This evaluates the rationality of the task decomposition and tool invocation logic, focusing on *Step Completeness (SC)* and *Tool Dependency (TD)*.
- **Alignment:** This quantifies the alignment coherence between the user task, task steps, and tool invoking graph, measured by *Task-Step Consistency (TSC)* and *Step-Tool Alignment (STA)*.

Following TaskBench (Shen et al., 2024b), each metric is rated on a 5-point scale. These metrics that we design are more fine-grained and comprehensive, thus enhancing the fidelity of evaluation.

Baselines. To ensure a fair comparison, we establish the following additional baselines:

- **GPT-3.5 Generated:** Regenerating these 50 samples using the GPT-3.5 model to represent a moderate baseline, illustrating the performance of a widely-used but less advanced LLM.
- **GPT-4 Generated:** Regenerating these 50 samples using the GPT-4 model to serve as a strong baseline, representing the outstanding general-purpose LLM's capability in synthesizing tool-use tasks.
- **Ablation: w/o Complex Tasks:** A degraded variant of these 50 samples is created by replacing all complex tasks, defined as those containing more than five procedural steps, with simplified tasks consisting of five or fewer steps from our GraphToolBench, used to isolate and validate the contribution of task complexity to overall data quality.
- **Human Gold Standard:** The 50 samples are distributed among these five domain experts, each manually generating 10 samples, thereby establishing the practical upper bound of data quality for evaluation. Note that in the subsequent quality assessment, each expert-generated sample is labeled and assigned to other experts for scoring to ensure anonymity and fairness.

Evaluation Results. Table 3 presents our data quality evaluation results. Compared to other methods, our approach achieves a high score of 4.55, demonstrating the smallest deviation from the *Human Gold Standard* and confirming the high quality of our dataset. Furthermore, results from the *Ablation: w/o Complex Tasks* baseline reveal two observations: (1) Scenario Plausibility (SP) decreases

when task complexity is reduced, reflecting the growing prevalence of complex real-world user scenarios, an advantage captured by our dataset; (2) Task-Step Consistency (TSC) and Step-Tool Alignment (STA) improve accordingly, which aligns with the known tendency of LLMs to perform better on lower-complexity tool-invocation tasks illustrated by the analysis in §3.

F Demonstration of Prompts

Figure 14 demonstrates the prompt for assessing tool conflict potential, and Figure 15 presents the prompt template used for data generation. Moreover, to maintain conciseness and avoid redundancy in the text, all prompts employed in GraphToolEval for evaluating LLMs' abilities in sequential graph comprehension and conflict identification are provided at `https://anonymous.4open.science/r/GraphToolBench` and are excluded from the paper.

G Comparison of KG, DT, and OS

Figure 16 exhibits an intuitive comparison of *User Task* decomposed into three formats: Knowledge Graph (KG), Decision Tree (DT), and Ordered Steps (OS) in the task planning stage.

H Additional Experimental Results and Analysis for GraphToolEval

H.1 Experimental Results of Proprietary LLMs

Figure 7 presents the performance of proprietary LLMs in the task planning stage. Comparative analysis of accuracy across the three graph formats reveals that even massively scaled proprietary LLMs exhibit weaknesses in sequential task graph generation. Furthermore, Table 4 presents the performance evaluation of proprietary LLMs across subsequent phases, revealing persistent limitations in multiple critical dimensions: tool graph recognition, tool invocation graph identification, parameter graph generation, and conflict recognition. These findings collectively demonstrate that simply scaling model parameters does not fundamentally enhance sequential graph comprehension or conflict resolution abilities, which suggests the need for deeper architectural innovations and specialized capability training beyond parameter expansion, thereby providing crucial guidance for future research directions in tool learning.

H.2 Impact of Tool and Token Quantities in Task Planning Stage

Given that task complexity may significantly impact models' sequential graph generation capability, and the number of tools required to solve a user task serves as an effective measure of this complexity, we further analyze how tool quantity affects performance. As established in *Sampling and Categorization* of §2.3, the maximum tool set size is 10, meaning that each user task requires up to 10 tools. We accordingly evaluate performance across four tool quantity intervals: $\leq$3, (3,5], (5,7], and $\geq$8. Additionally, considering the substantial influence of token count on model performance, and recognizing that tool descriptions vary in length where three tools may collectively contain more tokens than five others, we also investigate the impact of total input token volume. Based on the maximum observed token count of 1224, we establish four evaluation intervals: $<$450, [450,700), [700,950), and $\geq$950.

Figures 9 and 10 present the evaluation results for tool quantity and token count effects, respectively. The findings demonstrate that as tool counts and token volumes increase, LLMs' sequential graph generation performance generally declines. This degradation is particularly pronounced in the open-source model DeepSeek-R1-7B and the proprietary model Gemini-2.5-Flash. Therefore, mitigating the negative impact of tool quantity and token volume on sequential graph generation represents a crucial challenge for future research in tool learning.

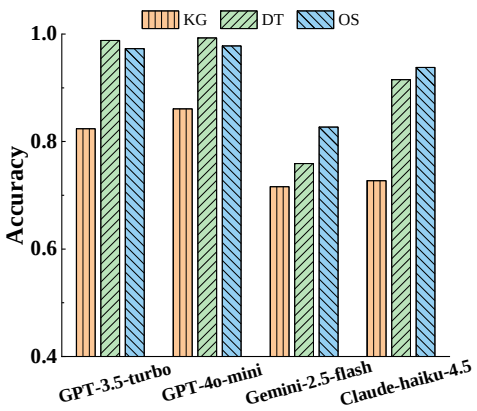

Figure 7: Performance of various proprietary LLMs in the task planning stage across three distinct graph formats: KG, DT, and OS.

H.3 Impact of Tool Graph Size in Tool Selection Stage

We systematically vary the value of X from 1 to 10 to investigate the impact of tool graph size on LLMs' graph recognition capability. As clearly demonstrated in Figure 11, three distinct trends emerge as X increases: *Tool Matching* shows a consistent decline, while *Tool Misuse* exhibits a corresponding rise, indicating progressively higher probabilities of tool omission and misselection by LLMs. Simultaneously, the decreasing trend in *Tool Reliance* reveals deteriorating capabilities in both global structural comprehension and local relationship identification as graph complexity grows. These patterns collectively demonstrate the significant negative correlation between expanding tool graph size and model performance in tool selection.

H.4 Impact of Conflict Level in Response Generation Stage

As described in the *Sampling and Categorization* (§2.3), we have categorized the tool sets in GraphToolBench into three conflict levels: high, medium, and low. To further assess the impact of conflict levels on model performance, we provide the execution results $\boldsymbol{R}$ (where R-Time, R-Prof, and R-Ambig represent modified variants of $\boldsymbol{R}$, as introduced in §3.2) from tool sets of each level to LLMs, and evaluate the effectiveness of conflict identification accordingly.

Figure 12 presents our experimental results. With the exceptions of DeepSeek-R1-32B, Qwen3-8B, and Gemini-2.5-Flash, we can observe a general trend that LLMs achieve the highest valid conflict recognition rates for high conflict level results and the lowest rates for low conflict level results. This indicates that models are more easily misled by outputs with low conflict levels, where contradictions tend to be more subtle and less detectable. This finding highlights a significant challenge for future LLM development: the need to equip models with comprehensive, fine-grained mechanisms for triggering doubt and enabling thorough reflective reasoning. Moreover, these results validate the robust design and effectiveness of GraphToolBench.

H.5 Discussion on Error Examples

We summarize the error types observed across different evaluation stages and present corresponding examples, as depicted in Tables 6, 7, 8, and 9, which correspond to the four critical stages: Task Planning, Tool Selection, Tool Calling, and Response Generation. Based on these four tables, a coherent analysis of the underlying error mechanisms can be summarized as follows:

(1) The errors observed in **Task Planning** (Table 6), such as *Incomplete Decomposition* and *Context Loss Across Sub-tasks*, stem primarily from the LLM's **incomplete comprehension of task graph boundaries and implicit constraints**. The LLM tends to capture only the most explicit user instructions, while overlooking real-world prerequisites and efficiency considerations. This reflects a form of semantic shallow parsing, where the model fails to perform deep situational reasoning, thereby propagating ambiguity and omission into later stages.

(2) During **Tool Selection** (Table 7), these comprehension shortcomings evolve into **functional misalignment**. As illustrated by *Irrelevant Tool Distraction* and *Contextual Constraint Ignorance*, the LLM is misled by superficial interface similarities and neglects higher-order constraints. This indicates a lack of deep functional reasoning within a complex tool graph ecosystem, where the LLM struggles to align tool capabilities with nuanced task requirements.

(3) Errors in **Tool Calling** (Table 8), such as *Conditional Branch Collapse* and *Structural Serialization Failure*, highlight the LLM's deficiencies in **structured execution and formal precision**. The LLM tends to flatten logical branches into linear sequences and produces malformed output schemas (e.g., JSON missing commas or brackets). These are not merely understanding errors but failures in procedural rigor and engineering accuracy, **the inability to reliably translate high-level plans into machine-executable, well-formed invocations**.

(4) Finally, in **Response Generation** (Table 9), challenges shift from execution to **critical integration and authority assessment**. Errors like *Temporal Conflict Blindness* and *Domain Terminology Oversimplification* reveal the model's difficulty in judging source credibility, reconciling conflicting information, and elevating outputs to professional standards. The model **prioritizes tool outputs over its own knowledge** without explicit qualification and fails to provide actionable verification guidance, demonstrating weaknesses in knowledge synthesis and meta-cognitive communication.

In summary, these staged errors expose a progression from **cognitive-understanding deficits** to **engineering-implementation failures**, and fi-

Metric	GPT-3.5-turbo	GPT-4o-mini	Gemini-2.5-flash	Claude-haiku-4.5
Tool Selection				
Tool Matching↑	0.408	**0.527**	0.345	0.473
Tool Misuse↓	0.166	0.207	0.139	**0.108**
Tool Reliance↑	0.764	0.729	0.691	**0.786**
Tool Calling				
Parameter Value↑	**0.885**	0.859	0.797	0.823
Parameter Type↑	0.956	**0.973**	0.938	0.914
One-time Invocation↑	0.741	**0.792**	0.655	0.671
Response Generation				
R-Time↑	**0.813**	0.781	0.762	0.696
R-Prof↑	0.935	0.973	0.989	**0.995**
R-Ambig↑	0.883	**0.937**	0.848	0.926

Table 4: Accuracy of various proprietary LLMs across various metrics in the tool selection, tool calling, and response generation phases, where the best results under each metric are highlighted in bold, and the poorest results are underlined.

nally to **judgment-and-integration shortcomings**. The root cause lies in the inherent nature of contemporary LLMs as probabilistic pattern connectors rather than systems equipped with deep causal reasoning, systematic engineering thinking, or robust value-judgment frameworks. When faced with multi-step, constraint-rich, open-ended tool-use sequential graph tasks, their architectural limitations become systematically evident across planning, selection, invocation, and response formulation.

H.6 Recommendation from Empirical Summaries

(1) Architectural Innovations: Future research should transcend parameter scaling and focus on novel model architectures that natively support modeling and reasoning of complex sequential relationships and dependency logic. *(2) Capability-Oriented Training*: Targeted training paradigms need to be developed to strengthen specialized model capabilities such as sequential graph comprehension, conflict recognition, and causal reasoning, moving beyond reliance on general-purpose pretraining objectives. *(3) Knowledge Updates and Critical Thinking*: Robust mechanisms–independent of prompts, context, or agent assistance–are needed to enable models to detect, question, and manage outdated or conflicting information, thereby improving their reliability and decision-making safety in dynamic environments. In-depth analysis of advanced architectures like Qwen3 and GPT will provide a valuable and momentous blueprint for these innovations.

H.7 Pearson Correlation Coefficient Investigation

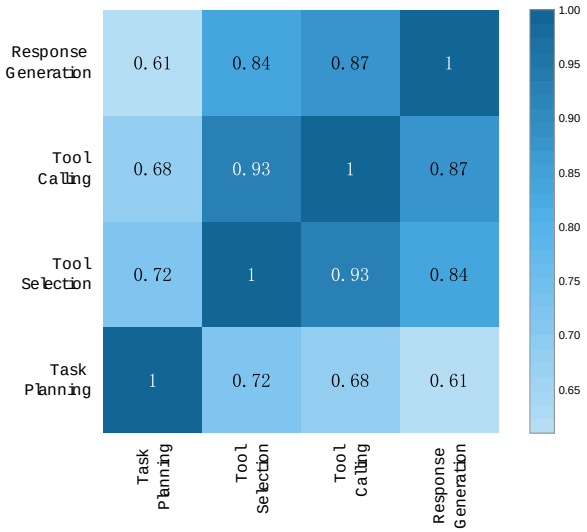

Figure 8: Pearson correlation coefficients between LLMs' various capabilities in four stages.

To conduct a deeper analysis of the correlation between sequential graph comprehension and conflict identification across different phases, we calculate Pearson correlation coefficients, as illustrated in Figure 8. Our analysis reveals the following valuable and important insights:

The task planning phase, as the starting point

of sequential generation, shows relatively moderate correlations (0.61–0.72) with subsequent graph recognition stages. This suggests that a model's performance in task decomposition and graph generation is somewhat independent and may rely more on its inherent reasoning priors rather than directly constraining later recognition of tool graphs and tool invoking graphs. This indicates a potential disconnect between graph comprehension in the initial planning stage and subsequent execution stages.

In contrast, the tool selection phase (tool graph recognition) and the tool calling phase (tool invocation graph recognition and parameter graph generation) exhibit almost synchronous performance, with a correlation coefficient of 0.93. This strong alignment indicates that the model's capability to recognize tool dependencies, extract parameters, and form invocation logic is highly consistent, reflecting coherent graph comprehension throughout the execution process.

Conflict identification, which occurs in the response generation phase, shows high correlation with the tool calling phase (0.87), suggesting that the accuracy of parameter graph generation significantly influences the model's ability to detect conflicts between tool execution results and its internal knowledge. Errors in tool invocation are likely to hinder effective conflict identification during result integration. The response generation phase also correlates strongly with tool selection (0.84), further indicating that the entire execution chain, from tool graph recognition to parameter graph generation, collectively underpins conflict detection.

Accordingly, during tool learning, LLMs exhibit a "decoupled planning, tightly coupled execution" pattern in sequential graph comprehension: task planning remains relatively independent, while tool selection, tool calling, and response generation form a highly correlated cluster. Within this cluster, the coordination of graph recognition and generation is crucial for accurate conflict identification. These findings suggest that enhancing tool learning performance should focus on strengthening the structural continuity between task planning and execution and improving the quality of parameter graph generation during tool calling, to ultimately bolster overall conflict recognition capabilities.

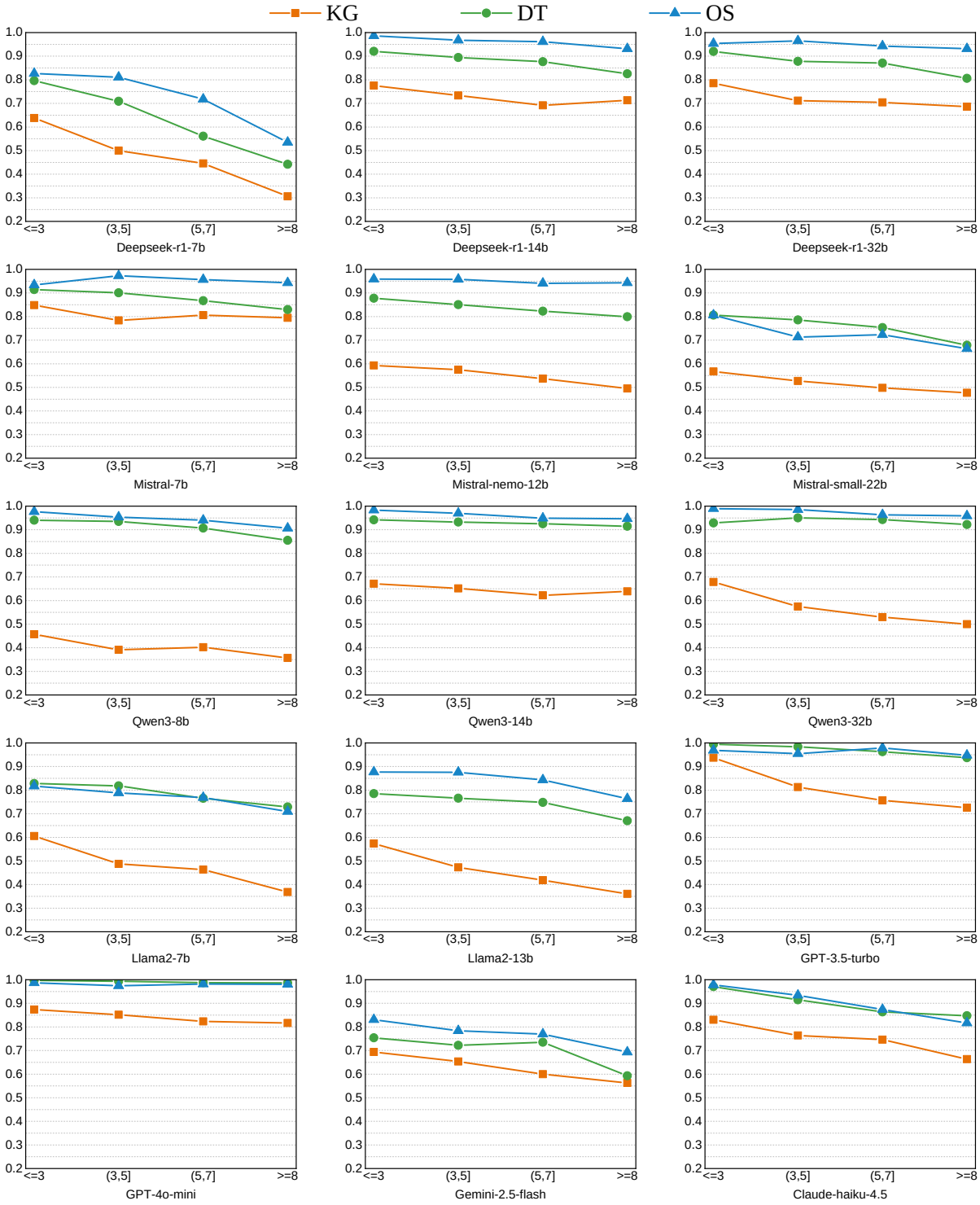

Figure 9: Performance of different LLMs in the task planning stage across distinct tool quantities.

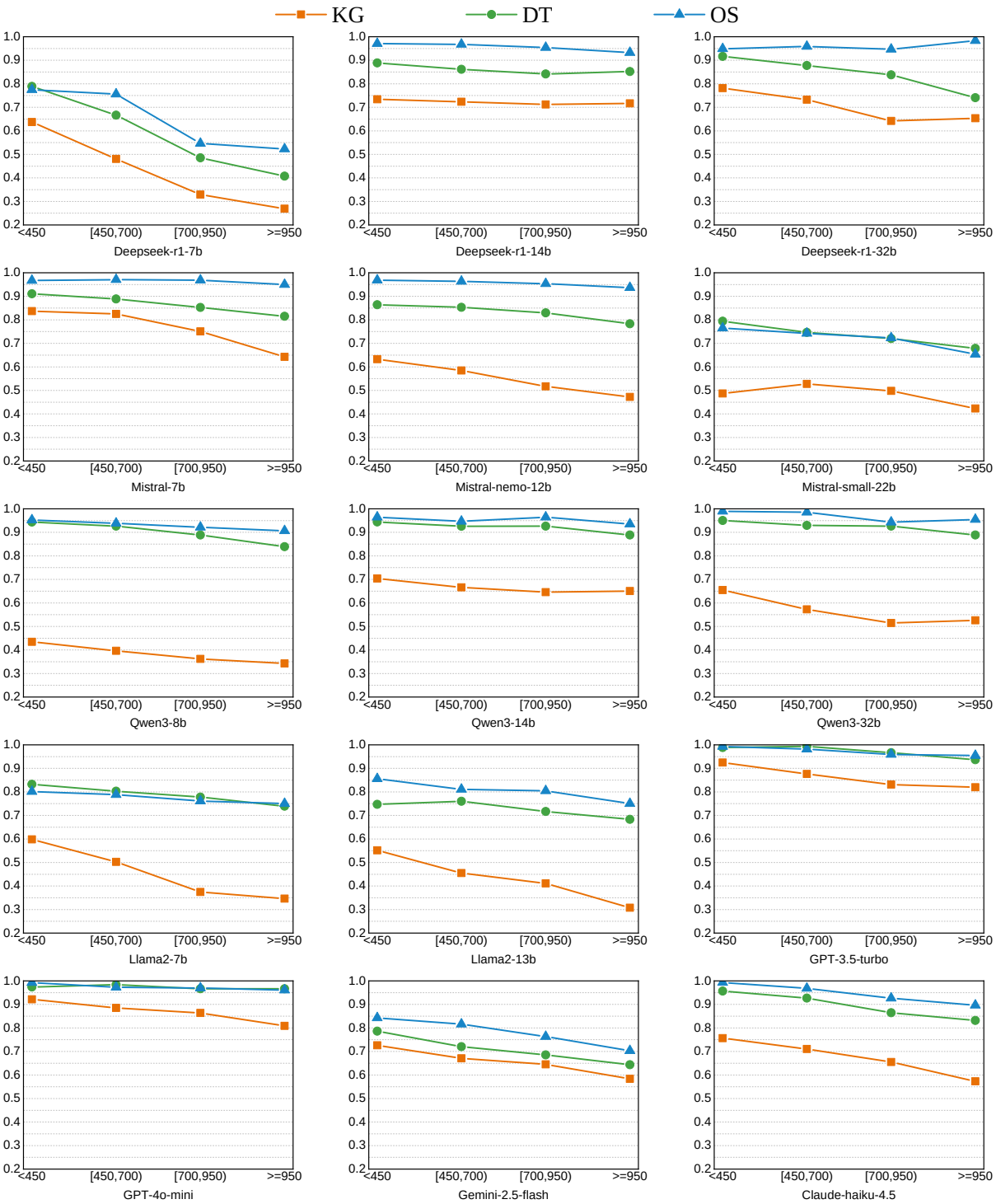

Figure 10: Performance of different LLMs in the task planning stage across distinct token quantities.

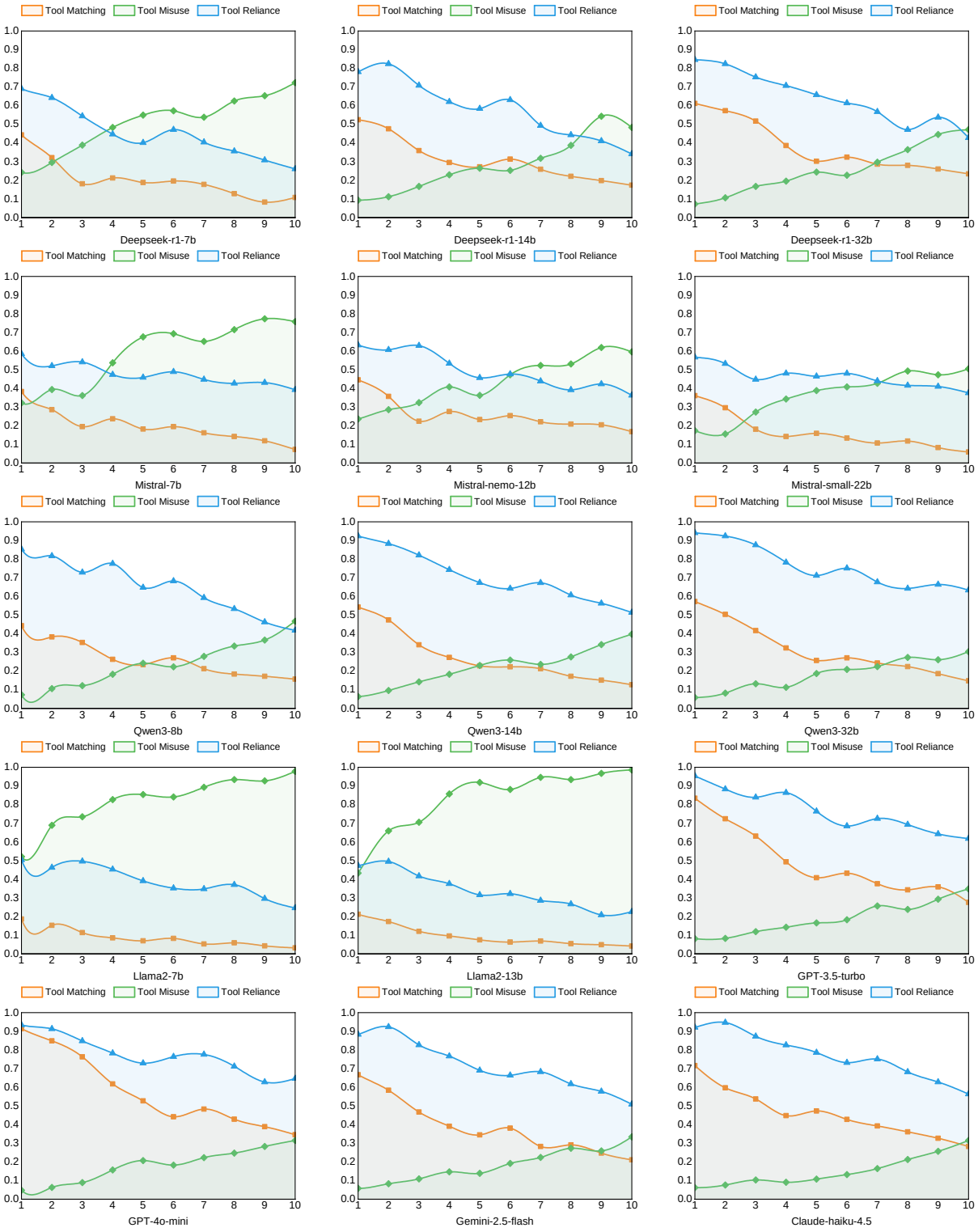

Figure 11: Performance of different LLMs in the tool selection stage across distinct X values, i.e., tool graph sizes.

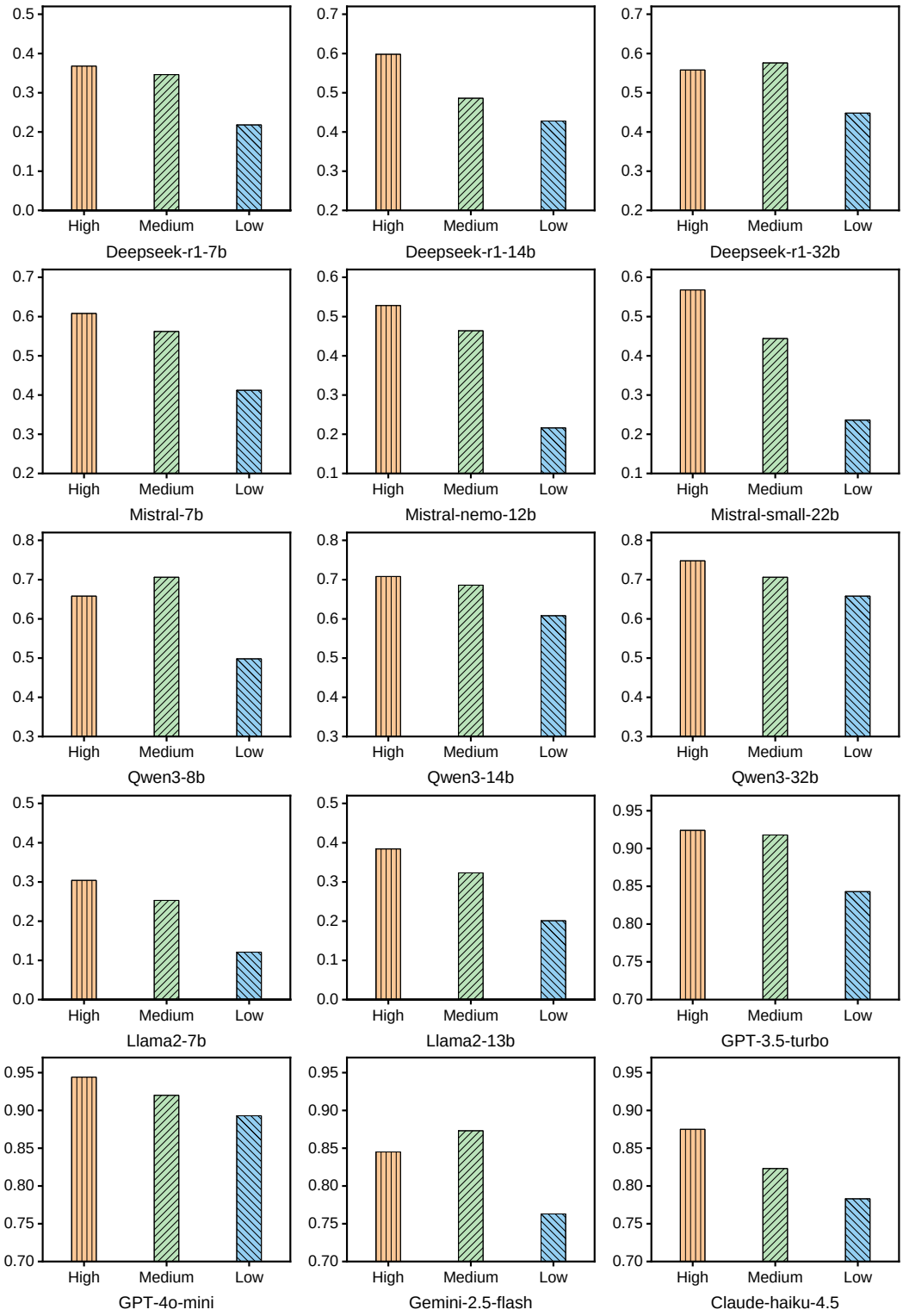

Figure 12: Performance of different LLMs in the response generation stage across distinct conflict levels

Tool Documentation

Tool Name: Google News **Headers:** google-news22.p.rapidapi.com
Tool Description: Find news articles using specific keywords, topics, date ranges, languages, and more. Access both real-time and historical trending news. Obtain comprehensive information about each article, including thumbnails, keywords, authors, and descriptions.
Required Parameters: (1) Country (*String, 2-letter ISO 3166-1 code of the country*) (2) Topic (*String, the topic field specifies the category of interest for the content you are requesting*) (3) Language (*String, 2-letter ISO 639-1 code of the article language.*)
Optional Parameters: (1) Date (Date, yyyy-mm-dd) (2) Page (Number, Default: 0)

Offline Executable Program

```
@mcp.tool()
def google_news(country: str, language: str, topic: str, date: Optional[str] = None, page: int = 0) -> Dict:
    """Description: Find news articles using specific keywords, topics, date ranges, languages, and more
       Returns: Dictionary containing status, total results, and articles"""
    # Prestore new data
    news_database = [
      {
          "title": "Google to invest mid-single-digit billion amount in Germany, says industry source",
          "description": "An industry source says Alphabet's Google will invest a mid-single-digit billion amount in Germany",
          "publishedAt": "2025-11-11",
          "country": "us",
          "language": "en",
          "topic": "business"
      },
      {
          "title": "谷歌据悉将于下周宣布德国投资计划，涉及基础设施和数据中心建设",
          "description": "据报道，谷歌计划公布其在德国有史以来规模最大的投资计划",
          "publishedAt": "2025-11-06",
          "country": "de",
          "language": "zh",
          "topic": "technology"
      },
      {
          "title": "Google's AI Data Centre Project on Christmas Island",
          "description": "Google announced plans to build a major AI data centre on Christmas Island",
          "publishedAt": "2025-11-10",
          "country": "au",
          "language": "en",
          "topic": "technology"
      }
    ]
    # Validate language parameter
    if language not in ['en', 'zh']:
        return {
            "status": "error",
            "message": "Only 'en' (English) and 'zh' (Chinese) languages are supported",
            "totalResults": 0,
            "articles": []
        }
    # Filter articles based on parameters
    filtered_articles = []
    for article in NEWS_DATABASE:
        # Check country match
        if article.get('country') != country:
            continue
        # Check language match
        if article.get('language') != language:
            continue
        # Check topic match (case-insensitive)
        if article.get('topic', '').lower() != topic.lower():
            continue
        # Check date filter if provided
        if date:
            try:
                article_date = datetime.datetime.strptime(article['publishedAt'], '%Y-%m-%d').date()
                filter_date = datetime.datetime.strptime(date, '%Y-%m-%d').date()
                if article_date != filter_date:
                    continue
            except ValueError:
                # Skip if date parsing fails
                continue
        filtered_articles.append(article)
    # Simple pagination (5 articles per page for demonstration)
    articles_per_page = 5
    start_index = page * articles_per_page
    end_index = start_index + articles_per_page
    paginated_articles = filtered_articles[start_index:end_index]
    # Prepare response
    response = {
        "status": "ok",
        "totalResults": len(filtered_articles),
        "articles": paginated_articles
    }

    return response
```

Figure 13: A complete example demonstration of MCP Function conversion. The *Tool Documentation* for the *Google News* tool is converted into an offline executable Python program. The function name matches the tool name, its parameters include all tool arguments, and the docstring contains a textual description and return type. Notably, we preload a variable named *news_database* with queryable news data to enable feasible offline execution.

Indicator	Definition	Criterion (Cumulative Conflict Score)
Information Timeliness	Revealing how temporal validity of tool execution results impact conflict triggering. Real-time data (e.g., stocks, news) are more likely to contradict LLMs' pre-trained knowledge than static data (e.g., historical events)	**(1) Real-time Data (+3):** Frequently updated information such as exchange rates, daily weather forecasts, and news reports **(2) Near-real-time Data (+2):** Information with slower update cycles including legal regulations, corporate financial data, and demographic statistics **(3) Static Data (+1):** Non-changing historical records and established facts
Professional Barriers	Reflecting how the tool's specialized knowledge in certain high-threshold fields increases the likelihood of model errors due to insufficient or biased training data	**(1) High-Stakes & Expertise-Dependent Fields (+3):** Domains requiring deep, specialized knowledge with sensitive or scarce data and where incorrect outputs can cause severe consequences (health, safety, major finance, legally protected data) **(2) Variable-Data & Context-Sensitive Fields (+2):** Domains with variable data quality and availability, moderate-to-high domain knowledge needs, and significant context or cultural dependence; errors are impactful but generally less catastrophic such as personalized educational tutoring and marketing content generation **(3) Data-Rich & General-Purpose Fields (+1):** Domains with abundant public data, low-to-moderate domain expertise required, low sensitivity like daily knowledge Q&A, basic programming assistance, and copywriting&email composition
Semantic Ambiguity	Evaluating the comprehensibility of tool descriptions and parameter specifications, which significantly affects tool-task matching accuracy. Poor matching may lead to erroneous tool execution	**(1) Ambiguous Understanding (+3):** Poorly defined tool descriptions and unclear parameter specifications that frequently result in incorrect tool-task matching and execution failures **(2) Acceptable Understanding (+2):** Sufficiently clear documentation that enables basic operational comprehension, though some aspects of functionality or parameter usage may require interpretation **(3) Precise Understanding (+1):** Well-structured documentation with explicit parameter definitions that ensure accurate tool-task alignment and reliable execution systems

Table 5: Indicators for assessing tool conflict potential.

You will be given an API name and a comprehensive API description. For each API, classify it on three independent dimensions (temporal risk, expertise risk, semantic ambiguity) and provide a brief, evidence-based rationale for each classification. Produce only a single JSON object (no surrounding text or code blocks) with the exact schema shown below.

Definitions and classification criteria:
- Temporal risk (information timeliness)
  - Real-time data: Highly time-sensitive, frequently updated streams (e.g., stock prices, daily news, live weather). Such data often conflicts with fixed pretraining; model hallucination risk is high.
  - Near-real-time data: Periodically updated authoritative data (e.g., laws/regulations, corporate financial filings, census/demographics). Updates occur at moderate intervals; conflicts possible but less frequent.
  - Static data: Infrequently changing or historical facts (e.g., historical events). Conflict risk is lowest though training bias errors remain possible.

- Expertise risk (professional barrier)
  - High-Stakes & Expertise-Dependent Fields : Domains requiring deep, specialized knowledge with sensitive or scarce data and where incorrect outputs can cause severe consequences (health, safety, major finance, legally protected data).
  - Variable-Data & Context-Sensitive Fields: Domains with variable data quality and availability, moderate-to-high domain knowledge needs, and significant context or cultural dependence; errors are impactful but generally less catastrophic such as personalized educational tutoring and marketing content generation.
  - Data-Rich & General-Purpose Fields: Domains with abundant public data, low-to-moderate domain expertise required, low sensitivity like daily knowledge Q&A, basic programming assistance, and copywriting&email composition.

- Semantic ambiguity (clarity of API description)
  - Ambiguous understanding: Poorly defined tool descriptions and unclear parameter specifications that frequently result in incorrect tool-task matching and execution failures.
  - Acceptable understanding: Sufficiently clear documentation that enables basic operational comprehension, though some aspects of functionality or parameter usage may require interpretation .
  - Precise understanding: Well-structured documentation with explicit parameter definitions that ensure accurate tool-task alignment and reliable execution systems.

Output format (strict JSON):
{
  "temporal_risk": {"class": "<one of: \"Real-time data\" | \"Near-real-time data\" | \"Static data\">", "reason": "<concise evidence-based rationale>"},
  "expertise_risk": {"class": "<one of: \"High-Stakes & Expertise-Dependent Fields\" | \"Variable-Data & Context-Sensitive Fields\" | \"Data-Rich & General-Purpose Fields\">", "reason": "<concise evidence-based rationale>"},
  "semantic_ambiguity": {"class": "<one of: \"Ambiguous understanding\" | \"Acceptable understanding\" | \"Precise understanding\">", "reason": "<concise evidence-based rationale>"}
}

Guidelines:
- Give concise, evidence-based reasons for classifications.
- Do not output anything other than the required JSON object.

Input:
API_name: {API_name}
API_description: {API_description}

Figure 14: The prompt for assessing tool conflict potential.

You are a professional tool call planning assistant. Please complete the following tasks based on the provided tool set information:

1. Generate a User Task (user_task): Create a natural, fluent user task description that describes a need which would exactly trigger the invocation of all tools in the provided set. The user task should be based on a realistic everyday scenario, logically coherent, and ensure that each tool in the set has a necessary reason to be called.

2. Create Task Steps (task_steps): Break down the user task into granular, ordered steps. Each step should correspond to exactly one tool call from the provided set, maintaining the logical sequence required to complete the task. Number each step sequentially and ensure the description clearly aligns with the purpose of its corresponding tool.

3. Construct a Tool Invoking Graph (tool_invoking_graph): Based on the logic required to solve the user task you generated, analyze the dependency relationships between the various tool calls. Represent these dependencies using knowledge graph triplets. The structure of each triplet is: (Prerequisite_Tool_Name, "next_call", Dependent_Tool_Name). This signifies that the latter tool can only be called after the former tool has been successfully executed. Ensure the graph accurately reflects the logical sequence of the call flow.

Output Format: Your output must and can only be a single, well-formatted JSON object containing exactly these three fields:
{
"user_task": The value is the user task string you generated.

"task_steps": The value is a list (Array). Each element in this list is a string representing a numbered step description, formatted as "Step X: [description of this step]".

"tool_Invoking_graph": The value is a list (Array). Each element in this list is another list (Array) representing a triplet, formatted as ["tool_name_A", "next_call", "tool_name_B"].
}

Please complete the tasks above based on the following provided tool set information:
{Provided_Tools}

Figure 15: The prompt for data generation.

User Task

Today is November 14th, 2025. Next Friday, I am flying from San Francisco to New York. First, check the weather in both cities for that day. If the temperature in New York is below 60 degrees Fahrenheit, convert it to Celsius. Then, calculate the driving time from JFK Airport to my booked downtown hotel. Next, find an Italian restaurant near the hotel under $50 per person. Finally, add the event 'Flight to NY: Hotel Check-in & Dinner' to my calendar for next Friday, including the restaurant address and the estimated drive time.

Knowledge Graph (KG)

< Check the weather in San Francisco and New York next Friday, Providing information, Convert the temperature value from Fahrenheit to Celsius >
< Convert the temperature value from Fahrenheit to Celsius, Preceding, Plan the route from JFK Airport to the downtown hotel >
< Plan the route from JFK Airport to the downtown hotel, Preceding, Recommend an Italian restaurant near the hotel for under $50 per person >
< Plan the route from JFK Airport to the downtown hotel, Providing information, Create a calendar entry for 'Flight to NY: Hotel Check-in & Dinner' next Friday >
< Recommend an Italian restaurant near the hotel for under $50 per person, Providing information, Create a calendar entry for 'Flight to NY: Hotel Check-in & Dinner' next Friday >

Decision Tree (DT)

Node 1. Check the weather in San Francisco and New York next Friday. Parent node: None
Node 2. Convert the temperature value from Fahrenheit to Celsius. Parent node: Node 1
Node 3. Plan the route from JFK Airport to the downtown hotel. Parent node: Node 2
Node 4. Recommend an Italian restaurant near the hotel for under $50 per person. Parent node: Node 3
Node 5. Create a calendar entry for 'Flight to NY: Hotel Check-in & Dinner' next Friday. Parent node: Node 3 and Node 4

Ordered Steps (OS)

1. Check the weather in San Francisco and New York next Friday.
2. Convert the temperature value from Fahrenheit to Celsius.
3. Plan the route from JFK Airport to the downtown hotel.
4. Recommend an Italian restaurant near the hotel for under $50 per person.
5. Create a calendar entry for 'Flight to NY: Hotel Check-in & Dinner' next Friday.

Figure 16: Comparison of *User Task* decomposed into three formats: Knowledge Graph (KG), Decision Tree (DT), and Ordered Steps (OS) in the **Task Planning** stage.

Error Type	Definition	Example
Incomplete Decomposition	Failing to break down the task into all necessary atomic sub-steps, leading to missing critical operations.	User Task: "Book a flight from Shanghai to London for next Monday, then find a 4-star hotel near London Heathrow for three nights." Erroneous decomposition: Step 1: Search flights; Step 2: Book flight; Step 3: Search hotels. Error: Missing sub-steps like checking visa requirements, comparing flight options, verifying hotel cancellation policies, and confirming airport-hotel transportation.
Incorrect Task Graph	Misidentifying the logical or temporal dependencies between sub-tasks, causing invalid execution order.	User: "Summarize the latest research paper on LLM alignment and email it to my team." Erroneous decomposition: Step 1: Send email; Step 2: Download paper; Step 3: Summarize content. Error: The email step depends on the summary, which depends on downloading the paper. The graph should be linear: Download $\rightarrow$ Summarize $\rightarrow$ Send.
Infeasible Sub-task Assignment	Assigning a sub-task that cannot be executed by available tools or APIs, or assuming capabilities that do not exist.	User: "Convert this hand-drawn sketch into a digital architectural blueprint." Erroneous decomposition: Step 1: Use image-to-CAD API to convert sketch to CAD format; Step 2: Adjust dimensions. . . . Error: Assuming an image-to-CAD API exists as a available tool, while in reality no such reliable tool may be available in the current toolset.
Context Loss Across Sub-tasks	Failing to propagate necessary context between dependent sub-tasks.	User: "Summarize customer feedback from April, then highlight complaints about the mobile app." Erroneous decomposition: Step 1: Summarize April feedback $\rightarrow$ output a general summary; Step 2: Extract mobile app complaints from the original data. Error: Step 2 should directly use the filtered or categorized data from Step 1, not start over, to maintain efficiency and relevance.

Table 6: Examples of different error types in task planning stage.

Error Type	Definition	Example
Irrelevant Tool Distraction	Selecting a new, irrelevant tool that shares superficial semantic features (e.g., keywords, domain) with the correct tool, but does not match the functional requirements of the subtask.	Subtasks: Convert the user's spoken query into text. Correct Tool: SpeechToTextTool (input: audio_file, output: text) Distractor Tool Added: AudioEnhancementTool (input: audio_file, output: cleaned_audio_file) (Focuses on audio quality, not transcription). Erroneous Selection: The model selects AudioEnhancementTool because both tools take an audio_file as input, failing to recognize that the subtask's required output is text, not enhanced audio.
Compositional Overcomplication	Selecting multiple, more complex new tools to imitate the functionality of a single, simpler correct tool, violating the principle of minimal sufficiency.	Subtasks: Resize an image to 500x500 pixels. Correct Tool: ImageResizeTool (input: image, dimensions, output: resized_image) Distractor Tools Added: ImageDimensionReadTool (input: image, output: width, height), PixelManipulationTool (input: image, scale_factor, output: scaled_image), ImageCropTool (input: image, coordinates, output: cropped_image). Erroneous Selection: The model composes a flawed plan: ImageDimensionReadTool $\rightarrow$ calculate scale factor $\rightarrow$ PixelManipulationTool $\rightarrow$ ImageCropTool to adjust aspect ratio. This is error-prone and inefficient compared to the single, dedicated ImageResizeTool.
Contextual Constraint Ignorance	Ignoring implicit constraints from the broader task context (e.g., privacy, cost, latency) and selecting a functionally correct but contextually inappropriate new tool.	Broader Task Context: "Process this sensitive user data locally without external API calls." Subtasks: Perform sentiment analysis on the provided text. Correct Tool: LocalSentimentAnalysisTool (On-device, slower but private). Distractor Tool Added: CloudSentimentAPITool (More accurate, but sends data externally). Erroneous Selection: The model selects the more powerful CloudSentimentAPITool, violating the overarching local-only constraint from the main task.

Table 7: Examples of different error types in tool selection stage.

Error Type	Definition	Example
Value Extraction Inaccuracy	Failure to accurately identify and extract explicit parameter values from user queries or contextual sources.	User: "Transfer $150 from checking to savings." Tool: BankTransferTool(source_account_type, target_account_type, amount) Faulty Graph: source_account_type: "checking" , target_account_type: "savings" , amount: "$150" Error: Extracts amount as string with currency symbol instead of numeric value 150.
Type Coercion Failure	Inability to properly convert data types between tool outputs and subsequent tool inputs.	Tool chain: GetUserID(name) $\rightarrow$ user_id: 7 $\rightarrow$ FetchDetails(id) Faulty Graph Connection: Passes user_id: 7 (integer) as id: "7" (string) Error: Type mismatch between integer output and string-requiring input.
Conditional Branch Collapse	Flattening conditional execution paths into a single linear parameter graph.	User: "If temperature > 30°C, turn on AC; else, open windows." Faulty Graph: Contains parameters for both ACControlTool and WindowControlTool Error: Cannot simultaneously execute both branches; should generate conditional graph structure.
Default Value Misapplication	Incorrectly assigning or omitting default parameter values.	Tool: ImageProcessorTool(image, format="JPEG", quality=85, resize=None) User: "Process this image to high quality." Faulty Graph: image: <data>, quality: "high" Error: Should convert qualitative "high" to quantitative quality: 95 and preserve defaults for format and resize.
Structural Serialization Failure	Inability to correctly serialize the parameter dependency graph into the required formal schema (e.g., JSON, YAML), leading to syntactically invalid or malformed output that cannot be parsed by the execution engine.	{ "Parameter Graph": { "user_task_parameters": { "cities": ["San Francisco" "New York"], *// Error 1: Missing comma in array* "date": "2025-11-21" *// Error 2: Missing 'airport' field (required for downstream tools)* }, "tool_parameters": { "Weather Query": { "input": ["cities", "date"], "output": ["weather_data"], "depends_on": ["user_task"] } *// Error 3: Missing comma between tool definitions* "Route Planner": { "input": ["airport", "hotel_address"], *// Error 4: 'airport' not defined above* "output": ["drive_time"] *// Error 5: Missing 'depends_on' field* } } } *// Error 6: Missing final closing brace for root*

Table 8: Examples of different error types in tool calling stage.

Error Type	Definition	Example
Temporal Conflict Blindness	Failure to detect and resolve conflicts between time-sensitive tool results and the model's own temporal knowledge cutoff.	Tool Result (R-Time Variant): "The current U.S. President is Donald Trump." Model's Internal Knowledge Cutoff: Information up to October 2023. Erroneous Response: "According to the latest information, Donald Trump is currently serving as the U.S. President." Error: The LLM does not recognize that the tool's "current" claim conflicts with its internal knowledge (Biden was president as of its last update). It fails to flag the temporal inconsistency or suggest verification from updated sources.
Domain Terminology Oversimplification	Accepting and regurgitating informal or layperson tool outputs without converting them into appropriate domain-specific terminology, or failing to recognize when such conversion is needed.	Tool Result (R-Prof Variant): "The patient's heart sometimes beats too fast and sometimes too slow." Expected Professional Term: "Cardiac arrhythmia" or "atrial fibrillation with variable rate." Erroneous Response: "The tool indicates the patient's heart beats irregularly—sometimes fast, sometimes slow." Error: The LLM does not elevate the informal description to the standard medical terminology, missing an opportunity to demonstrate professional integration of domain knowledge.
Ambiguity Propagation	Failing to detect that a tool result is semantically ambiguous or comes from a mismatched tool, and propagating that ambiguity without clarification or qualification.	Tool Context: A user asks for legal document analysis. Tool Result (R-Ambig Variant): Provides a literary sentiment analysis output (e.g., "This text expresses strong emotional conflict"). Erroneous Response: "The legal document shows strong emotional conflict, which may indicate contentious clauses." Error: The LLM does not recognize that the tool result is irrelevant to the legal domain and blindly incorporates it, producing a misleading response.
Source Authority Over-reliance	Uncritically accepting tool results as authoritative, even when they conflict with the model's reliable internal knowledge, without noting the discrepancy or advising verification.	Tool Result: "The population of City X is 2 million." Model's Internal Knowledge (from reputable census data): City X population is 1.5 million. Erroneous Response: "The population of City X is 2 million." Error: The LLM suppresses its own accurate knowledge in favor of the tool output, without questioning or highlighting the conflict.

Table 9: Examples of different error types in response generation stage.