TA-CALLBENCH: BENCHMARKING NESTED TOOL CALLING FOR CONDITIONAL TRIGGER–ACTION TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) can select tools and generate arguments from user requests, supporting immediate queries, software operations, and task automation. However, existing benchmarks primarily evaluate whether models generate correct function calls for the current request, and rarely examine their ability to generate nested tool calls for conditional trigger–action tasks, where an operation must be executed only after a specified event or state occurs. Because such a task combines triggering conditions with executable Actions, its representation requires an outer call that organizes these components. To evaluate this ability, we introduce TA-CallBench, which requires a model to select inner tools (the individual Trigger and Action functions) and outer tools (management tools that organize inner calls into complete conditional tasks) from the available tool set and generate the required nested tool calls. TA-CallBench provides thousands of bilingual dialogues across six application scenarios and 12 task types, including single-task creation, multi-task creation, multi-turn modification, and fine-grained component editing, which together require a model to identify task boundaries and organize components within nested calls. We construct the data through Trigger-anchored tool-combination sampling, LLM-based generation, structural validation, and combined model and human review. The evaluation centers on Tool-Call Accuracy (TCA), which checks whether a model correctly generates the Trigger and Action calls and their arguments. All evaluated models achieve TCA below 55% on TA-CallBench. For models evaluated on both benchmarks, their TA-CallBench TCA is substantially lower than their performance on the Berkeley Function Calling Leaderboard (BFCL)*. This gap indicates that current LLMs still struggle to reliably generate such nested calls, highlighting the need for dedicated evaluation of this capability.

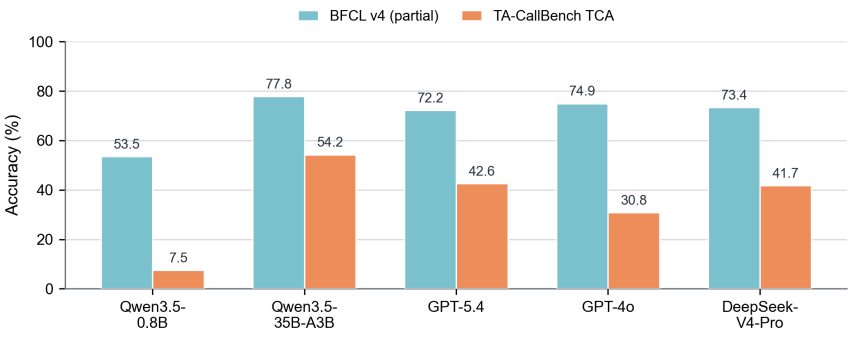


Figure 1: Comparison between BFCL scores and Tool-Call Accuracy on TA-CallBench in Chinese under the System+Guidance setting.

*Throughout this paper, BFCL scores are computed as the aggregate over the Non-Live, Live, and Multi-Turn subsets of BFCL v4.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Large language models (LLMs) are evolving from question-answering systems into agents that can invoke external tools to complete user tasks. Given a request, a model selects an appropriate function and generates arguments according to its schema, enabling information retrieval, software operations, and multi-step execution. Prior work has studied tool-use learning and tool-document retrieval (Schick et al., 2023; Patil et al., 2024) and has introduced benchmarks along three main directions: tool selection and argument generation for individual function calls (Li et al., 2023; Qin et al., 2024; Chen et al., 2024; Patil et al., 2025), stateful tool use in multi-turn interactions (Farn & Shin, 2023; Yao et al., 2025; Lu et al., 2025), and multi-step task execution in realistic environments (Wang et al., 2025; Guo et al., 2026).

Despite their different settings, these benchmarks generally follow an independent tool-calling paradigm, in which each generated call is immediately executable. A model decomposes a request into one or more directly executable function calls and produces them sequentially or in parallel. Even when a multi-turn or multi-step task contains tool dependencies, later calls are typically generated from the execution results of earlier calls. The prediction target therefore remains an individual call, a call sequence, or a final task outcome, so the model only reacts to the current request by determining the calls to execute now.

Real-world tasks can also contain explicit trigger conditions (Ur et al., 2016). Although agent applications (OpenClaw, 2025; Research, 2025) for scheduled automation have emerged, their automation primarily follows predefined schedules rather than events or environmental states, so a user must know the exact execution time in advance and tasks that depend on dynamically occurring states remain out of reach. To assist users more intelligently, agents must further support perceptual trigger conditions grounded in events and environmental states. For example, health monitoring and environmental sensing require an operation to be executed only after a specified event occurs or a state is satisfied. Such condition-triggered tasks are especially frequent on mobile devices, where a phone assistant can turn off all the lights at home once the user leaves. These tasks require a structured output that jointly represents *Triggers*, *Actions*, and the logical relations among *Triggers*. Independent tool calling can produce individual, parallel, or sequential calls, but it does not explicitly encode how these components are grouped into tasks. A representation that preserves both component grouping and *Trigger* logic is therefore essential.

To address this gap, we introduce a Trigger–Action-based nested tool-calling paradigm, where nesting serves to organize Trigger and Action components into a task rather than to pass outputs between dependent calls. A prediction for such tasks must jointly encode the definitions of condition and action components, their grouping into tasks, and the logical relations among trigger conditions. The paradigm therefore adopts a hierarchical structure by design, decoupling verifiable component definitions from task-level organization. Inner tools consist of *Trigger* functions that describe the conditions to monitor and *Action* functions that describe the operations to execute, which preserves verifiable component definitions. Outer tools organize the inner calls and their relations, which preserves task boundaries and *Trigger* logic. A model therefore organizes its prediction hierarchically, first selecting the inner *Trigger* and *Action* tools required by each conditional task and filling in their required arguments, and then placing the complete inner calls into the argument slots of the corresponding outer tool. Each outer call corresponds to one complete conditional task, and independent tasks map to separate outer calls in request order. This representation preserves task boundaries, component ownership, and modification scope in a unified structured format.

Building on this paradigm, we construct TA-CallBench, a benchmark for conditional nested tool calling. The benchmark contains 3,022 Chinese dialogues and 2,940 English dialogues across 12 task types, covering single-turn and multi-turn interactions, independent task separation, and targeted changes to *Triggers*, *Actions*, and *Trigger* logic. The creation and composition types require a model to determine task boundaries and group components within one task, whereas the nine modification types require it to locate the affected component and produce the complete updated call. Its inner tool catalog contains 20 *Triggers* and 40 *Actions* spanning six application scenarios, and its outer layer contains separate tools for task creation and task update. We construct the data through tool-combination sampling, LLM-based generation, structural validation, and combined model and human review. Our contributions are threefold.

- We introduce a nested tool-calling paradigm for conditional trigger–action tasks, whose hierarchical structure decouples verifiable component definitions from task-level organization. Outer tools organize inner *Trigger* and *Action* calls and their logical relations.
- We construct TA-CallBench, the first benchmark that evaluates nested tool calling for conditional trigger–action tasks. The benchmark spans six application scenarios, 20 *Trigger* and 40 *Action* tools, and 12 creation, composition, and modification task types, with both Chinese and English versions.
- We establish an evaluation protocol centered on complete tool-call correctness and systematically evaluate open-source and proprietary LLMs, revealing that strong standard function-calling performance does not transfer to conditional nested tool calling.

2 Related Work

Tool use by large language models. Existing work improves the tool-use capability of LLMs through tool-use training and tool-document retrieval. Toolformer uses self-supervised learning to teach a model when to call tools, which tool to select, and how to use returned results (Schick et al., 2023). Gorilla combines Application Programming Interface (API) document retrieval with model training to improve call accuracy over a large candidate set and reduce calls to invalid APIs (Patil et al., 2024). These studies primarily examine how LLMs learn tool functionality and generate correct tool calls in standard function-calling settings, and their reported progress is itself measured by such benchmarks.

Tool-calling benchmarks and call dependencies. Existing benchmarks cover tool selection and argument generation, parallel calls, multi-turn interactions, and multi-step execution in external environments (Li et al., 2023; Qin et al., 2024; Chen et al., 2024; Patil et al., 2025; Farn & Shin, 2023; Yao et al., 2025; Lu et al., 2025; Guo et al., 2026). Among them, GraphQLRestBench evaluates sequential calls with variable mappings between functions (Saha et al., 2024), and NESTFUL and NesTools further study nested call sequences in which the output of an earlier tool becomes an argument of a later tool (Basu et al., 2025; Han et al., 2025). These benchmarks focus on data dependencies and execution order across calls, and none of them evaluates calls defined by explicit trigger conditions, although the creation of such condition-triggered automations has been studied in trigger-action programming (Yusuf et al., 2022; Gao et al., 2024; Gallo et al., 2024; Badalov & Yoon, 2026). In contrast, TA-CallBench studies structured nesting for conditional trigger–action tasks. An outer tool uses dedicated argument fields to organize complete *Trigger* and *Action* calls and their logical relations, allowing TA-CallBench to evaluate whether a model correctly handles component assignment, task boundaries, and multi-turn modification.

3 Benchmark Construction

This section describes the construction of TA-CallBench, which evaluates whether LLMs can generate nested tool calls for conditional trigger–action tasks. Figure 2 overviews the entire workflow, from tool-catalog design and dialogue synthesis to evaluation.

3.1 Nested Tool-Calling Paradigm

Conditional trigger–action tasks require a model to organize one or more triggering conditions, executable Actions, and their logical relations in a single structured output. The proposed paradigm represents concrete Triggers and Actions as inner tools and uses outer tools to organize these inner calls, decoupling verifiable component definitions from task-level organization.

Inner and outer tools. The inner tool catalog consists of 20 manually designed Triggers and 40 manually designed Actions. Each schema specifies a tool name, a functional description, parameter names and types, required parameters, and allowed enumeration values. A Trigger describes an event or state to monitor, while an Action describes an operation to execute when the condition is satisfied. Each inner call contains a tool name and its arguments.

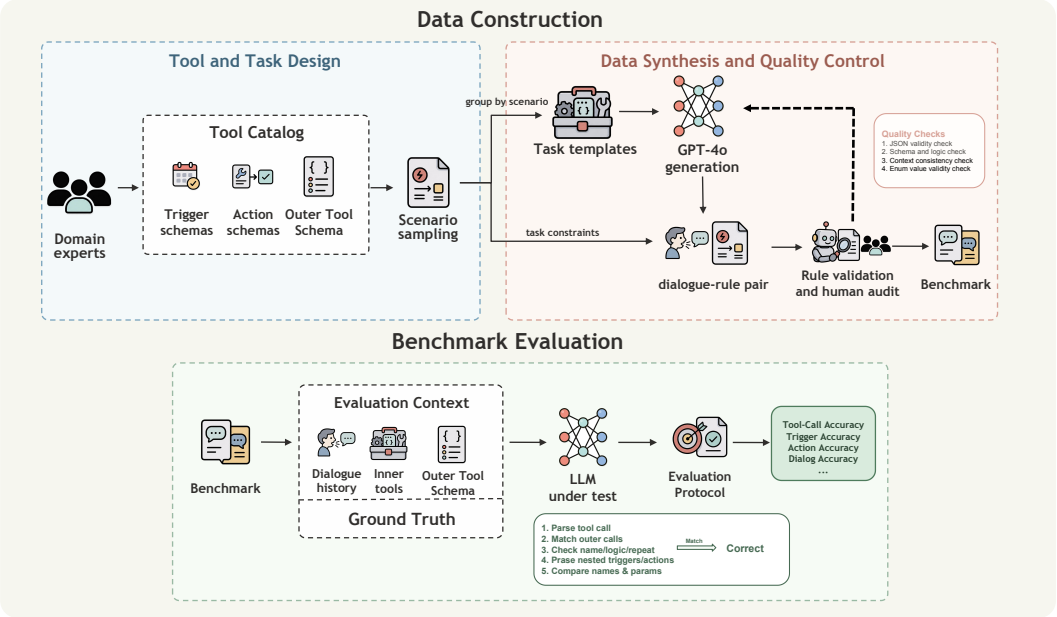


Figure 2: Overview of TA-CallBench construction and evaluation. Domain experts design the tool catalog, and GPT-4o generates dialogue–target pairs that undergo structural validation and human review. During evaluation, nested calls are compared with the reference target.

We define two outer tools, `create_trigger_task` and `update_trigger_task`, for creating and updating conditional tasks. Both outer tools require `triggers` and `actions`, whose values are inner call lists encoded as JavaScript Object Notation (JSON) strings. Encoding complete inner calls as JSON strings mirrors how structured values are passed in mainstream function-calling interfaces and agent tool protocols (Anthropic, 2024), so the nested representation can be evaluated without any custom execution framework. When an outer call contains multiple Triggers, `logic` specifies an `and` or `or` relation. The optional `repeat` argument is included only when the user explicitly requests repetition. The update tool can additionally use `workflow_id` to associate the call with an existing task. Each outer call corresponds to one complete conditional task, while its inner calls define the triggering conditions and executable Actions. This organization explicitly represents task boundaries, component ownership, and Trigger logic. Appendix A details the tool organization and representative schemas. The complete bilingual inner-tool schemas are distributed with the benchmark.

Dialogue format and prediction target. Each dialogue follows a general message format compatible with the OpenAI Chat Completions API and is stored as a JSON array containing `system`, `user`, `assistant`, and `tool` messages when applicable. A dialogue contains one or two user turns. For a second-turn request, the input history contains the reference tool call and tool response from the first turn, allowing the model to modify the existing task state. The model predicts only `assistant` messages that contain `tool_calls`.

Formally, given a dialogue history $\mathcal{D}$ and a set of tool schemas $\mathcal{S}$, the model predicts an ordered list of outer tool calls $\mathcal{C} = [c_1, \ldots, c_m]$. Each c_j invokes `create_trigger_task` or `update_trigger_task` and provides a complete inner call structure r_j in its arguments. After parsing the inner JSON strings, the structure can be written as

$$r = \{\texttt{triggers} : [\tau_1, \ldots, \tau_p],\ \texttt{actions} : [\alpha_1, \ldots, \alpha_q],\ \texttt{logic} : \ell,\ \texttt{repeat} : \rho\}.$$

Here, τ_i denotes a Trigger call with complete arguments and α_j denotes an Action call with complete arguments. When the structure contains multiple Triggers, ℓ denotes the optional `and`/`or` relation, and ρ denotes the optional repetition mode. The equation shows parsed lists for readability; in the

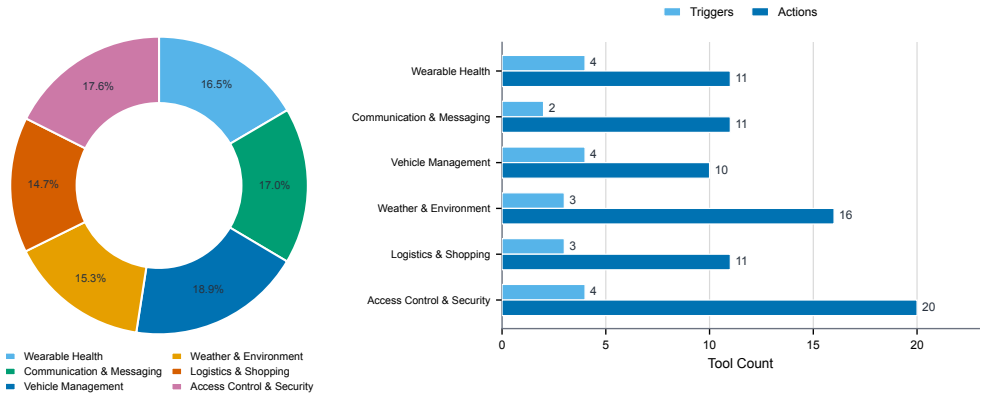


Figure 3: Scenario coverage of TA-CallBench. The donut chart shows the distribution of Chinese dialogues, and the bars show the number of Trigger and Action schemas associated with each scenario. An Action may be associated with multiple scenarios, so the per-scenario Action counts do not sum to 40.

actual output, `triggers` and `actions` must be serialized as JSON strings. The `workflow_id` in an update call associates the request with an existing task and is not part of the target call content.

The number of outer calls m is determined by the number of independent tasks in the user request, rather than by the number of Triggers or Actions. A task with multiple Triggers or Actions still corresponds to one outer call. If a request contains multiple independent tasks, the model must generate one outer call for each task in the same order as the request. For a modification request, the prediction target is the complete updated call, not the modification operation or only the changed field. TA-CallBench evaluates call generation and does not execute Triggers or Actions at runtime.

3.2 Data Construction Pipeline

Tool design and pairing. We begin with six application scenarios: wearable health, communication and messaging, vehicle management, weather and environment, logistics and shopping, and access control and security. Figure 3 shows the dialogue distribution across these scenarios together with the number of Trigger and Action schemas associated with each. For each scenario, we design Triggers and Actions that cover conditional monitoring and executable operations. We then use each Trigger as an anchor and select semantically compatible Actions to form a scenario-level candidate set. Actions are not restricted to a single scenario and can be paired with Triggers from different scenarios. This design filters implausible combinations while preserving diversity for subsequent dialogue synthesis.

Scenario-constrained dialogue synthesis. For each dialogue, we randomly sample one or more Triggers and Actions from a scenario-level candidate set and configure their argument values, Trigger logic, repetition mode, and task type. We design a generation prompt for each of the 12 task types. The prompt includes the selected schemas, argument values, component counts, logical relations, and required dialogue structure. GPT-4o (OpenAI, 2024) generates a natural-language dialogue and its target tool calls under these constraints, where the user request must contain all information required to construct the target call. If one request contains multiple independent tasks, each task maps to a separate outer call, and the dialogue and target calls preserve the same order. For modification tasks, the first turn and its target call establish the context; the second user request specifies only the component to change, while the target remains the complete updated call.

Quality control. Every generated sample undergoes two automated validation passes. The first pass checks the JSON structure, inner and outer tool names, parameter names and types, required parameters, enumeration values, Trigger logic, and dialogue order constraints such as creation before modification. We then use DeepSeek-V4-Flash (DeepSeek-AI, 2026) to polish the natural-language request and manually inspect linguistic fluency and consistency between user intent and the target

calls. The second pass repeats structural validation and checks consistency among the dialogue, application scenario, and target calls. After the two validation passes, samples that fail are manually corrected or discarded. In total, 86% of the generated data pass validation, and fewer than 10% are discarded.

The English data are produced by translating validated Chinese dialogues while preserving the task intent and Trigger–Action pairing. We check each translation for semantic completeness, fluency, tool selection, argument values, and call structure. Translations that fail these checks are discarded, so the two language versions are not strictly aligned one to one.

Task types and dataset size. TA-CallBench contains three creation task types and nine modification task types. Together, they cover single-turn and multi-turn interactions, independent task separation, and targeted changes to Triggers, Actions, and Trigger logic. Appendix A.5 provides the complete task-type definitions together with the operation and dialogue count of each type. Treating every `assistant` turn with `tool_calls` as one evaluation instance, the 3,022 Chinese and 2,940 English dialogues yield 5,258 and 5,138 evaluation instances, respectively.

4 EVALUATION

This section defines the evaluation protocol and metrics of TA-CallBench. The protocol specifies model inputs, prediction targets, and structural matching for nested calls. The metrics measure the correctness of inner Trigger and Action calls and of the complete outer call.

4.1 EVALUATION PROTOCOL

Evaluation setup. Each `assistant` message containing `tool_calls` forms one evaluation instance. The input comprises the complete dialogue history before the prediction point, the schemas of all 20 Trigger tools and 40 Action tools, and the two outer management tools, `create_trigger_task` and `update_trigger_task`, from which the model must select the relevant Triggers and Actions. A single-turn dialogue contains one prediction point. In a multi-turn dialogue, the creation and modification turns are evaluated separately. For a modification turn, the history also contains the reference tool call and response from the first turn, so every model starts from the same correct task state.

Output and matching. The model generates one or more outer `tool_calls` for the current user request. Each call invokes the creation or update tool and provides complete inner calls through the `triggers` and `actions` arguments. A call with multiple Triggers must also provide their logical relation. The evaluator considers only `tool_calls`, ignoring reasoning traces, natural-language responses, and tool-call identifiers, and it does not execute any Trigger or Action.

Scoring uses deterministic structural matching. The evaluator first parses the outer arguments and then parses the Trigger and Action calls encoded as JSON strings. It converts both levels into a canonical JSON representation. Matching ignores whitespace, escaping choices, and object-key order, but preserves the order of outer calls and the order of Trigger and Action lists. The optional `logic` and `repeat` arguments must be omitted when they are not specified by the user; extra values affect matching. An output that cannot be parsed at either level is incorrect, and the evaluator does not repair model predictions. Producing parseable argument values is itself part of the evaluated capability, since real function-calling interfaces likewise require models to emit valid JSON; parse failures are therefore recorded as a separate error category and reported alongside the structural errors, so that the two sources of incorrectness can be distinguished.

4.2 EVALUATION METRICS

Primary metric. The primary metric is Tool-Call Accuracy (TCA). Let the test set contain D dialogues, and let $\mathcal{M}_d$ be the set of `assistant` turns with reference `tool_calls` in dialogue d. For turn $i \in \mathcal{M}_d$, let $\mathcal{C}_{d,i}$ and $\widehat{\mathcal{C}}_{d,i}$ denote the ordered lists of reference and predicted outer calls. Under the structural normalization above, the turn-level correctness is

$$s_{d,i} = \mathbf{1}\left[\widehat{\mathcal{C}}_{d,i} \equiv \mathcal{C}_{d,i}\right].$$

Table 1: Tool-Call Accuracy (%) under the System and Tool List configurations.

Type	Model	System		Tool List	
		CN	EN	CN	EN
Open	Qwen3.5-0.8B	6.60	8.12	6.79	7.01
	Qwen3.5-4B	15.90	17.21	16.55	16.97
	Qwen3.5-9B	19.76	18.82	19.80	18.82
	Qwen3.5-27B	21.39	20.37	20.56	20.04
	Qwen3.5-35B-A3B	21.76	21.41	21.80	21.12
Proprietary	GPT-5.4	23.41	24.99	24.88	26.02
	GPT-4o	7.30	7.47	9.13	10.59
	DeepSeek-V4-Pro	19.56	18.29	19.91	18.55

The relation $\equiv$ requires the two lists to contain the same number of calls and requires every aligned call to match exactly in outer tool name, `logic`, `repeat`, Trigger list, and Action list. Trigger and Action order, names, and arguments all participate in matching. With $N = \sum_{d=1}^{D} |\mathcal{M}_d|$ evaluation turns, TCA is

$$\mathrm{TCA} = \frac{1}{N} \sum_{d=1}^{D} \sum_{i \in \mathcal{M}_d} s_{d,i}.$$

An omitted or redundant outer call, or an error in any nested component, makes the current turn incorrect. All primary results and grouped comparisons use TCA.

Auxiliary analysis. Dialogue Accuracy (DA) measures whether a model predicts every evaluated turn in a dialogue correctly:

$$\mathrm{DA} = \frac{1}{D} \sum_{d=1}^{D} \mathbf{1}\left[\sum_{i \in \mathcal{M}_d} s_{d,i} = |\mathcal{M}_d| \right].$$

A multi-turn dialogue is correct only if both its creation and modification turns are correct. We use DA only as an auxiliary metric and not for primary model comparisons. We also report Trigger Accuracy and Action Accuracy. Each requires the corresponding inner list to match in component count, order, tool name, and arguments. For calls with multiple Triggers, Logic Accuracy checks the predicted `and`/`or` relation. These component metrics help identify whether a TCA error originates from triggering conditions, executable Actions, or their logical relation.

The error analysis records Trigger–Action mismatches, call-count errors, outer-tool name errors, `repeat` and `logic` errors, name or argument errors in inner tools, and parse failures at either the outer or inner level. Inner Trigger and Action name or argument errors are counted per inner call; all other categories are counted per `assistant` turn. A prediction may contain multiple error types, so the category counts are not mutually exclusive. Across all evaluated models and both languages, no parse failure is observed at either the outer or inner level, indicating that the errors arise from component selection, argument generation, and task organization rather than from JSON formatting. An additional diagnostic concerns schema-format deviations, where a model produces the correct tool and argument values but violates the schema wrapper, for example by using a `type` key instead of `name`. This category affects a few models substantially, and Appendix D.5 quantifies its extent and discusses why TA-CallBench retains the strict interpretation.

5 Experiments

We evaluate eight LLMs on the Chinese and English versions of TA-CallBench and use BFCL (Patil et al., 2025) as an external reference for the standard function-calling ability of five models, namely Qwen3.5-0.8B, Qwen3.5-35B-A3B, GPT-5.4, GPT-4o, and DeepSeek-V4-Pro. BFCL scores are the aggregate over the Non-Live, Live, and Multi-Turn subsets of BFCL v4, as defined in the abstract footnote. The evaluated systems include Qwen3.5-0.8B, Qwen3.5-4B, Qwen3.5-9B, Qwen3.5-27B, and Qwen3.5-35B-A3B from the Qwen family (Qwen Team, 2026); GPT-5.4 and GPT-4o from

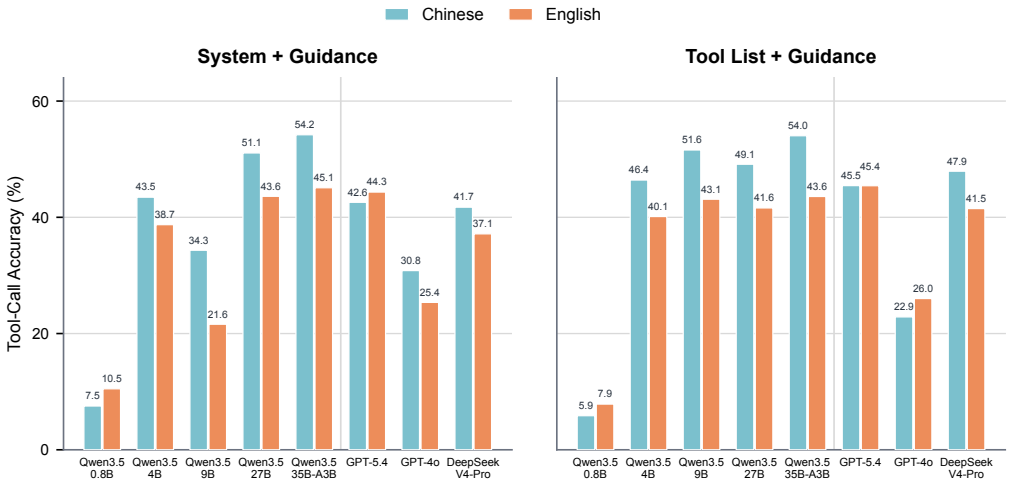


Figure 4: Tool-Call Accuracy (%) under System+Guidance and Tool List+Guidance. Each model is reported separately on the Chinese and English versions.

OpenAI (OpenAI, 2026; 2024); and DeepSeek-V4-Pro from DeepSeek (DeepSeek-AI, 2026). Appendix E details the BFCL reference results.

We use four interface configurations. **System** places the Trigger and Action definitions in the system prompt, whereas **Tool List** places them in the API tool list together with the outer tool definitions. **System+Guidance** augments System with an explicit description of the nested call structure. **Tool List+Guidance** places the same structural description in the `description` field of the outer tools. These configurations examine the effects of inner-schema placement and explicit nested-structure guidance. Table 1 reports System and Tool List, while Figure 4 reports the corresponding Guidance configurations.

5.1 Main Results

Standard function-calling ability does not guarantee conditional nested tool calling. All individual TCA results for the eight models with complete results under Guidance are below 55%, with a maximum of 54.20% in Figure 4. As an external reference, the five shared models obtain BFCL scores between 53.50% and 77.80%, whereas their TCA values on the Chinese TA-CallBench under System+Guidance range from 7.53% to 54.20% in Figure 1. GPT-4o, for example, obtains 74.92% on BFCL but 30.83% on TA-CallBench. A high standard function-calling score therefore does not directly reflect the ability to organize an outer tool, nested Trigger and Action calls, and their logical relations in one prediction.

Model performance is more sensitive to explicit nested-structure guidance. Across the eight models with complete results, the two configurations without Guidance have an average TCA of 17.22%, compared with 37.02% for the two configurations with Guidance. Guidance produces a higher result in 31 of the 32 model–language–placement combinations. To separate this effect from schema placement, we compare configurations under the same Guidance condition. Macro-averaged across the eight models and two languages, System and Tool List obtain 17.02% and 17.41%, respectively, a difference of 0.39 percentage points. System+Guidance and Tool List+Guidance obtain 35.76% and 38.28%, respectively, a difference of 2.52 points. After aggregating the two languages per model, four models favor System+Guidance and four favor Tool List+Guidance. Model performance is thus more sensitive to whether the nested organization is stated explicitly, while the preferred schema placement depends on the model. Appendix D.1 reports the complete per-model results under all four configurations.

Table 2: TCA (%) by task group with and without structural Guidance, macro-averaged over the eight models with complete results, two languages, and two schema placements.

Task group	No Guid.	Guid.
Single-turn single-task	2.14	38.39
Single-turn multi-task	0.03	21.17
Multi-turn single-task	10.77	30.37
Multi-turn modification	21.01	38.81

Table 3: Trigger and Action Accuracy (%) under the two Guidance configurations, macro-averaged over eight models with complete results.

Metric	System+Guid.		Tool List+Guid.	
	CN	EN	CN	EN
Trigger Acc.	79.32	73.78	81.32	77.71
Action Acc.	78.12	72.16	78.63	70.74

Language differences depend on the model and task type. Across the two Guidance configurations, the average TCA is 39.31% for Chinese and 34.73% for English. The Chinese–English gaps of Qwen3.5-9B and Qwen3.5-35B-A3B are 10.60 and 9.79 percentage points, respectively, while GPT-5.4 performs 0.87 points better in English. At the task-group level, the Chinese advantage is 9.85 points on single-turn multi-task but 3.73 points on multi-turn modification, as shown in Appendix D.2. We therefore report the two languages separately rather than treating their overall mean difference as a uniform language effect.

5.2 Task Group and Turn Analysis

Single-turn multi-task requests are the most challenging group. Table 2 combines the nine fine-grained modification types into a multi-turn modification group and compares four task groups with and without structural Guidance. Without Guidance, single-turn single-task and single-turn multi-task obtain only 2.14% and 0.03% average TCA, respectively, whereas multi-turn modification obtains 21.01%. One possible explanation is that a modification turn can reuse the existing call as a structural reference, which reduces the need to organize the nested structure from scratch. With Guidance, single-turn single-task rises to 38.39%, but single-turn multi-task remains at 21.17%, below multi-turn single-task at 30.37% and multi-turn modification at 38.81%. Single-turn multi-task is the weakest group in 24 of the 32 valid model–language–Guidance combinations. Because these requests require several independent tasks to be separated within one user turn, identifying the number of outer calls, task boundaries, and inner-component ownership remains a central challenge.

5.3 Error Analysis

Correct local Trigger and Action calls do not ensure a correct complete call. Table 3 shows that Trigger Accuracy and Action Accuracy average 78.03% and 74.91% across the two Guidance configurations and two languages, while the corresponding TCA is only 37.02%. The error records further show that argument errors occur more often than name errors for both Triggers and Actions under System+Guidance, as detailed in Appendix D.4. TCA requires the outer call and every nested Trigger and Action to be correct simultaneously, so local errors accumulate under complete matching and create a large gap between component accuracy and complete-call accuracy.

6 Conclusion

This work addresses the evaluation gap for conditional nested tool calling through a nested tool-calling paradigm, the TA-CallBench benchmark, and a TCA-centered evaluation protocol that checks whether an outer call and all of its nested components are generated correctly. Across eight LLMs, all evaluated TCA results remain below 55%, and the five models evaluated on both benchmarks obtain lower results on TA-CallBench than on BFCL, showing that standard function-calling capability does not transfer to this setting. The persistent weakness on single-turn multi-task requests and the large gains from explicit structural guidance suggest that training with structure-aware supervision is a promising direction. Together, the proposed paradigm, benchmark, and evaluation protocol provide a unified empirical foundation for developing LLMs that handle conditional nested tool calling reliably.

AI USE STATEMENT

In this work, we used generative AI tools to synthesize and polish benchmark dialogues, translate validated Chinese dialogues into English, and assist with language editing of the manuscript. Schema-based validators and human reviewers checked the AI-generated benchmark data, and the authors reviewed all AI-assisted manuscript text. We take responsibility for the final content of this work, including text, claims, data, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

TA-CallBench consists of synthetic dialogues and does not contain data collected from human subjects or real users. The benchmark evaluates the generation of structured calls and does not execute the represented Triggers or Actions. Because several scenarios include health monitoring, access control, and device operations, benchmark performance should not be interpreted as authorization to deploy model-generated calls in safety-critical environments without independent validation and user confirmation.

REPRODUCIBILITY STATEMENT

Section 3 describes the tool organization, dialogue format, prediction target, construction pipeline, and dataset composition. Section 4 specifies deterministic matching and all reported metrics, while Section 5 defines the evaluated configurations and reports the main results. The benchmark release will include the complete tool schemas, generation prompts, structured data, and evaluation code.

REFERENCES

Anthropic. Model context protocol specification. https://modelcontextprotocol.io/specification/, 2024. Accessed: 2026-07-13.

Khusrav Badalov and Young Yoon. FARM: Field-aware resolution model for intelligent trigger-action automation, 2026. URL https://arxiv.org/abs/2601.15687.

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, Xin Wang, Luis A. Lastras, and Pavan Kapanipathi. NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33538–33547, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1702. URL https://aclanthology.org/2025.emnlp-main.1702/.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9510–9529, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.515. URL https://aclanthology.org/2024.acl-long.515/.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Nicholas Farn and Richard Shin. Tooltalk: Evaluating tool-usage in a conversational setting, 2023. URL https://arxiv.org/abs/2311.10775.

Simone Gallo, Fabio Paternò, and Alessio Malizia. A conversational agent for creating automations exploiting large language models. *Personal and Ubiquitous Computing*, 28:931–946, 2024. doi: 10.1007/s00779-024-01825-5.

Yi Gao, Kaijie Xiao, Fu Li, Weifeng Xu, Jiaming Huang, and Wei Dong. ChatIoT: Zero-code generation of trigger-action based IoT programs. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 8(3):103:1–103:29, 2024. doi: 10.1145/3678585.

Zikang Guo, Benfeng Xu, Chiwei Zhu, Wentao Hong, Xiaorui Wang, and Zhendong Mao. MCP-AgentBench: Evaluating real-world language agent performance with MCP-mediated tools. In *Proceedings of the 40th AAAI Conference on Artificial Intelligence*, volume 40, pp. 30888–30896, 2026. doi: 10.1609/aaai.v40i37.40347.

Han Han, Tong Zhu, Xiang Zhang, MengSong Wu, Xiong Hao, and Wenliang Chen. NesTools: A dataset for evaluating nested tool learning abilities of large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 9824–9844, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.657/.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.187. URL https://aclanthology.org/2023.emnlp-main.187/.

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. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, Albuquerque, New Mexico, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

OpenAI. GPT-4o system card. https://openai.com/index/gpt-4o-system-card/, 2024. Accessed: 2026-07-28.

OpenAI. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/, 2026. Accessed: 2026-07-28.

OpenClaw. openclaw: Your own personal ai assistant. https://github.com/openclaw/openclaw, 2025. Accessed: 2026-09-22.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 126544–126565. Curran Associates, Inc., 2024. doi: 10.52202/079017-4020. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

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. ToolLLM: Facilitating large language models to master 16,000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Qwen Team. Qwen3.5 model collection. https://huggingface.co/collections/Qwen/qwen35, 2026. Accessed: 2026-07-28.

Nous Research. hermes-agent: The agent that grows with you. https://github.com/nousresearch/hermes-agent, 2025. Accessed: 2026-09-22.

Avirup Saha, Lakshmi Mandal, Balaji Ganesan, Sambit Ghosh, Renuka Sindhgatta, Carlos Eberhardt, Dan Debrunner, and Sameep Mehta. Sequential API function calling using GraphQL schema. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 19452–19458, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1083. URL https://aclanthology.org/2024.emnlp-main.1083/.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L. Littman. Trigger-action programming in the wild: An analysis of 200,000 IFTTT recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, and Eugene Siow. MCP-Bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers, 2025. URL https://arxiv.org/abs/2508.20453.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b126cc38b8638e07bef37e7b2bb72bf-Abstract-Conference.html.

Imam Nur Bani Yusuf, Diyanah Abdul Jamal, Lingxiao Jiang, and David Lo. RecipeGen++: An automated trigger action programs generator. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1672–1676, 2022. doi: 10.1145/3540250.3558913.

APPENDIX

A TOOL AND TASK SPECIFICATIONS

This section specifies the nested calling structure, the shared schema of the inner tools, catalog-level statistics, a structure-preserving example, and the precise boundaries of the 12 task types in TA-CallBench. The presentation retains the structural information required to understand the task and evaluation, while the complete machine-readable catalogs accompany the benchmark release.

A.1 ABSTRACT NESTED-CALLING FORMAT

TA-CallBench organizes a prediction as an ordered list of outer calls. Each outer call represents one conditional task and contains two ordered inner-call lists: a Trigger list $\mathcal{T} = [\tau_1, \ldots, \tau_p]$ and an Action list $\mathcal{A} = [\alpha_1, \ldots, \alpha_q]$. A call can be represented abstractly as

$$o = (u, \mathcal{T}, \mathcal{A}, \ell, \rho, w),$$

where $u \in \{\text{CREATE}, \text{UPDATE}\}$ denotes the outer operation, $\ell \in \{\texttt{and}, \texttt{or}\}$ is included only for multiple Triggers, ρ is an optional repetition mode, and w is an optional identifier available to the update operation. The Trigger and Action lists are serialized within the concrete interface. The notation below abstracts away its field names and serialization details while preserving the structural relations evaluated by TA-CallBench.

```
{
  "<OUTER_OPERATION>": "<CREATE_OR_UPDATE>",
  "<CONDITION_COMPONENTS>": ["<C_1>", "...", "<C_p>"],
  "<EXECUTION_COMPONENTS>": ["<E_1>", "...", "<E_q>"],
  "<CONDITION_RELATION>": "<OPTIONAL_RELATION>",
  "<REPETITION_CONTROL>": "<OPTIONAL_MODE>",
  "<UPDATE_REFERENCE>": "<UPDATE_ONLY>"
}
```

The number of outer calls is determined by the number of independent tasks in the request, not by the number of inner calls. Multiple Triggers or Actions that belong to the same task remain in one outer call, whereas independent condition–action pairs map to separate outer calls in request order. For an update request, the prediction contains the complete updated Trigger and Action lists rather than only the changed component.

The two configurations without Guidance and the two Guidance configurations use the same operation set, argument types, required fields, and enumeration constraints. Without Guidance, the outer arguments supply only their functional descriptions. Guidance additionally states that `triggers` and `actions` contain serialized complete inner calls, that `logic` applies only to multiple Triggers, and that `repeat` is included only when explicitly requested. Guidance changes only the structural descriptions of the outer-tool fields; it does not provide in-context demonstrations or reference tool calls.

A.2 ABSTRACT INNER-TOOL SCHEMA

All inner tools follow a common schema organization. A Trigger describes an event or state to monitor, whereas an Action describes an operation to execute. Each definition provides a tool identifier, a functional description, typed arguments, mandatory arguments, and value constraints when applicable. Abstracting from the concrete schema syntax, both catalog types follow the component specification below.

```
{
  "<COMPONENT_IDENTIFIER>": "<C_i_OR_E_j>",
  "<SEMANTIC_ROLE>": "<MONITORED_CONDITION_OR_OPERATION>",
  "<ARGUMENT_SPECIFICATIONS>": [
    {
      "<ARGUMENT_IDENTIFIER>": "<X_k>",
      "<VALUE_TYPE>": "<TYPE>",
```

```
      "<VALUE_CONSTRAINT>": "<OPTIONAL_CONSTRAINT>"
    }
  ],
  "<MANDATORY_ARGUMENTS>": ["<X_k>", "..."]
}
```

Benchmark targets store inner argument objects under `parameters`. Before structural matching, the evaluator also accepts `arguments` as an equivalent alias. Tool names, inner-call order, required arguments, argument values, and applicable enumeration values all participate in evaluation.

A.3 Inner-Tool Catalog Statistics

Table 4 summarizes the structural complexity of the inner catalog without enumerating individual interfaces. The Chinese and English catalogs share the same identifiers, parameter structures, and constraints; only their natural-language descriptions differ.

Trigger schemas contain between one and five parameters, and Action schemas contain between one and three. Across the Trigger catalog, the 54 parameters comprise 36 strings, 11 integers, 5 real-valued numbers, 1 Boolean, and 1 array. Across the Action catalog, the 61 parameters comprise 53 strings, 2 integers, 4 real-valued numbers, and 2 Booleans. These schemas require models to resolve both discrete tool selection and heterogeneous argument constraints.

A.4 Structure-Preserving Nested-Call Example

Consider the request "If monitor A becomes active and metric B rises above 75, send a status alert. Separately, when monitor C reports ready, record a note." The first condition contains two Triggers joined by `and`; the second condition defines an independent task. The following structure-preserving example represents the resulting prediction without reproducing concrete interface fields or serialization syntax. Here, C_i and E_j denote condition and execution components, respectively.

```
PREDICTION = [
  OUTER_1 {
    OUTER_OPERATION      = CREATE,
    CONDITION_COMPONENTS = [C_STATE(active), C_THRESHOLD(75, above)],
    CONDITION_RELATION   = AND,
    EXECUTION_COMPONENTS = [E_NOTIFY(status_alert)]
  },
  OUTER_2 {
    OUTER_OPERATION      = CREATE,
    CONDITION_COMPONENTS = [C_STATUS(ready)],
    EXECUTION_COMPONENTS = [E_RECORD(ready_status)]
  }
]
```

The first outer call groups two Triggers and one Action and therefore includes `logic`. The second outer call contains a separate Trigger–Action pair and omits `logic`. Both calls omit `repeat` because the request does not specify a repetition mode. This example illustrates that task boundaries are represented by outer calls, while component composition and Trigger logic are represented within each call.

Table 4: Structural statistics of the inner-tool catalog. Parameters and required fields are counted from the machine-readable schemas; Params/tool and Required/tool report averages.

Catalog	Tools	Parameters	Params/tool	Required/tool	Tools with enum
Trigger	20	54	2.70	2.10	20
Action	40	61	1.53	1.23	26

Table 5: Definitions of the 12 task types and dialogue counts by language.

Task type	Interaction form and target output	CN	EN
Task creation			
Single-turn single-task	One user turn produces one creation call containing one or more Triggers and Actions.	454	451
Single-turn multi-task	One user turn requests two independent tasks and produces two creation calls in request order.	234	233
Multi-turn single-task	Two user turns independently request one task each, and each turn produces one creation call.	249	249
Task modification			
Modify Trigger logic	The second turn changes only the `and`/`or` relation among existing Triggers.	285	276
Modify Trigger parameter	The second turn changes one argument of an existing Trigger.	276	269
Modify Action parameter	The second turn changes one argument of an existing Action.	216	182
Delete Trigger	The second turn removes one existing Trigger.	210	208
Delete Action	The second turn removes one existing Action.	213	211
Add Trigger	The second turn adds one new Trigger.	200	193
Add Action	The second turn adds one new Action.	204	194
Replace Trigger	The second turn replaces an existing Trigger with another Trigger.	283	282
Replace Action	The second turn replaces an existing Action with another Action.	198	192

A.5 Definitions of the 12 Task Types

Table 5 defines the 12 task types. The first three types create conditional tasks. In the multi-turn single-task type, each of the two user turns independently requests one task and produces one creation call. For the other nine types, the first turn creates a task and the second turn updates a specified component. Every modification target contains the complete updated outer call rather than only the changed field.

B Dataset Construction Details

This section supplements the construction pipeline in Section 3.2 with the specifications used to control dialogue synthesis and data validation. We present a representative abstract prompt template and task-specific constraints that preserve the operative generation requirements.

B.1 Representative Prompt Specifications

All task-specific prompts share the following specification. Angle-bracketed fields denote information injected for an individual sample, including the selected components, assigned argument values, and interaction structure. Additional constraints determine how these inputs are organized for each task type.

```
ROLE
Generate a natural-language dialogue and its corresponding
target nested tool calls from the supplied task specification.

INPUTS
- Application scenario: <SCENARIO>
- Trigger specifications: <TRIGGER_SPECIFICATIONS>
- Action specifications: <ACTION_SPECIFICATIONS>
- Assigned argument values: <ARGUMENT_ASSIGNMENTS>
- Task type: <TASK_TYPE>
- Interaction structure: <INTERACTION_CONSTRAINTS>
- Logical and repetition constraints: <CONTROL_CONSTRAINTS>

REQUIREMENTS
1. Express every selected Trigger, Action, and required
   argument in the user request without exposing interface
   identifiers or schema terminology.
```

```
2. Do not introduce any component or argument that is absent
   from the supplied inputs.
3. Group components by conditional task. Components belonging
   to one task share one outer call; independent tasks use
   separate outer calls.
4. Preserve the order of independent tasks and the order of
   components within each task.
5. Apply a logical relation only when multiple Triggers belong
   to the same task. Include repetition only when it is
   explicitly specified.
6. For a modification request, change only the requested
   component, preserve all unspecified content, and produce
   the complete updated target.
7. Use concise, fluent, and natural user language. Do not ask
   for information already supplied in the inputs.
8. Ensure that every target call satisfies the supplied type,
   required-argument, and value constraints.

OUTPUT
Return the dialogue and its target calls in the required
structured dialogue format.
```

The common specification controls semantic grounding, task boundaries, component ordering, and state preservation. Task-specific specifications further determine the number of dialogue turns, the number of outer calls, and the component affected by a modification.

Table 6 gives three representative task-specific specifications. Together, they expose the principal structural variations in TA-CallBench: composing components within one task, separating independent tasks, and preserving state during a targeted modification.

The remaining task-specific prompts follow the same organization. Creation prompts vary the number and grouping of components, while modification prompts specify whether the second turn changes, deletes, adds, or replaces a Trigger, an Action, a Trigger argument, an Action argument, or the Trigger relation. Every modification target represents the complete updated state.

B.2 Quality-Control and Translation Protocols

Initial structural validation. Generated samples are first checked against the inner- and outer-tool specifications. The validator parses the dialogue and both levels of tool-call arguments, verifies message and call structure, and checks tool identifiers, argument names and types, mandatory arguments, and enumeration values. It also checks the consistency between the number of Triggers and the presence of a logical relation, the expected creation or update operation for the task type, the order of independent calls, and dialogue-level constraints such as requiring creation before modification.

Language review and revalidation. After the initial validation, a language model polishes the natural-language dialogue without changing its intended components or argument values. Human reviewers inspect sampled dialogues for fluency and verify that the user intent agrees with the target calls. The resulting samples then pass a second structural validation, which repeats the schema

Table 6: Representative task-specific constraints combined with the common prompt specification.

Task type	Task-specific constraints
Single-turn single-task	Generate one user turn that fully specifies one conditional task and one outer call. Keep all Triggers and Actions of that task in the same call. Include a logical relation only when the task contains multiple Triggers.
Single-turn multi-task	Generate one user turn containing independent conditional tasks. Map each task to a separate outer call, preserve the order in which the tasks are mentioned, and do not merge components across task boundaries.
Delete Action	Generate two user turns. The first turn creates a task with multiple Actions. The second turn removes exactly one specified Action. Its target is a complete update that retains the original Triggers, control fields, remaining Actions, and all arguments not changed by the user.

checks and verifies consistency among the assigned application scenario, the dialogue, and the target structure. Samples that do not satisfy these criteria are corrected and revalidated or discarded.

English translation. Only validated Chinese dialogues enter the translation stage. Translation preserves the task intent, task boundaries, Trigger–Action pairing, modification scope, and assigned argument values. Each English sample is checked for semantic completeness and fluency and for consistency with the intended tool selection, arguments, and nested structure. Samples that fail these checks are discarded. Consequently, the English set is derived from the Chinese set but is not aligned with it one to one.

C Evaluation Details

This section supplements the evaluation protocol in Section 4 with the normalization rules, counting units, and diagnostic error categories used by TA-CallBench. The protocol evaluates the structural correctness of model-generated calls without executing the represented Triggers or Actions.

C.1 Matching and Normalization

Evaluation unit and reference state. Each `assistant` turn with reference tool calls defines one evaluation instance. Creation and modification turns in a multi-turn dialogue are evaluated separately. When a modification turn is evaluated, its dialogue history contains the reference call and tool response from the preceding turn. All models therefore receive the same valid prior state, and an earlier prediction cannot propagate an error into a later instance.

Evaluated output. Only model-generated tool calls participate in matching. Natural-language responses, reasoning traces, and call identifiers are excluded, and tool responses are not predicted. The evaluator compares the ordered list of outer calls and, within each aligned outer call, the outer operation, logical relation, repetition mode, ordered Trigger list, and ordered Action list. The update identifier associates a request with an existing task but is excluded from the target content.

Structural normalization. Outer arguments and serialized inner calls are parsed before comparison. Differences in whitespace, escaping, and object-key order do not affect a match. The labels `parameters` and `arguments` are treated as equivalent for inner arguments, while the order of outer calls and the order of Trigger and Action components remain significant. Missing, null, and empty forms of an unspecified logical relation or repetition mode are normalized to the same empty state; a nonempty value where none is requested is incorrect. A prediction that cannot be converted into the required nested structure does not match the reference.

C.2 Metrics and Counting Units

Table 7 summarizes the metrics derived from structural matching. TCA is the primary comparison metric. The remaining metrics localize errors at the dialogue, outer-call, or inner-component level and are used only for auxiliary analysis.

Trigger and Action name and argument accuracies provide a finer decomposition of the corresponding component-list metrics. Name accuracy checks the selected inner tool at each aligned position, while argument accuracy checks the complete argument object of that inner call. These local measurements can remain high even when TCA is low because TCA requires all outer and inner components in a prediction turn to be correct simultaneously.

C.3 Diagnostic Error Categories

Table 8 defines the error categories used in the analysis. Categories are assigned at the level where an error is observed. A prediction can contain several error types, so these counts overlap and should not be summed as a partition of incorrect predictions.

Table 7: Metrics and their counting units. TCA is the primary metric; the other metrics support diagnostic analysis.

Metric	Counting unit	Correctness condition
Tool-Call Accuracy (TCA)	Assistant prediction turn	The outer-call count is correct, and every aligned call matches in operation, logical relation, repetition mode, and complete ordered Trigger and Action lists.
Dialogue Accuracy (DA)	Dialogue	Every evaluated assistant turn in the dialogue is correct under TCA.
Outer-Call Count Accuracy	Assistant prediction turn	The predicted and reference lists contain the same number of outer calls.
Outer-Tool Name Accuracy	Aligned outer call	The predicted creation or update operation matches the reference operation at the same position.
Arguments Accuracy	Aligned outer call	The logical relation, repetition mode, Trigger list, and Action list all match.
Trigger Accuracy	Aligned outer call	The complete Trigger list matches in component count, order, tool identifier, and arguments.
Action Accuracy	Aligned outer call	The complete Action list matches in component count, order, tool identifier, and arguments.
Logic Accuracy	Aligned outer call	The normalized relation among multiple Triggers matches the reference.
Repeat Accuracy	Aligned outer call	The normalized repetition mode matches the reference.

Table 8: Diagnostic error categories and counting units. Categories may overlap within one prediction.

Error category	Counting unit	Definition
Outer-call count	Assistant prediction turn	The prediction omits a required outer call or produces a redundant one.
Outer-tool name	Aligned outer call	The predicted creation or update operation is incorrect.
Logical relation	Aligned outer call	The relation among multiple Triggers is missing, redundant, or incorrect.
Repetition mode	Aligned outer call	The predicted repetition setting differs from the requested setting.
Trigger mismatch	Aligned outer call	The complete Trigger list differs in count, order, identity, or arguments.
Action mismatch	Aligned outer call	The complete Action list differs in count, order, identity, or arguments.
Trigger name/argument	Inner Trigger call	An aligned Trigger uses an incorrect tool identifier or argument object.
Action name/argument	Inner Action call	An aligned Action uses an incorrect tool identifier or argument object.

D Complete Results

This section provides the complete results underlying the comparisons in Section 5. We report the eight models with valid results in all four interface configurations and preserve separate Chinese and English measurements throughout.

D.1 Complete Tool-Call Accuracy Results

Table 9 reports TCA for every model, language, and interface configuration used in the complete comparison. Each entry is formatted as CN/EN. The full results show that Guidance improves performance in nearly all comparisons across model families, while the preferred placement of the inner-tool schemas remains model dependent.

Table 9: Complete TCA (%) under the four interface configurations, reported as CN/EN. The mean is macro-averaged over the eight models.

Model	System	Tool List	System+Guidance	Tool List+Guidance
Open-source models				
Qwen3.5-0.8B	6.60/8.12	6.79/7.01	7.53/10.49	5.86/7.86
Qwen3.5-4B	15.90/17.21	16.55/16.97	43.46/38.73	46.41/40.13
Qwen3.5-9B	19.76/18.82	19.80/18.82	34.33/21.60	51.58/43.11
Qwen3.5-27B	21.39/20.37	20.56/20.04	51.07/43.61	49.13/41.61
Qwen3.5-35B-A3B	21.76/21.41	21.80/21.12	54.20/45.08	54.03/43.58
Proprietary models				
GPT-5.4	23.41/24.99	24.88/26.02	42.58/44.34	45.45/45.43
GPT-4o	7.30/7.47	9.13/10.59	30.83/25.36	22.86/26.02
DeepSeek-V4-Pro	19.56/18.29	19.91/18.55	41.74/37.15	47.93/41.50
Mean	16.96/17.09	17.43/17.39	38.22/33.30	40.41/36.16

Table 10: TCA (%) by task group and language. Results are macro-averaged over eight models and two schema placements.

Task group	No Guidance		Guidance	
	CN	EN	CN	EN
Single-turn single-task	2.24	2.04	42.85	33.93
Single-turn multi-task	0.05	0.00	26.09	16.24
Multi-turn single-task	10.27	11.27	32.65	28.09
Multi-turn modification	21.00	21.01	40.68	36.95

D.2 Task-Group and Language Breakdown

Table 10 expands the task-group comparison in Table 2 by reporting Chinese and English separately. Each value is macro-averaged over the eight models with complete results and the two schema placements within the stated Guidance condition.

Guidance yields higher TCA for every task group in both languages. Single-turn multi-task remains the lowest-performing group in each language, preserving the central finding that separating multiple independent tasks within one request is particularly demanding. The magnitude of the Chinese–English difference varies across task groups, which supports reporting language-specific results rather than relying only on an overall bilingual average.

D.3 Trigger and Action Accuracy by Model

Table 11 decomposes the two Guidance configurations into Trigger Accuracy and Action Accuracy. Each metric requires the corresponding ordered component list, including all tool identifiers and arguments, to match completely.

The component results remain substantially higher than the corresponding TCA values in Table 9. This gap appears across all model families and supports the distinction between locally correct components and a completely correct nested prediction.

D.4 Inner-Component Error Counts

Table 12 reports the Trigger and Action name and argument errors under System+Guidance. This configuration is used because it explicitly provides the nested-structure specification while keeping the inner tool definitions in the system prompt. Each cell reports Chinese/English (CN/EN) counts, and the mean is computed across the eight models with complete results. Because one prediction can contain multiple errors, the columns are overlapping diagnostic counts rather than a partition of incorrect predictions.

Table 11: Trigger Accuracy and Action Accuracy (%) under the two Guidance configurations, reported as CN/EN.

Model	System+Guidance		Tool List+Guidance	
	Trigger	Action	Trigger	Action
Qwen3.5-0.8B	57.87/55.63	65.16/61.01	69.54/71.03	61.53/55.56
Qwen3.5-4B	81.25/78.90	76.77/70.50	78.92/77.92	76.11/68.39
Qwen3.5-9B	74.83/58.70	79.88/71.68	82.91/78.36	83.82/72.95
Qwen3.5-27B	81.49/78.22	78.63/71.65	78.35/76.58	79.73/69.18
Qwen3.5-35B-A3B	85.27/80.79	82.77/72.04	84.41/78.04	83.44/70.45
GPT-5.4	83.74/83.37	79.17/79.04	83.00/82.76	79.22/79.26
GPT-4o	86.25/74.97	80.06/77.34	88.82/75.86	81.86/75.52
DeepSeek-V4-Pro	83.86/79.67	82.55/73.99	84.58/81.14	83.31/74.57
Mean	79.32/73.78	78.12/72.16	81.32/77.71	78.63/70.74

Table 12: Inner-component error counts under System+Guidance, reported as CN/EN. Name and argument errors are counted over aligned inner calls. Categories may overlap.

Model	Trigger Name	Trigger Args.	Action Name	Action Args.
Qwen3.5-0.8B	81/131	183/331	52/83	152/280
Qwen3.5-4B	64/73	617/874	8/7	724/1,136
Qwen3.5-9B	58/27	735/994	9/10	758/1,054
Qwen3.5-27B	109/88	700/893	46/46	885/1,185
Qwen3.5-35B-A3B	76/77	585/823	5/8	690/1,205
GPT-5.4	75/67	661/650	18/10	767/773
GPT-4o	21/23	300/627	0/1	473/537
DeepSeek-V4-Pro	66/64	570/746	45/46	577/944
Mean	68.75/68.75	543.88/742.25	22.88/26.38	628.25/889.25

Argument errors dominate name errors for every model and in both languages. Averaged over the eight models, the Chinese set records 543.88 Trigger argument errors against 68.75 Trigger name errors, roughly eight times as many, and 628.25 Action argument errors against 22.88 Action name errors, roughly 27 times as many. The contrast is even sharper in English, where the corresponding averages are 742.25 against 68.75 and 889.25 against 26.38, roughly eleven and 34 times as many. Because name and argument errors share the same aligned inner calls as their denominator, these ratios are unaffected by the call volume of each model. This consistent pattern shows that models usually select an appropriate inner tool but fail to recover every schema-constrained argument required by the complete conditional task, which identifies argument generation as a central source of complete-call errors in TA-CallBench.

D.5 Schema-Format Deviations

The strict matching in TA-CallBench treats an inner call as correct only if it follows the schema-defined wrapper, in which each call is an object with a `name` field and a `parameters` object. The error audit reveals a related but distinct category: a model produces the correct tool identity and argument values, but violates this wrapper. Two representative forms appear. In the first, the model emits the tool identifier under a `type` key and flattens the arguments to the top level:

```
GT   : [{"name": "heart_rate_fence",
         "parameters": {"threshold_value": 180,
                        "comparison": "above"}}]
PRED : [{"type": "heart_rate_fence",
          "threshold_value": 180, "comparison": "above"}]
```

Both calls carry the intended tool and values, but only the reference form conforms to the schema. Under a relaxed interpretation that compares only the tool identity and argument values, 459 of the

4,114 incorrect Chinese instances and 253 of the 4,038 incorrect English instances for Qwen3.5-35B-A3B would be judged correct, corresponding to 11.2% and 6.3% of its incorrect instances. The deviation affects GPT-5.4 (10.4% of incorrect instances in English) and DeepSeek-V4-Pro (1.0% in English) to a lesser extent and is nearly absent for the remaining models. Two observations motivate retaining the strict interpretation. First, the wrapper is part of the tool contract that downstream executors rely on, and a call that ignores it would not execute in a real interface. Second, the deviations are highly model specific, reflecting a fixed output habit rather than a general inability to organize nested calls. We therefore report these numbers as a diagnostic rather than adjusting TCA.

E BFCL Reference Results

This section provides the BFCL results used as the external reference in Figure 1. We obtain these scores by running the official BFCL v4 evaluation code. For a model-matched comparison, Table 13 includes the five models evaluated on both benchmarks. The TA-CallBench column reports TCA on the Chinese version under System+Guidance, following the setting used in Figure 1.

Table 13: Model-matched comparison of BFCL scores and TCA (%) on the Chinese TA-CallBench under System+Guidance. BFCL values are obtained with the official BFCL v4 evaluation code.

Model	BFCL	TA-CallBench TCA
Qwen3.5-0.8B	53.50	7.53
Qwen3.5-35B-A3B	77.80	54.20
GPT-5.4	72.19	42.58
GPT-4o	74.92	30.83
DeepSeek-V4-Pro	73.36	41.74
Mean	70.35	35.38

Across the five shared models, the average score is 70.35% on BFCL and 35.38% on TA-CallBench, with per-model differences ranging from 23.60 to 45.97 percentage points. The consistent separation shows that performance on standard function calling does not fully characterize the ability to generate conditional nested tool calls, where the outer operation, nested Trigger and Action calls, arguments, and logical relations must be jointly correct. TA-CallBench therefore complements existing function-calling evaluation by measuring this structured calling capability directly.