RTCBench: Evaluating Tool Use of Large Language Models Beyond Oracle Access

Qi Wei
College of Computing and Data Science
Nanyang Technological University
weiq0010@e.ntu.edu.sg

Bo An*
College of Computing and Data Science
Nanyang Technological University
boan@ntu.edu.sg

Abstract

Large language models (LLMs) have demonstrated promising capabilities in tool calling. However, existing benchmarks predominantly evaluate models in an idealized setting where ground-truth tool definitions are provided in context. This assumption diverges significantly from real-world deployment scenarios, where the model must first identify user intent and retrieve relevant tools from a large, heterogeneous tool repository before invoking them. In this work, we introduce a Retrieval-augmented Tool-calling Benchmark (RTCBench). In RTCBench, we collect 1,840 tasks from existing benchmarks and augment them with two tool environments: 1) `None`: no available tools provided in the prompt and 2) `Distractor`: strong distractor tools, while the target tools are excluded. By removing oracle tool access, RTCBench compels LLMs to actively retrieve relevant tools from a large-scale repository via a unified search tool before making tool calls, thereby faithfully simulating the end-to-end complexity of real-world tool-use pipelines. Through comprehensive evaluation of diverse LLMs ranging from 4B-parameter models to proprietary frontier models, we identify two core deficiencies: unreliable tool discovery, where models fail to initiate effective retrieval actions, and post-retrieval grounding failures, where models cannot reliably select and invoke the correct tools from semantically similar retrieved candidates. Furthermore, to mitigate these issues, we release 13,278 retrieval-augmented tool-calling expert trajectories with high-quality chain-of-thought reasoning for LLM post-training. Experiments show that a 4B-parameter model post-trained with GRPO improves its tool-calling accuracy on RTCBench from approximately 47.9% to over 85.8%, approaching the performance of the state-of-the-art model Gemini-3.1-Pro.

1 Introduction

Tool-integrated reasoning has emerged as a key mechanism for extending large language models (LLMs) beyond generation based solely on internal knowledge [Xue et al., 2025]. By invoking external tools such as APIs, search engines, and MCPs with domain-specific software, LLMs can access up-to-date information and perform actions beyond the models' intrinsic capabilities. Consequently, improving the tool-use capability of LLMs has become an important research topic.

To assess LLMs' tool-calling performance, several benchmarks have been proposed, including BFCL [Patil et al., 2025], API-Bank [Li et al., 2023], and ACEBench [Chen et al., 2025]. However, these benchmarks are largely built on a idealized setting: during the evaluation, the system prompt of LLMs is typically given a small, curated set of tools (often fewer than ten candidates) that already contains the ground-truth tool. As a result, they mainly verify whether LLMs can select the correct

*Corresponding author.
Project link: `https://github.com/1998v7/RTCBench`

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

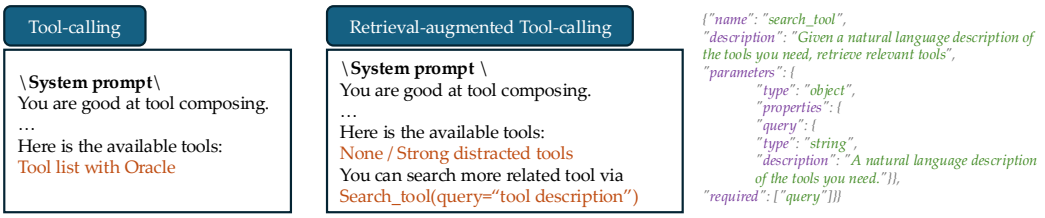


Figure 1: Comparison previous tool-calling task with the retrieval-augmented tool-calling ones from the perspective of system prompt, where a retrieval tool named `Search_tool` is given to retrieve relevant tools from external tool banks.

tool and provide appropriate arguments when the correct answer is guaranteed to be visible, instead of testing the full decision-making ability required in realistic large-scale tool-calling environments.

This limitation becomes increasingly severe as tool ecosystems grow. In practical deployments, exposing the entire tool inventory to LLMs is costly and ineffective: it incurs substantial prompt overhead, increases latency and monetary cost, and introduces a noisy decision space in which many candidate tools are irrelevant, redundant, or semantically overlapping. Under such conditions, simply asking the model to choose from a monolithic in-context tool list is unlikely to scale.

A more realistic tool-calling paradigm is therefore *retrieval-then-call*: before tool execution, the agent first retrieves a small subset of potentially relevant tools from a large repository, and then decides whether and how to invoke tools from this retrieved tool set. This formulation separates large-scale tool discovery from downstream action generation, making tool use more compatible with practical context constraints. Nevertheless, it introduces a new evaluation challenge. LLMs' performance now depends not only on whether it can generate a correct tool call when the right tool is present, but also on whether it remains robust to retrieval errors, near-duplicate tools, semantically overlapping tools, and variations in candidate-set quality.

To address this gap, we introduce RTCBench, a Retrieval-augmented Tool-Calling Benchmark for evaluating the tool-calling capabilities of modern LLMs under the *retrieval-then-call* paradigm (see Figure 1). RTCBench is built from 1840 tasks collected from existing benchmarks [Patil et al., 2025, Wang et al., 2024], where we augment with two challenging tool environments: `None`, no available tools provided in the prompt, and `Distractor`: strongly distracted tools without target tools. These settings require LLMs to implicitly perform the multi-stage tool-use pipeline: identifying user intent, reasoning about the tools needed in context, retrieving relevant tools from a large-scale tool-bank through a unified search interface, and finally invoking the selected tools. Specifically, we formalize RTCBench as an end-to-end retrieval-then-call evaluation pipeline, equipped with four metrics for *tool-calling accuracy*, *tool recall score*, *retrieval usage*, and *hallucination detection*, thereby enabling fine-grained analysis of model failures across tool discovery, search behavior, and grounded function-call execution.

Through comprehensive evaluation of models ranging from 4B-parameter systems to frontier models, we discover two challenging failure cases in such scenario: **unreliable tool-discovery behavior**, where models struggle to translate user intent and contextual tool insufficiency into effective retrieval actions; and **post-retrieval grounding failures**, where LLMs struggle to ground their tool-use decisions in retrieved tool sets containing highly semantically similar candidates. Beyond evaluation, RTCBench also provides a post-training resource: 13.4k high-quality retrieval-augmented tool-calling expert trajectories with chain-of-reasoning thinking, including intent reasoning, search behavior, grounded tool invocation, and refusal when no suitable tool is retrieved. Using this data, a 4B-parameter model can be substantially improved, increasing Level-1 accuracy from 47.9% to 85.8% and Level-2 accuracy from 12.3% to 56.5%.

2 Related Works

Tool-Integrated Reasoning. Equipping large language models (LLMs) with external tools has emerged as a widely adopted paradigm for extending their capabilities beyond parametric generation [Qin et al., 2024a, Wen et al., 2026]. By delegating specific tasks to specialized executors—such

as a Python interpreter for mathematical reasoning [Gao et al., 2023], web search engines for knowledge grounding [Nakano et al., 2022], or domain-specific APIs for real-world actions [Schick et al., 2023]—tool-integrated reasoning enables LLMs to obtain new knowledge and operate more accurately across diverse task domains. Specifically, early work primarily focused on framework-level designs that enable LLMs to interleave reasoning with tool-using actions, such as ReAct-style reasoning [Yao et al., 2023] and plan-then-execute paradigms [Wang et al., 2023]. More recent studies has shifted towards training-time optimization through reinforcement learning for multi-step tool reasoning, as exemplified by Search-R1 [Jin et al., 2025], SimpleTIR [Xue et al., 2025], and ReTool[Feng et al., 2025]. Despite this progress, existing studies largely assume a small, fixed set of tools, leaving it unclear how tool-using agents can scale to large, dynamically retrieved tool environments.

Tool-Calling Benchmarks. Many benchmarks have been developed to evaluate the tool-use capabilities of LLMs from different perspectives. For example, API-Bank [Li et al., 2023] provides a hierarchical evaluation spanning single to multi-step tool calls, while ToolBench [Qin et al., 2024b] and its stabilized variant StableToolBench [Guo et al., 2024] scale evaluation to thousands of real-world APIs with improved reproducibility. BFCL [Patil et al., 2025] focuses on fine-grained tool-calling accuracy across diverse programming languages via AST-based assessment, and T-Eval [Chen et al., 2024b] decomposes tool use into sub-skills such as planning, retrieval, and reasoning to provide more fine-grained evaluation metrics. However, these benchmarks generally assume that the relevant tools are directly provided in context or pre-filtered before inference, and thus do not capture the challenges of tool selection at scale. Our work complements this landscape by evaluating LLMs under the *retrieval-then-call* paradigm, where the candidate tool list is dynamically retrieved and may contain noise or semantically overlapping tools.

Tool Retrieval. As tool ecosystems scale to thousands of APIs, retrieving a high-related and compact list of tools has become a critical bottleneck for tool-using LLMs [Shi et al., 2025a, Huang et al., 2023]. Existing works on tool retrieval can be roughly summarized into two aspects. One line is to improve the performance of information retrieval (IR) model, normally fine-tuning a embedding model in tool field. For example, ToolLLM [Qin et al., 2024b] collected 16000+ real-world APIs and further tuned a API retriever, while ToolRet [Shi et al., 2025b] shows that general-purpose retrieval models suffer from heavy performance degradation in tool retrieval and provides dedicated data for tool-specific retriever training. Another line improves the quality of retrieval query side by using LLMs to infer user intent and generate more retrieval-friendly queries. For example, PLUTO [Huang et al., 2024] decomposes complex requests into actionable sub-tasks before retrieval, and Re-Invoke [Chen et al., 2024a] rewrites user requests into tool-invocation-oriented queries. In contrast to these works, RTCBench evaluates tool retrieval within an end-to-end retrieval-then-call setting, where models must not only retrieve relevant tools but also decide when to search, ground semantically similar retrieved candidates, and invoke the correct tools.

3 RTCBench: Retrieval-Augmented Tool-Calling Benchmark

In this section, we propose a Retrieval-augmented tool-calling benchmark, termed as RTCBench. Unlike existing tool-calling benchmarks, which directly provide a small tool list with ground-truth tool definitions in the prompt and reduce tool use to a selection-and-invocation problem, RTCBench provides a more realistic evaluation setting where no correct tool is guaranteed to be visible at the context/prompt. The model must first determine what tools are needed based on the user request, actively retrieve them from a tool bank containing hundreds to over a thousand candidates, and only then decide how to call them. This design shifts the evaluation focus from *"can the model call the right tool when it is given"* to *"can the model find and call the right tool when it is not"*. In the rest of this section, we first describe the task formulation (§3.1), then detail the data sources and construction process (§3.2), the evaluation metrics (§3.3), and finally the high-quality expert trajectories pipeline and dataset (§3.4).

3.1 Task Formulation

RTCBench adopts a multi-phase retrieval-to-call inference protocol which simulates the realistic workflow of a tool-use agent operating over a large tool bank. Rather than directly providing target tool descriptions in the system prompt as the former tool-calling benchmarks like BFCL Patil

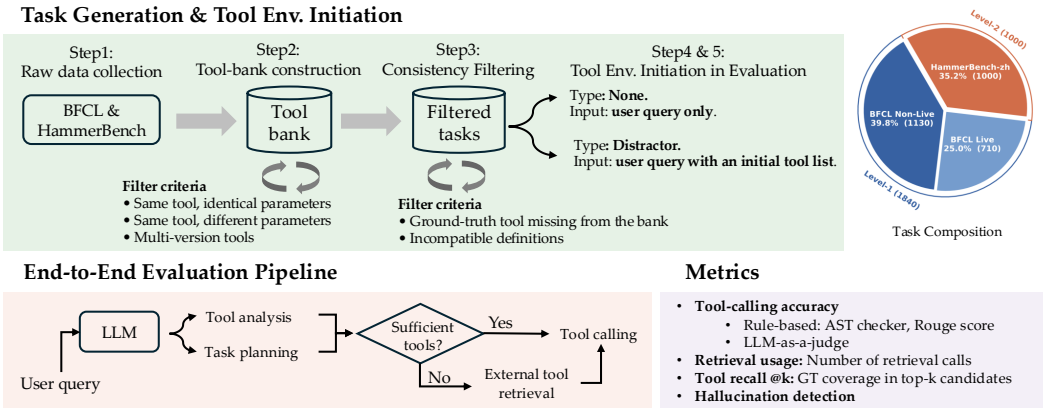


Figure 2: The RTCBench construction and evaluation pipeline: a five-stage process from raw data collection through tool-bank construction, consistency filtering, and tool-environment initiation under NONE / DISTRACTOR conditions, paired with an end-to-end agent loop and four orthogonal evaluation axes (accuracy, retrieval usage, tool recall@k, hallucination).

et al. [2025], the model is initially provided with some unrelated tools and a meta-tool, named `Search_tool`, which serves as a unified interface to a retrieval system backed by a large-scale tool bank. The definition of `Search_tool` can be found in Figure 1.

Each evaluation instance follows a *retrieval-then-call* pipeline, where LLMs must autonomously navigate both phases within an end-to-end interaction.

- **Retrieval phase:** LLMs receive a system prompt (see Figure 6 in Appendix) containing the `search_tool` schema and an optional set of initial tool definitions. Given the user query, the model must decide whether the currently available tools are sufficient to fulfill the request. If they are not, it should issue one or more `search_tool` calls with self-composed natural language queries (e.g., `search_tool(query="a tool that can place an express delivery order")`), and the system returns the `top-K` candidates from the tool bank.
- **Tool call phase:** a tool list including retrieved tool definitions inserted to the conversation context, and then the model generates the final tool call(s)—or, if no required tool was retrieved, declines to call rather than fabricate one.

Crucially, the two phases are not evaluated in isolation. A model that formulates an excellent retrieval query but misuses the returned tools, or one that can produce a perfect function call but only when the right tool is handed to it, will both be penalized under RTCBench's end-to-end accuracy. This tightly coupled design surfaces the full decision chain—from intent comprehension, through query formulation and candidate assessment, to grounded invocation—as a single, indivisible competency.

3.2 Benchmark Generation Pipeline

The construction of test instances in RTCBench follows a multi-step pipeline that transforms tasks from existing benchmarks into a unified, retrieval-ready evaluation suite. An illustration of the construction details is shown in Figure 2. More details and prompts are shown in Appendix B.

- **Step1: Raw Data Collection.** We first collect tasks from two widely-used tool-calling benchmarks. From BFCL [Patil et al., 2025], we gather 2,351 single-turn tasks across `non-live` (Python functions) and `live` (real-world REST APIs) splits, each consisting of a user query, ground-truth tool definitions. From HammerBench [Wang et al., 2024], we extract 500 Chinese single-turn tasks from the ST-Perfect subset, offering hierarchically named app-level APIs (e.g., `OnlineShopping.ExpressDelivery.placeExpressOrder`) that closely simulate real-world mobile and web service interfaces. Based on these data sources, we refer to the BFCL-derived tasks as **Level-1** and the HammerBench-derived tasks as **Level-2** throughout the paper.
- **Step 2: Tool Bank Construction.** We aggregate all candidate tools appearing across the source data into a tool bank. We then conduct the operation of *deduplication* and *quality filtering* to remove incomplete definitions, and project each tool into a searchable document by

concatenating its name, natural language description, and parameter-level descriptions. Before cross-split deduplication, the Level-1 tool inventory consists of 610 non-live tools and 435 live tools, yielding 1,045 tool entries in total. After removing six duplicate entries shared between the two splits, the final Level-1 tool bank contains 1,039 unique tools. The Level-2 tool bank contains 1,063 tools. Both tool banks serve as the retrieval targets during evaluation.

- **Step 3: Consistency Filtering.** To enable fair retrieval-augmented evaluation, we conduct the consistency filtering on these two banks, requiring that each task's ground-truth tools exist in the corresponding tool bank with matching definitions. Specifically, any task whose ground-truth tool name is absent from the bank, or whose parameter schema does not match the bank entry, is removed. This step filters out 1,431 BFCL tasks (60.9%), yielding 920 validated tasks.
- **Step 4: Tool Environment Initiation.** Beyond the oracle-access setting, we augment each task with two challenging tool environments:NONE and DISTRACTOR. In NONE, no tool definitions are provided in the prompt, requiring the model to proactively discover relevant tools through retrieval. In DISTRACTOR, the system prompt includes only misleading but incorrect tools, requiring the model to recognize the insufficiency of the visible candidates and initiate retrieval instead of prematurely invoking a plausible distractor. These two environments are designed to evaluate complementary aspects of tool-use behavior: proactive tool discovery in the absence of visible candidates and robustness to misleading in-context tool information.

  To construct distractor tools, we rank all candidate tools in the corresponding level-specific tool bank by their semantic similarity to the user query using an open-source embedding model [Qin et al., 2024b]. The ranking is performed over the entire tool bank rather than a pre-filtered subset. We then randomly select 5–10 tools from ranks 20–30, excluding the ground-truth tools. This rank-controlled sampling strategy introduces semantically relevant but incorrect candidates while avoiding the most similar tools that may be functionally equivalent to the ground truth. This produces topically related distractors that are plausible enough to mislead but remain incorrect, making the evaluation more realistic and informative than uniformly sampled noise.

  Step 5: Variant Assignment and Assembly. We instantiate each validated task under both NONE and DISTRACTOR conditions, keeping the user query and ground-truth tool calls fixed across variants. This design isolates the effect of tool availability while preserving the same task intent and expected solution. As a result, RTCBench contains 1,840 BFCL-derived instances for Level-1 (920 tasks $\times$ 2 conditions) and 1,000 HammerBench-derived instances for Level-2 (500 tasks $\times$ 2 conditions).

3.3 Evaluation Setup

Retrieval Backends. During evaluation, LLMs should pose natural language queries via `Search_tool` and then the external retrieval system will conduct information retrieval and feedback the tools. RTCBench supports two retrieval backends to enable systematic study. 1) **Sparse retrieval (BM25)**, which enables keyword matching over tokenized tool documents. 2) **Dense retrieval (Embedding)**, cosine similarity over sentence embeddings computed by a embedding encoder. Note that we adopt `ToolBench/ToolBench_IR_bert_based_uncased`[2] for Level-1 tasks and `BAAI/bge-base-zh-v1.5`[3] for Level-2 tasks, with corpus embeddings pre-computed and cached to avoid redundant computation.

Evaluation Metrics. In RTCBench, we adopt a wide range of metrics to comprehensively evaluate tool-using agent performance, where the metrics along four orthogonal axes:

- **Tool-Calling Accuracy**. A tool call is considered correct only if the final call matches the ground truth in both tool name and arguments content. For Level-1 tasks we use the official AST checker, which is same as that in BFCL; For Level-2 tasks, we use ROUGE-L-based matching, where both the tool name and each parameter value must achieve a ROUGE-L F1 score of at least 0.7 against the ground truth. For Level-1, we additionally employ GPT-4o-mini as an LLM-based equivalence judge for predictions that can be parsed as valid tool calls but fail the AST-based evaluation. The judge assesses whether the predicted tool calls are functionally equivalent to the ground truth and can fulfill the user request. This additional equivalence assessment is applied exclusively to Level-1, where different tools may provide overlapping functionality. For Level-2,

[2]https://huggingface.co/ToolBench/ToolBench_IR_bert_based_uncased
[3]https://huggingface.co/BAAI/bge-base-zh-v1.5

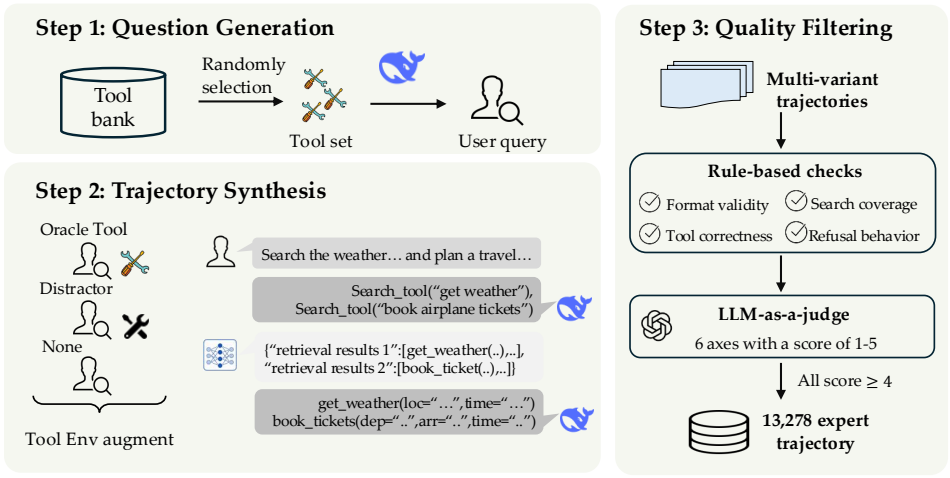


Figure 3: Overview of the expert-trajectory synthesis pipeline. We first sample tools from the RTCBench Level-1 tool bank and generate user queries with DeepSeek-V3.2. We then synthesize retrieval-augmented trajectories under *oracle*, *distractor*, and *none* tool environments, and finally apply rule-based checks and LLM-as-a-judge filtering to obtain 13,278 high-quality expert trajectories.

we rely on the ROUGE-L-based matching criterion without an additional LLM equivalence judge, as the Level-2 tool bank is constructed without functionally substitutable alternatives.

- **Recall@k**, which measures the coverage of ground-truth tools across the complete retrieval trajectory of each task. Let $\mathcal{G}$ denote the ground-truth tool set and $\mathcal{R}$ denote the union of retrieval results returned across multiple `search_tool` calls. Here, K specifies the maximum number of candidates returned per search call. The metric Recall@k can be represented by $|\mathcal{G} \cap \mathcal{R}|/|\mathcal{G}|$.
- **Retrieval Usage**. Since the agent itself decides whether or how to make tool retrieval, we report the average number of search calls per task (**Avg Search**), which can reflect the event of repeated searches at every turn, skipping retrieval, and so on.
- **Hallucination**. We track the hallucination from two dimensions: 1) **Skip-search**, LLMs do not conduct tool search first, despite no ground-truth tool being visible in the initial context. 2) **Phantom Tool**, the model calls a tool that appears in neither the initial context nor any retrieval result.

3.4 Expert Training Data

To complement the benchmark, we further synthesize a corpus of high-quality retrieval-augmented tool-calling trajectories, which can be used for supervised fine-tuning (SFT) or reinforcement learning (RL) and improve LLMs' performance in such a task. The synthesis pipeline reuses the RTCBench Level-1's tool bank and retriever, ensuring that training and evaluation are exposed to the same tool bank and follow the same multi-step format. More details and prompts are shown in Appendix C.

Question Generation. Referring to the setting of task difficulty in BFCL, we introduce four task types, *simple*, *multiple*, *parallel*, and *parallel_multiple*. Then, we sample one to five tools from the pre-collected tool bank and prompt a strong expert LLM, DeepSeek V3.2, with task-type-specific instructions and few-shot demonstrations. The expert model is asked to generate a natural user query that can only be completed by the selected tools.

Trajectory Synthesis. Given each generated user request, we prompt the expert model, DeepSeek V3.2, under the RTCBench system prompt to produce a complete inference trajectory, including `<think>` reasoning content, `search_tool` queries, the actual top-K retrieval results returned by the dense retriever, and the final tool call. Meanwhile, to explicitly cover refusal behavior (i.e., honest rejection when no target tools are found), we additionally construct a negative branch in which the ground-truth tools are masked from retrieval. We retain trajectories where the expert correctly refuses to output tool calls, rather than force-calling a near-duplicate.

Quality Filtering, which mainly contains two parts. Firstly, each trajectory is verified using six rule-based deterministic checks: format validity, output format, `<think>` length, call correctness,

Table 1: Performance comparisons with varying LLM models on **RTCBench Level-1**. Note that Recall@10 is computed only over trajectories in which the model successfully issued at least one parseable `search_tool` call; samples where the retrieval call could not be parsed are excluded.

Model	Non-Live	Live	Overall↑	Recall@10↑	Avg Search	Halluc(‰)↓
Proprietary Models						
Gemini 3.1 Pro	89.7	73.4	83.4	93.1	1.22	4.89
GPT-5.4	85.5	75.2	81.5	95.6	1.15	6.52
Gemini 3 Flash	81.8	64.6	75.2	94.2	1.20	7.61
Grok 4	77.4	57.9	69.9	92.6	1.06	5.98
Opus 4.6	75.8	59.9	69.7	95.8	1.28	11.04
Sonnet 4.6	68.8	58.3	63.6	94.7	0.92	13.85
Open-Source Models						
Qwen3-235B-A22B-Thinking	87.3	66.8	79.3	92.8	1.16	8.22
Qwen3.5-122B-A10B	83.5	71.4	78.8	**96.5**	2.48	9.13
Qwen3.5-397B-A17B	83.1	66.6	76.7	94.3	1.93	10.50
Qwen3.5-27B	77.5	64.2	72.4	95.5	1.85	10.33
DeepSeek V3.2	75.0	59.0	68.9	92.3	1.76	2.72
Kimi K2.5	65.3	57.2	62.2	**96.5**	1.53	7.61
Qwen3-14B	66.0	51.8	60.5	93.1	1.10	14.2
Qwen3-4B-Instruct	55.4	35.9	47.9	90.6	0.77	16.76
Qwen3-4B-Instruct (Ours)	**93.1**	**74.1**	**85.8**	94.9	1.11	**2.9**

Table 2: Performance comparisons with varying LLM models on **RTCBench Level-2**.

Model	Accuracy↑	Recall@10↑	Avg Search	Halluc(‰)↓
Proprietary Models				
Gemini 3.1 Pro	**64.9**	93.6	1.35	62
GPT-5.4	54.6	91.3	1.04	91
Gemini 3 Flash	59.1	92.8	2.05	64
Sonnet 4.6	52.4	92.5	0.90	103
Opus 4.6	52.3	93.5	0.99	96
Grok 4	51.4	90.3	1.02	70
Open-Source Models				
Qwen3.5-397B-A17B	56.3	94.1	1.76	90
Qwen3.5-122B-A10B	56.1	94.1	2.17	109
Qwen3-235B-A22B-Thinking	55.2	90.3	1.13	118
DeepSeek V3.2	54.3	93.1	1.90	49
Qwen3.5-27B	52.7	94.6	1.73	102
Kimi K2.5	51.2	**94.9**	2.15	69
Qwen3-14B	48.9	87.9	1.15	126
ToolACE-8B	50.6	89.3	1.40	119
Qwen3-4B-Instruct	12.3	86.5	0.75	141
Qwen3-4B-Instruct (Ours)	56.5	91.3	0.98	72

retrieval coverage, and refusal behavior when the ground-truth tool is missing. We then apply an LLM-as-judge, GPT-o4-mini, to score each trajectory along other six dimensions on a 1–5 scale. Only trajectories receiving a score of at least 4 on every dimension are retained.

Overall, the pipeline yields 13,278 expert trajectories, which can serve as a reusable resource for future research on tool calling and related post-training methods.

4 Experiments

In this section, we evaluate the performance of varying LLMs on retrieval-augmented tool calling in our constructed RTCBench. Meanwhile, we also verify if a small open-source model (Qwen3-4B) simply post-trained on our expert trajectories can achieve the comparable performance with the frontier model. Finally, we visualize retrieval behavior and conduct a fine-grained error analysis to diagnose why existing models struggle under the retrieval-then-call setting.



Figure 4: Retrieval performance of different models on RTCBench. **Search Rate** denotes the fraction of trajectories posing at least one parseable `search_tool` call; **Recall@10 | searched** reports Recall@10 score among that searched subset.

4.1 Experimental Setup

To verify the effectiveness of our proposed RTCBench, we benchmark the following baseline models: Gemini-3.1-Pro [Google DeepMind, 2026], GPT-5.4 [OpenAI, 2026], Claude Sonnet 4.6 [Anthropic, 2026b], Claude Opus 4.6 [Anthropic, 2026a], Gemini-3-Flash [Google DeepMind, 2025], Grok 4 [xAI, 2025], Qwen3-235B-A22B-Thinking, Qwen3-32B [Qwen Team, 2025], DeepSeek-V3.2 [DeepSeek-AI, 2025], Qwen3.5-122B-A10B, Qwen3.5-397B-A17B, Qwen3.5-27B [Qwen Team, 2026], and Kimi K2.5 [Kimi Team, 2026].

Training Details. We post-train a Qwen3-4B-Instruct model in two stages on the synthesized trajectories, 1) we perform full-parameter SFT with LlamaFactory [Zheng et al., 2024], teaching the model to learn the retrieval-then-call pattern and the basic output format. 2) We continue from the SFT checkpoint with GRPO [Shao et al., 2024] in VeRL framework [Sheng et al., 2024]. Specifically, we introduce dense process reward in addition to the outcome reward, which is utilized to improve the retrieval efficiency in full inference trajectory. The details of reward design and training parameters are shown in Appendix A.

4.2 Main Results

Table 1 reports the results on the Level-1 branch of RTCBench. Overall, proprietary models show strong performance, with Gemini 3.1 Pro and GPT-5.4 achieving 83.4% and 81.5% overall tool-calling accuracy, respectively. Among the open-source baselines, the best-performing models still lag behind these frontier proprietary models. However, high retrieval recall alone does not guarantee strong end-to-end performance. For example, Kimi K2.5 achieves 96.5% Recall@10 but only 62.2% overall accuracy, suggesting that models still struggle to distinguish candidate tools with similar semantic signals and to generate grounded function calls after retrieval. In contrast, after post-training on our synthesized data, Qwen3-4B-Instruct improves substantially from 47.9% to 85.8% overall accuracy, surpassing all evaluated baselines on Level-1 tasks. Meanwhile, its hallucination rate is reduced from 16.76‰ to 2.9‰.

Table 2 further evaluates these LLMs under the more challenging RTCBench Level-2 setting. All models exhibit a clear accuracy drop compared with Level-1, confirming that Level-2 tasks require stronger retrieval-augmented reasoning and more robust tool-calling ability. Gemini 3.1 Pro remains the strongest proprietary model, achieving 64.9% accuracy, while most other proprietary and open-source models cluster around 50–60% accuracy. The original Qwen3-4B-Instruct performs poorly, with only 12.3% accuracy despite 86.5% Recall@10, again showing that retrieving relevant tools is insufficient without effective post-training for grounded tool use. After GRPO, our 4B model reaches 56.5% accuracy, matching or surpassing much larger open-source models while requiring fewer searches on average than many of them. At the same time, its hallucination rate is reduced from 141‰ to 72‰, demonstrating improved reliability in tool selection and function-call generation.

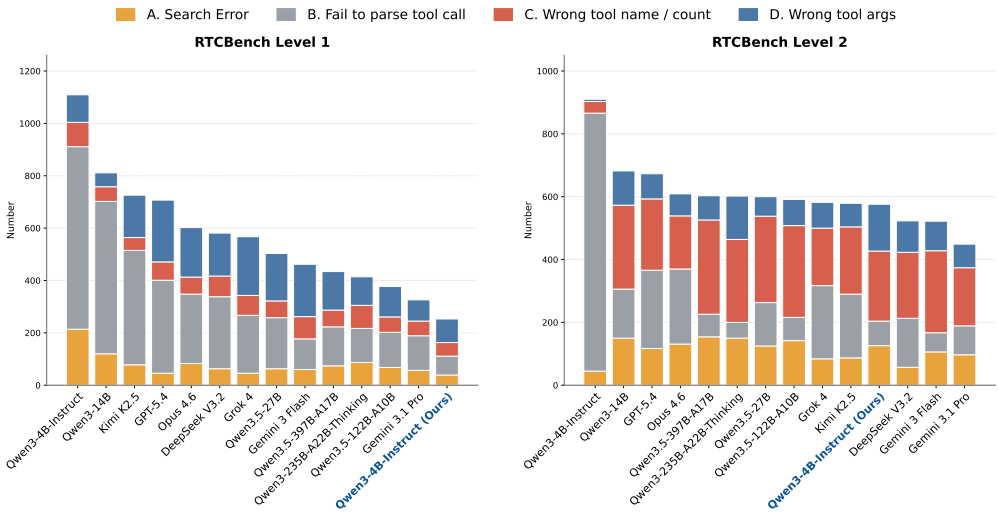


Figure 5: **Task failure type decomposition** of different models on RTCBench, including **(a) Search error:** the model fails to invoke search_tool when no tool is provided in the prompt, or invokes it with a query that does not retrieve any of the required ground-truth tools; **(b) Fail to parse tool call:** the model's output cannot be parsed as a valid tool call; **(c) Wrong tool name / count:** the model calls a tool whose name does not match the ground truth, or calls the wrong number of tools; **(d) Wrong tool args:** the model invokes correct tool(s), but at least one argument differs from the ground truth.

These results verify that targeted post-training can substantially improve small-scale LLMs in realistic retrieval-augmented tool-calling scenarios.

4.3 Error Case Analysis

To understand why existing models struggle to achieve high performance on RTCBench, especially in Level-2 tasks, we further visualize their retrieval behavior in Figure 4 and decompose their task failures in Figure 5, which reveals two fundamental bottlenecks.

First, LLMs exhibit **unreliable tool-discovery behavior**. Specifically, as shown in Figure 4, LLMs vary significantly in their search ratio, indicating that many of them do not consistently recognize when the initially visible tools are insufficient and when retrieval should be triggered. For example, the strong proprietary model GPT-5.4 does not reach the promising search ratio on either Level-1 or Level-2, suggesting that frontier models may still fail to consistently initiate retrieval when the required tools are absent from the initial context. Meanwhile, Recall@10 among successfully searched trajectories is generally high for many models, suggesting that the bottleneck is not only the external retriever, but also the model's ability to translate user intent and tool insufficiency into effective retrieval actions.

Second, models heavily suffer from **post-retrieval grounding failures**. In Figure 5, we count the failure cases from four dimensions. The result shows that a large portion of errors fall into wrong tool name/count and wrong argument categories, rather than only search errors. This indicates that even returned a set of candidate tools with ground-truth, models also fail to ground the correct ones. In particular, Figure 4 and Tables 1–2 reveal a clear gap between retrieval coverage on successfully searched trajectories and end-to-end tool-calling accuracy. For example, Kimi K2.5 achieves 96.5% Recall@10 on the searched subset of Level-1, but only 62.2% overall accuracy; similarly, the original Qwen3-4B-Instruct reaches 86.5% Recall@10 on the searched subset of Level-2, but only 12.3% accuracy. This suggests that retrieving relevant tools, when search is successfully triggered, is still insufficient for end-to-end success, since models must also decide when to search and faithfully ground retrieved candidates into correct tool calls.

Overall, these findings highlight why RTCBench evaluates retrieval and tool calling in a joint evaluation system: isolated retrieval metrics cannot fully reflect tool-use capability, while end-to-end accuracy alone cannot reveal whether failures arise from missed retrieval, ineffective search, or incorrect grounding after retrieval.

5 Conclusion

We introduced RTCBench, a retrieval-augmented tool-calling benchmark that evaluates LLMs beyond oracle tool access under a realistic retrieval-then-call paradigm. RTCBench requires models to identify user intent, retrieve relevant tools from large-scale tool banks, and ground retrieved candidates into executable tool calls. Our evaluation reveals two key failure modes: unreliable tool discovery and post-retrieval grounding failures. To further facilitate progress, we synthesize 13,278 high-quality expert trajectories covering intent reasoning, search behavior, grounded invocation, and refusal behavior. Experiments show that post-training a 4B model on these trajectories substantially improves its performance on RTCBench, demonstrating the value of retrieval-augmented supervision for building more reliable tool-using agents. We hope RTCBench can serve as a diagnostic testbed and training resource for /future research on scalable, retrieval-grounded tool use.

In future work, we plan to extend RTCBench with multi-turn tool-use cases. Such scenarios are more challenging and practical because user intent and tool requirements may evolve across turns, and models must track dialogue state, previous tool outputs, and when to re-trigger retrieval.

References

Anthropic. Claude Opus 4.6. `https://www.anthropic.com/news/claude-opus-4-6`, 2026a. Accessed: 2026-05-03.

Anthropic. Claude Sonnet 4.6. `https://www.anthropic.com/claude/sonnet`, 2026b. Accessed: 2026-05-03.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, et al. Acebench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*, 2025.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. In *Findings of the Association for Computational Linguistics*, 2024a.

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*, pages 9510–9529, 2024b.

DeepSeek-AI. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models, 2025. URL `https://arxiv.org/abs/2512.02556`.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.

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

Google DeepMind. Gemini 3 Flash Model Card. `https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf`, December 2025. Accessed: 2026-05-03.

Google DeepMind. Gemini 3.1 Pro Model Card. `https://deepmind.google/models/model-cards/gemini-3-1-pro/`, February 2026. Accessed: 2026-05-03.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Tenghao Huang, Dongwon Jung, and Muhao Chen. Planning and editing what you retrieve for enhanced tool learning. In *Findings of the Association for Computational Linguistics*, pages 975–988, 2024.

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

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Kimi Team. Kimi K2.5: Visual Agentic Intelligence, 2026. URL `https://arxiv.org/abs/2602.02276`.

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*, pages 3102–3116, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2022.

OpenAI. Introducing gpt-5.4. `https://openai.com/index/introducing-gpt-5-4/`, 2026. Accessed: 2026-05-07.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, 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*, 2025.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024a.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, 2024b.

Qwen Team. Qwen3 Technical Report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Qwen Team. Qwen3.5 blog. `https://qwen.ai/blog?id=qwen3.5`, 2026. Accessed: 2026-05-07.

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. *Advances in neural information processing systems*, 36:68539–68551, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*, 2024.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24497–24524, 2025a.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025b.

Jun Wang, Jiamu Zhou, Muning Wen, Xiaoyun Mo, Haoyu Zhang, Qiqiang Lin, Cheng Jin, Xihuai Wang, Weinan Zhang, and Qiuying Peng. Hammerbench: Fine-grained function-calling evaluation in real mobile device scenarios. *arXiv preprint arXiv:2412.16516*, 2024.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st annual meeting of the association for computational linguistics*, pages 2609–2634, 2023.

Yi Wen, Hao Chen, Wanyu Wang, Maolin Wang, Pengyue Jia, Derong Xu, Huize Tan, Yingyi Zhang, Wenlin Zhang, Weihong Luo, Xiku Du, and Xiangyu Zhao. Trajevolve: Trajectory evolution for reinforcement learning with hindsight credit assignment, 2026. URL `https://arxiv.org/abs/2610.28060`.

xAI. Grok 4 Model Card. `https://data.x.ai/2025-08-20-grok-4-model-card.pdf`, August 2025. Accessed: 2026-05-03.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Yaowei Zheng, Richong Zhang, Junhao Zhang, YeYanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics.

Broader Impacts

This work aims to evaluate and improve the capability of LLMs on a practical and challenging setting - retrieval-augmented tool-calling. By evaluating tool discovery, retrieval behavior, hallucinated tool calls, and grounded function-call generation, RTCBench can help researchers identify failure modes that are often hidden in oracle tool-calling settings. The synthesized trajectories may also support the development of more efficient and trustworthy tool-using agents.

A Post-training on the Synthesized Expert Trajectories

A.1 Reward Design in RL training

In such a retrieval-augmented tool-calling pipeline, the agent can fail at multiple stages, so a binary $0/1$ outcome reward provides almost no gradient in the dominant regime. We therefore decompose the reward into three additive terms, including

(1) Format reward. We require every assistant turn to start with the explicit tag before any tool call or response. Trajectories that violate this rule could receive the reward $R_{\text{fmt}} = -3$ *and* skip the rest context; otherwise $R_{\text{fmt}} = +1$.

(2) Process Reward (the search-behavior reward). Let $n(o)$ be the number of valid `search_tool` calls in the inference trajectory o and $\mathrm{Recall@}K$ the union recall of the ground-truth tools across the retrieval results returned to the agent. Given the number of `search_tool` calls $n^\star$ used by expert trajectory for the same prompt, the reward is variant-dependent:

$$R_{\text{search}} = \begin{cases} +3, & v = \text{WITH_GT},\ n = 0 \quad \text{(direct call, correct)} \\ -2, & v = \text{WITH_GT},\ n \geq 1 \quad \text{(redundant search)} \\ -2, & v = \text{NONE,DISTRACTOR},\ n = 0 \quad \text{(skip-search)} \\ 1 + 2 \cdot \mathrm{Recall@}K - \beta \cdot \mathbb{1}\big[n > n^\star + 1\big], & v = \text{NONE,DISTRACTOR},\ n \geq 1 \end{cases} \tag{1}$$

where β denotes a penalty strength and we set $\beta = 0.1$ in experiments. In the last situation, the base bonus of $+1$ ensures that simply attempting retrieval is always strictly better than skipping it (even at zero recall), which mitigates the "policy stops searching" failure mode. Besides, the soft count penalty discourages over-searching to achieve a high value of $\mathrm{Recall@}K$, while allowing one extra exploration call beyond the SFT trajectory.

Honest reject. Additionally, considering that the retrieval tools might not contain the ground-truth tool, i.e. $\mathrm{Recall@}K = 0$, the agent should recognize the gap and point it out rather than fabricating a call. Consequently, we set $R_{\text{search}} = +0.5$ to encourage agents to give an honest rejection.

(3) Outcome reward. For trajectories that pass the previous gates, we adopt the fine-grained, partial-credit matcher of ToolRL [Qian et al., 2025] to score the final call list $\hat{C}$ against the ground truth C, which jointly checks function-name overlap, required-argument coverage, and per-parameter value correctness. We linearly rescale the raw score $s(\hat{C}, C) \in [0, s^\star]$ onto $[-3, +3]$:

$$R_{\text{out}}(q, o) \;=\; 6 \cdot \frac{s(\hat{C}, C)}{s^\star} \;-\; 3, \tag{2}$$

where $s^\star$ is the prompt-dependent maximum.

Overall, in our GRPO training phase, these three terms are gated which can be written as:

$$R(q, o) \;=\; R_{\text{fmt}}(o) \;+\; R_{\text{search}}(q, o) \;+\; R_{\text{out}}(q, o), \quad R \in [-3, +7]. \tag{3}$$

A.2 Training Setup

We post-train a Qwen3-4B-Instruct in two stages on the synthesized tasks, where the training hyperparameters are shown in the Table 3.

Table 3: Training hyperparameters for the SFT and GRPO stages of our 4B model.

Hyperparameter	**SFT (LlamaFactory)**	**GRPO (verl)**
Base model	Qwen3-4B-Instruct-2507	SFT checkpoint
Trainable params	full (4B)	full (4B)
Epochs	2	10
Optimizer	AdamW (wd=0.01, $\beta = (0.9, 0.999)$)	AdamW (wd=0.01, $\beta = (0.9, 0.999)$)
Learning rate	1×10^{-5} (cosine, warmup 0.03)	1×10^{-6} (constant)
Global batch size	64 ($2 \times 8 \times 4$)	256
Sequence length	16,384	prompt 4,096 / response 16,384
Precision	bf16	bf16
Parallelism	DeepSpeed ZeRO-3	FSDP (no offload)
Gradient clip	1.0	1.0
GPUs	$4\times$ H100	$8\times$ H100

Models	**Overall**	**Non-Live**	**Live**
Closed-Source Models			
GPT-5-2025-08-07	65.59	72.92	58.25
GPT-4o-2024-11-20	77.21	83.88	70.54
Gemini-2.5-Pro	74.50	85.04	63.95
Gemini-3.1-Pro	83.42	87.27	79.57
o3-2025-04-16	53.01	39.98	66.03
Amazon-Nova-Pro	81.78	85.25	78.31
Grok-4-0709	79.80	85.21	74.39
Open-Source Models			
Moonshotai-Kimi-K2-Inst	80.80	84.02	77.57
DeepSeek-R1-0528	78.22	75.73	<u>80.90</u>
Qwen3-235B-A22B-Inst-2507	83.37	**90.12**	76.61
Llama-4-Maverick	80.90	88.15	73.65
ToolACE-8B	78.23	84.94	71.52
Qwen3-4B-Instruct (Ours)	**85.41**	<u>89.79</u>	**81.03**

Table 4: BFCL Single-Turn performance. Best results are in **bold**; second-best results are <u>underlined</u>.

A.3 More Training Results and Analysis

Training visualization with different reward designs. Figure 6 shows the count of search-tool calls during RL training with different reward designs. Naive GRPO gradually increases the count of search-tool calls with the training proceeds, suggesting that outcome-only reward encourages the model to search more in order to improve tool recall and final tool-calling accuracy. However, this can lead to inefficient over-searching because retrieval cost is not explicitly penalized. After adding our process reward (see Eq. 1), the search behavior becomes more controllable. With a mild over-search penalty ($\beta = 0.1$), the search count is controlled with a moderate range, while a stronger penalty ($\beta = 0.2$) keeps the average search count lower, though it may also reduce the model's willingness to explore external retrieval. As a consequence, we adopt the $\beta = 0.1$.

Results on RTCBench with different training strategies. Table 5 shows the effect of different post-training strategies. We can see that compared to SFT, naive GRPO improves accuracy on both Level-1 and Level-2 settings, but also increases the average count of search calls, which indicting that GRPO tends to improve final reward/performance by searching more. This behavior would increase tool recall but also leads to longer contexts and higher inference cost.

With the proposed process reward, the model achieves a better balance in final performance and the retrieval behaviors. The setting of $\beta = 0.1$ achieves the best tool call accuracy while using fewer searches and producing fewer hallucinations. When the penalty is stronger ($\beta = 0.2$), the search count is further reduced, but accuracy and Recall@10 also drops slightly, indicating that an a stronger penalty may discourage the model from exploring external retrieval. Overall, these results show

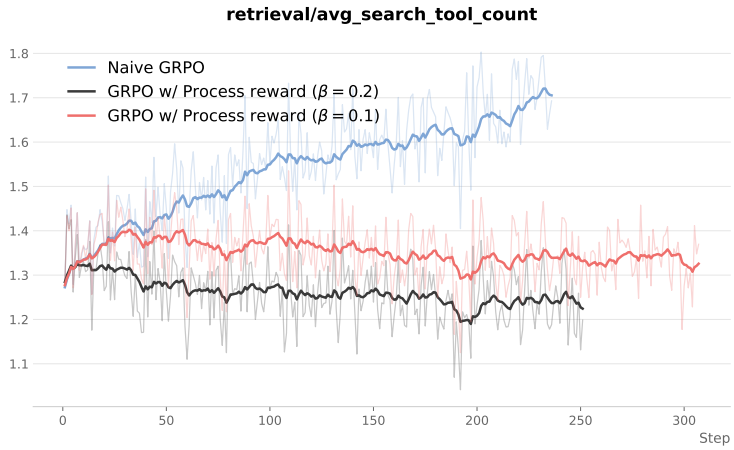


Figure 6: **Average count of search-tool usage** during RL training with different reward designs.

Table 5: Performance comparisons of different post-training strategies on RTCBench.

Level	Metric	SFT	GRPO	GRPO w/ Penalty ($\beta = 0.1$)	GRPO w/ Penalty ($\beta = 0.2$)
Level-1	Accuracy (%)	81.7	84.9	**85.8**	84.1
	Recall@10 (%)	93.8	**96.1**	94.9	94.5
	Avg Search	1.37	1.54	1.11	1.01
	Hallucination (‰)	8.4	5.7	**2.9**	5.4
Level-2	Accuracy (%)	52.8	54.9	**56.5**	55.6
	Recall@10 (%)	90.7	**93.1**	91.3	90.9
	Avg Search	1.24	1.41	0.98	0.92
	Hallucination (‰)	109	94	**72**	84

that our reward design effectively improves retrieval efficiency while maintaining strong tool-calling performance.

B RTCBench Construction Details

B.1 Discussion of two settings: Level-1 & Level-2

Our evaluation consists of two complementary settings. Level-1 serves as an in-inventory evaluation setting, where the expert trajectories used for post-training and the evaluation tasks share the same tool bank, although their task instances are distinct. This setting measures the improvements achievable when the model is trained and evaluated within the same tool environment. In contrast, Level-2 serves as a cross-inventory generalization setting: the expert trajectories are synthesized exclusively from the Level-1 tool bank, whereas Level-2 uses a disjoint tool bank derived from HammerBench Wang et al. [2024], whose tools and task instances are not used during trajectory synthesis. This setting evaluates whether the tool-discovery and grounding capabilities acquired through post-training transfer to an unseen tool inventory. Together, the two settings distinguish improvements within a familiar tool inventory from generalization to a different tool inventory.

B.2 Inference-Time System Prompt

During evaluation in this RTCBench, every tested model is given the system prompt as shown in the Table 6. Note that the placeholder `tools` is replaced with the initial visible tools according to the evaluation condition: empty for NONE, strong distractors without ground-truth for DISTRACTOR, and normal tools with ground-truth for WITH_GT. For HammerBench-zh we additionally require the `search_tool` query to be written in Chinese.

You are an expert in function calling. You are given a user question and required to find the appropriate tools to fulfill the request. You must strictly follow this decision policy:

1. First determine whether the currently available tools are sufficient to fulfill the user's request.
   - If the currently available tools can fulfill the request, directly call the appropriate function(s).
   - If any required capability is missing, you MUST call the `search_tool` function to retrieve more relevant tools before giving any final answer.
2. If `search_tool` has already been called and the returned tools cannot fulfill to the user's request, please point it out in your final answer. Do NOT call `search_tool` again.

Important Instructions:
- Do NOT answer the user's question directly with your internal knowledge.
- The user's request must be handled only through the available tools or tools retrieved from `search_tool`.

Output format:
- To retrieve tools: `<think>`*reasoning*`</think>`[search_tool(query="..."), ...]
- To call functions: `<think>`*reasoning*`</think>`[func_name(param=value, ...), ...]
- Otherwise: `<think>`*reasoning*`</think>`Your final response

There is a `search_tool()` function that can retrieves relevant tools. `search_tool`.
Here are the available tools: `tools`

Table 6: **System prompt template** for RTCBench. `search_tool` will be replaced with the JSON schema of `search_tool`. `tools` will be replaced with the initial tool definitions according to the evaluation condition: empty for NONE, tools with the JSON format for DISTRACTOR and WITH_GT.

B.3 Tool-bank Deduplication

The unified tool bank used by the benchmark is firstly collected from all tested tasks and then deduplicated to remove functionally equivalent APIs/tools. Specifically, for each anchor tool, we retrieve its top-20 similar tools within a dense embedding model and filter those where the similarity score is larger than 0.5. Then, the anchor tool and its peer tools will be sent to a LLM judge to decide which candidates are functionally interchangeable. Finally, the equivalent tools will be removed based the LLM output. The prompt for this LLM judge is shown as follows

Tool-Bank Deduplication Prompt

```
You are an expert at identifying functionally equivalent API tools.
Given an **anchor tool** and a list of **candidate tools**, determine which candidates are
      **functionally equivalent** to the anchor.
Two tools are equivalent if:
1. They serve the same purpose and would produce the same result for the same intent
2. Their parameters are substantially overlapping (names may differ)
3. One could replace the other in any reasonable use case
## Anchor Tool
{anchor}
## Candidate Tools
{candidates}
Return a JSON object (no markdown fences):
{"equivalent": ["candidate_name_1", ...], "reasons": {"candidate_name_1": "brief reason",
     ...}}
If NO candidate is equivalent, return: {"equivalent": [], "reasons": {}}
```

B.4 Post-Evaluation LLM Judge

Despite some effort we made in tool-bank deduplication, we found there are still some function-similar or equivalent tools existed. To avoid underestimating the model's performance on selecting a *different but equivalent* tool, we forward every response that can parses correctly yet fails on ground-truth matching. Here, we adopt GPT-o4-mini as LLM judge to confirms if the agent selects equivalent tools compared with the given ground-truth. The prompt is shown as follows

Post-Evaluation LLM-Judge Prompt

```
You are an expert tool-calling evaluator.
A LLM was asked to call tool(s) to answer a user question. It called different tool(s)
    than the expected ground-truth (GT).
Determine whether the tools called by the LLM is **functionally equivalent** to the GT and
     able to solve the user request.
## User Question
{question}
## Tool Definitions
{tool_defs}
## Ground-Truth Calls
{gt_calls}
## Model's Calls
{model_calls}
## You SHOULD consider the following criteria:
- Do the model's calls serve the same purpose as the Ground-Truth calls for this question?
- Are the arguments semantically equivalent (even if parameter names differ)?
- Would executing the model's calls give the user an answer equivalent to the Ground-Truth
    ?
Answer with a JSON object (no markdown fences):
{"equivalent": true/false, "reason": "brief explanation"}
```

C Synthesized Expert Training-Data

C.1 Synthesis Details

The expert-trajectory pipeline of Section 3.4 relies on a strong proprietary teacher model and rule-based and LLM-as-Judge filters. Both roles are driven by the prompts below.

Stage 1 - Question Generation. We first sample 1-5 tools from the pre-defined tool bank. For every sampled tool group, we ask the teacher to imagine a realistic user scenario and emit a paired (question, ground-truth-call) record. Few-shot exemplars (omitted here for brevity) are appended in the `<Examples>` block. Here, we use DeepSeek-v3.2 as the expert model.

Question Generation Prompt

```
You are creating training data for a function-calling benchmark.
Given the following tool definition(s), generate a realistic, natural user question
that would require calling the tool(s) as specified below.
<Task type>
{task_type_description}
</Task type>
<Examples>
{few_shot_examples}
</Examples>
IMPORTANT:
- Create a single coherent scenario that naturally requires ALL given tools.
- The tools should serve a unified user intent, not unrelated requests stitched together.
- For parallel/parallel_multiple tasks: all tool calls MUST be independent -- no call
    should depend on the output of another call. The user should be able to invoke all
    tools simultaneously. Do NOT create sequential/chained workflows like "first get X,
    then use X's result to call Y".
Now generate for these tools:
{tools_json}
Output ONLY a JSON object (no markdown fences, no extra text) with:
- "question": a natural language user question
- "ground_truth": a string of function calls in bracket format: [func_name(param1=value1,
    param2="value2"), ...]
  String parameter values must be quoted, numeric/boolean values unquoted.
  Use ONLY parameters defined in the tool schemas. Use realistic parameter values.
  Function names must EXACTLY match the tool "name" field.
```

```
CRITICAL: The "question" MUST contain enough information for EVERY parameter in the
     ground_truth calls.
If a ground_truth call includes a parameter, the question must explicitly mention or imply
      the value.
Do NOT include parameters in ground_truth whose values cannot be inferred from the
     question.
```

Stage 2 - Expert Trajectory Synthesis. The expert model then completes a multi-turn rollout under the same retrieval interface that adopts in the RTCBench. We instantiate three visibility variants: **With_gt** (ground-truth tools plus distractors already visible), **Distractor** (tools without ground-truth) and **None** (only `search_tool` is visible). In last two cases, the model is required to search first and then call tools from the returned the tool list. In this phase, we use the same prompt as the inference system prompt shown in Table 6.

Stage 3 - LLM-as-Judge. Every accepted trajectory is independently scored by GPT-o4-mini judges along six rubrics; a sample is kept only when all scores are larger than 4.

LLM-as-Judge Prompt

```
You are a strict quality evaluator for function-calling training data.
You will be given:
1. A conversation trajectory (system/user/assistant messages)
2. The tool definitions (name, description, parameters with types/descriptions, required
     fields)
3. The ground-truth calls (for reference only -- they may contain errors)
4. Whether the retrieval results contained the correct tools ([RETRIEVAL HAS GT])
Your PRIMARY task is to evaluate whether the assistant's FINAL tool calls can correctly
     solve the user's request, based on the tool definitions. Do NOT blindly trust the
     ground-truth -- evaluate the assistant's output on its own merits.
Evaluate on these dimensions (1-5 scale):
1. **think_coherence**: Is the reasoning inside <think> blocks logically sound, step-by-
     step, and relevant to the user's question?
2. **call_correctness**: Check the assistant's FINAL tool calls against the tool
     definitions:
   (a) Are ALL required parameters present?
   (b) Do parameter VALUES strictly follow the tool description (types, units, decimal vs
     percent)?
   (c) Do parameter values respect enum constraints?
   (d) Are parameter values consistent with what the user actually asked for?
3. **search_usage**: Did the assistant use search_tool correctly?
    (a) Searched for ALL distinct sub-tasks needing different tools?
    (b) Queries are specific and non-redundant?
    (c) Correctly skipped search when tools were already available?
4. **refusal_quality**:
   (a) When [RETRIEVAL HAS GT] is False, the assistant MUST refuse gracefully instead of
     force-calling wrong tools.
   (b) When GT was retrieved, score based on correct usage.
5. **answerability**:
   (a) Can the final tool calls actually answer the user's question?
   (b) Are the right tools chosen with appropriate parameters to produce a useful result?
6. **overall_quality**: Is this trajectory natural, coherent, and suitable as training
     data?
Output ONLY a JSON object:
{"think_coherence": N, "call_correctness": N, "search_usage": N,
 "refusal_quality": N, "answerability": N, "overall_quality": N,
 "verdict": "pass" or "fail"}
A trajectory PASSES only if ALL six scores are >= 4. Otherwise it FAILS.
```

C.2 Data Distribution Visualization

In Figure 7, we visualize the distribution statistics of the synthesized training set from the following six aspects, a) the scope from which a tool originates (live or non_live type in the Level 1); b) the

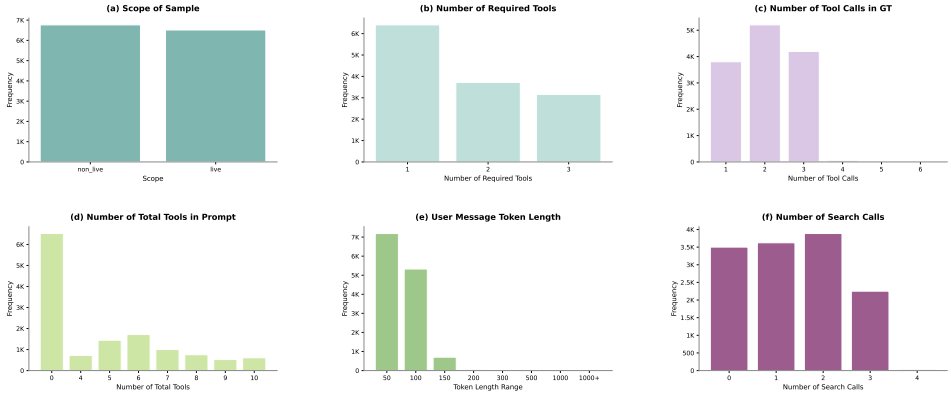


Figure 7: Distribution statistics of expert trajectories provided by RTCBench, where the size is 13,278.

number of distinct target tools required by the user prompt; c) the total number of tool-calling for completing this task; d) the number of candidate tools provided in the system prompt; e) the token length distribution of the user prompt, which is approximated by $\text{len}(chat)/4$; and f) the number of `search_tool` invocations in all expert trajectories.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the main contributions, including RTCBench, the retrieval-then-call evaluation setting, the synthesized expert trajectories, and the post-training results. The reported claims are supported by the benchmark construction, evaluation metrics, and experimental results in Sections 3 and 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations in the conclusion and future work, including that RTCBench currently focuses on single-turn settings and will be extended to multi-turn tool-use scenarios.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not present theoretical results, theorems, or formal proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper describes the benchmark construction pipeline, tool-bank construction, evaluation protocol, metrics, expert-trajectory synthesis, reward design, and training hyperparameters. These details provide the information needed to reproduce the main experimental results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We will release the RTCBench benchmark, synthesized expert trajectories, evaluation scripts in an anonymized repository upon publication. The paper provides the construction and evaluation details needed to use these assets.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the benchmark data sources, tool-bank construction, evaluation settings, retrieval backends, metrics, model list, training stages, reward design, and hyperparameters in the main paper and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No].

   Justification: We report deterministic benchmark scores for a broad set of models but do not include error bars or statistical significance tests, mainly due to the high computational cost of repeated full benchmark evaluation and post-training runs.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The appendix reports the main compute setup for post-training, including the number and type of GPUs used for SFT and GRPO. We use 4 H100 GPUs for SFT and 8 H100 GPUs for GRPO.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research follows the NeurIPS Code of Ethics. The benchmark is constructed from existing tool-calling benchmarks and synthesized tool-use trajectories, and does not involve human subjects or sensitive personal data collection.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: answerYes

Justification: The paper studies retrieval-augmented tool calling, which can improve the reliability and efficiency of tool-using agents. We also acknowledge potential risks from more capable tool-using agents, such as incorrect external actions or misuse, and discuss mitigation through hallucination detection, refusal behavior, and grounded tool invocation.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A].

Justification: The paper does not release a general-purpose pretrained model or scraped web dataset with high misuse risk. The released assets are intended for benchmark evaluation and tool-calling research.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the existing benchmarks, models, retrievers, and training frameworks used in this work, including BFCL, HammerBench, ToolBench retriever, BGE, LlamaFactory, and VeRL. We will include corresponding license and usage information in the released package.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces RTCBench and 13,278 synthesized expert trajectories. The benchmark construction, evaluation protocol, data synthesis pipeline, filtering criteria, prompts, and data distribution statistics are documented in the main paper and appendix.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing experiments or human-subject studies. The data are constructed from existing benchmarks and LLM-generated synthetic trajectories.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human-subject research, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are an important part of the methodology. The paper describes the use of DeepSeek-V3.2 for question generation and expert trajectory synthesis, GPT-o4-mini for trajectory quality judging, and GPT-4o-mini for post-evaluation judging.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.