On the Robustness of Agentic Function Calling

Ella Rabinovich
IBM Research
ella.rabinovich1@ibm.com

Ateret Anaby-Tavor
IBM Research
atereta@il.ibm.com

Abstract

Large Language Models (LLMs) are increasingly acting as autonomous agents, with function calling (FC) capabilities enabling them to invoke specific tools for tasks. While prior research has primarily focused on improving FC accuracy, little attention has been given to the robustness of these agents to perturbations in their input. We introduce a benchmark assessing FC robustness in two key areas: resilience to naturalistic query variations, and stability in function calling when the toolkit expands with semantically related tools. Evaluating best-performing FC models on a carefully expanded subset of the Berkeley function calling leaderboard (BFCL), we identify critical weaknesses in existing evaluation methodologies, and highlight areas for improvement in real-world agentic deployments.

1 Introduction

Large Language Models (LLMs) are reshaping artificial intelligence, shifting from static language processors to dynamic, task-oriented *agents* capable of planning, executing, and refining their actions. These agents hold the potential for transformative applications across various domains, including healthcare (Abbasian et al., 2023; Mehandru et al., 2024), finance (Li et al., 2024; Xiao et al., 2024; Ding et al., 2024), education (Yang et al., 2024; Xu et al., 2024), and customer support (Huang et al., 2024; Rome et al., 2024). LLM agents have been revolutionarily positioned as routing systems that can act independently, make decisions and perform tasks with minimal human intervention.

Agentic Function Calling Function calling (FC), the process by which an agent autonomously selects and invokes a specific function to retrieve information or execute a task, serves as a fundamental building block of an agentic system. In this context, a full execution trajectory can be seen as a complex, multi-turn (i.e., involving user interaction) sequence of function calls, ultimately achieving a given goal. Models specifically optimized for FC are typically designed to generate a function call in response to a natural-language user request (Bai et al., 2023; Dubey et al., 2024; Zhang et al., 2024). The function (also known as a tool) is chosen from a predefined "toolkit"—a compact set of function descriptions[1]—provided as part of the model's prompt. The agent is expected to produce a syntactically correct tool invocation, ensuring that parameter values are appropriately assigned to function arguments (a process known as slot filling). For instance, given the query, "What is the record for the highest number of points scored by a single player in an NBA game?" and the compact `json` tool description in Figure 1 (top), the model is expected to generate the invocation code shown in Figure 1 (bottom). Several datasets and evaluation methodologies have been proposed to assess LLMs' function calling capabilities (Patil et al., 2023; Liu et al., 2024), and various benchmarks have been created for evaluating a range of FC scenarios, BFCL leaderboard (Patil et al., 2023) among the most prominent ones.

Robustness of Large Language Models In the context of the more "traditional" LLM usage, a *model robustness* quantifies an LLM's ability to generate semantically equivalent outputs, given semantically equivalent inputs (Raj et al., 2023; Rabinovich et al., 2023; Ackerman et al., 2024). Robustness benchmarks assess, among other factors, how well LLMs handle naturally-occurring, non-malicious perturbations in user input, such as paraphrased questions in a QA task, typos, variations in punctuation, whitespace, or diacritics. Extending this notion to agentic FC would require a model to produce an equivalent tool invocation despite naturalistic, yet, strictly meaning-preserving, per-

[1]Descriptions are often provided in the `json` format.

```
{"name": "basketball.most_points_single_game",
 "description": "returns the record for the highest <...>",
 "parameters":
   {"type": "dict", "properties": {
      "league": {"type": "string", "description": <...> }
      },
    "required": ["league"]
   }
}
```

```
{
 "basketball.most_points_single_game":
   {"league": ["NBA"] }
}
```

Figure 1: Compact function definition example (top), and agent's output, triggering the function call with assigned parameter values (bottom), per user request "What is the record for the highest number of points scored by a single player in an NBA game?".

turbations in the input query. Considering Figure 1, a semantically equivalent paraphrase "What is the highest number of points ever scored by a single player in an NBA game?" should result in the same tool invocation as the original request.

Despite its clear practical significance, research on the robustness of agentic function calling remains sparse, with only two studies, to the best of our knowledge, examining agent resilience to modifications in tool descriptions. Ye et al. (2024) introduce a series of increasingly aggressive alterations to *function names*, *parameter names*, and their *descriptions* – to the point where a tool (or a parameter) name (or description) becomes arbitrary or entirely uninformative about its functionality. Similarly, Lu et al. (2024) conduct multiple interventions, including tool distractions, within a different evaluation framework that evaluates tool sequencing at the *system* rather than *function* level. While these studies offer valuable insights, they provide limited evidence on agent resilience to real-world perturbations, as system developers typically exert *substantial control* over the faithfulness and level of detail in function and parameter names, along with their descriptions.

Moreover, a typical "toolkit" (the list of available functions) in these studies is limited to a single tool or a small number of unrelated tools. A realistic scenario may involve a system specification with thousands of available tools,[2] which in practice is normally reduced to top-K most relevant function definitions through a shortlisting module (Qin et al., 2023), such as semantic search over the set of tools, towards constructing the context (here, prompt) of a FC agent. In the example toolkit in Figure 1 (top), additional tools may include: `basketball.most_points_career()`, `basketball.most_points_single_season()`, `basketball.game_stats()`.

Contribution We focus on two aspects of robustness, capturing input variations that can be expected in real-world agentic deployments but are *not easily controlled* by a developer: (1) generating meaning-preserving rephrasings of user requests and (2) expanding the toolkit to include a set of semantically related tools that are likely to be shortlisted by a selection module. Using one of the (single-turn) challenging BFCL (Patil et al., 2023) test sets as our starting point, we first carefully build a benchmark dataset, comprising variations pertaining to the two aforementioned aspects (Section 2). Next, we evaluate the robustness of several best-performing LLMs[3], and discuss the breakdown of failures, highlighting (among others) prominent weaknesses of the existing agentic FC evaluation benchmarks (Section 3). Our benchmark data is available at `https://huggingface.co/datasets/ibm-research/BFCL-FC-robustness`.

2 Dataset Generation

We next provide details on the generation of our benchmark dataset. Specifically, we describe the creation of (1) meaning-preserving rephrasings of user requests and (2) expanding the toolkit to include a set of semantically related tools.

2.1 User Query Perturbations

Building on the study by Ackerman et al. (2024), who tested LLMs' sensitivity to paraphrased user queries in the QA and classification settings, we investigate whether agents' FC capabilities remain robust to meaning-preserving variations in user requests. Here, the task presents additional challenge, as the rewording must strictly maintain precise parameter values to ensure accurate slot filling for the sake of evaluation. For instance, the request "Calculate the depreciated value of a property costing $200,000 with an annual depreciation rate of 3%

[2]A software engineering (SWE) agent fixing git issues, has access to about 1.2K tools exposed through github docs.

[3]According to the BFCL leaderboard (Jan 2025).

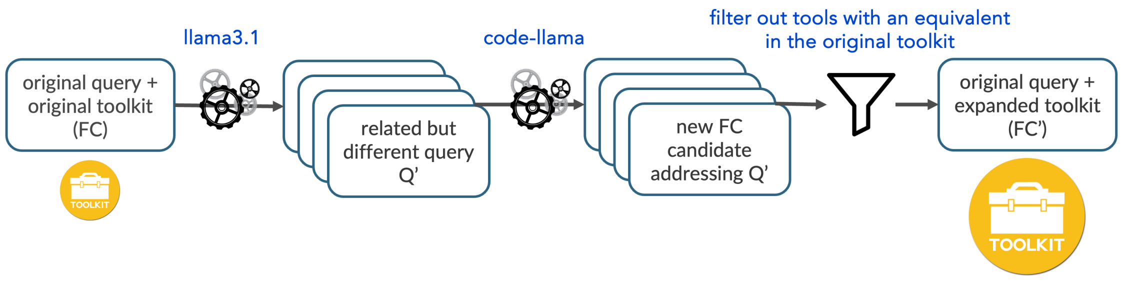
llama3.1
code-llama
filter out tools with an equivalent in the original toolkit
original query + original toolkit (FC)
TOOLKIT
related but different query Q'
new FC candidate addressing Q'
original query + expanded toolkit (FC')
TOOLKIT

Figure 2: A toolkit expansion steps: (1) request variants are generated using the LLama3.1-70B model (Dubey et al., 2024), (2) function `json` definitions for executing these requests are generated using the Code-Llama-13B model (Roziere et al., 2023), and a filtering step (3) is applied to filter out tools semantically identical to any of the original functions. The process is completed when the expanded toolkit is created for testing the original query.

original request	What is the record for the most points scored by a single player in an NBA game?
original toolkit	basketball.most_points_single_game(...)
request variants	Who holds the record for the highest number of assists made by a female basketball player? What is the longest winning streak in NBA history? ...

additional tools	basketball.most_points_career(...)
	basketball.records_history(...)
	...

Table 1: A toolkit expansion steps: request variants and additional tools addressing those variants.

for 5 years." can be safely rephrased as "Determine the value of a $200,000 asset which loses 3 percent of its worth each year, after five years." Contemporary LLMs handle this task effectively, and we used the Llama3.1-70B model (Dubey et al., 2024), with appropriate prompting and in-context learning. A manual review of 50 examples by one of the authors revealed no instances of semantic drift or parameter misalignment. Appendix 7.1 provides details on the prompt used for this task.

A substantial portion of the paraphrases targeted named entities, which are natural candidates for surface form variability. For instance, the user query "What is the humidity level in `Miami,Florida` in the upcoming 7 days?" was rephrased as "How will the humidity levels change over the next seven days in `Miami,FL`?". These seemingly minor modifications led to a notable drop in benchmark performance – we analyze and interpret this decline, and propose strategies to mitigate it in Section 3.

2.2 Expanding Agent's Toolkit

Aiming at expanding the (originally) "thin" agent's toolkit, simulating the scenario where function definitions are retrieved by a shortlister, we follow the steps illustrated in Figure 2 and outlined in Table 1.

(1) We generate *related yet different* request variants using the Llama3.1-70B model (Dubey et al., 2024), see Appendix 7.2 for the detailed prompt.

(2) For each request variant, a tool definition is generated to enable request fulfillment. Here, we used the CodeLlama-13B model (Roziere et al., 2023) with a carefully designed prompt and few-shot examples, ensuring that the generated definitions conform not only to the required `json` format but also to the naming conventions, style, and level of detail in function and parameter descriptions. Notably, based on our manual inspection, the style of the generated tool definitions is indistinguishable from that of the original function(s).

(3) In rare cases, a generated tool was found to be strictly functionally equivalent to the original one, despite differences in name, description, or parameter order (see Appendix 7.3). We eliminate such cases by (a) concatenating the original tool properties into a "signature," and (b) filtering out any newly generated tool whose "signature" exceeded a predefined similarity threshold to the original tool, as measured via cosine similarity of their embeddings, computed using the sentence-transformers module (Reimers and Gurevych, 2019).

Table 1 presents example original request (and its tool), along with the expansion process: additional (related but not strictly identical) request variants, and additional tools, fulfilling those additional requests. The mean number of tools in the expanded toolkit is 5.6 compared to the 2.7 (seemingly unrelated) tools in the original BFCL dataset, meaning that three semantically-related functions were added on average to each one of the 200 test-cases. Next, we evaluate the FC performance of multiple agents using the generated benchmark.

3 Agentic FC Robustness Evaluation

3.1 Experimental Setup

Models We evaluate several top-performing LLMs from the BFCL leaderboard, both API-accessible and locally hosted, as FC agents. Closed models include GPT4o-mini and o1-mini,[4] as well as Claude-3.5-Haiku and Claude-3.5-Sonnet.[5] Locally hosted models include Llama3.1-70B and its more advanced version Llama3.3-70B (Dubey et al., 2024), Granite3.1-8B-instruct (Granite Team, 2024), DeepSeek-v2.5 (DeepSeek-AI, 2024), and Qwen2.5-72B (Qwen Team, 2024).

Evaluation Approach BFCL employ a two-phase FC evaluation approach: (1) assessment of the generated tool call through the tree-matching abstract syntax tree (AST) methodology, and (2) evaluation of the tool execution in a simulated environment (Patil et al., 2023). Our focus in this study is the evaluation of FC construction provided interventions in its input; we, therefore, adhere to the first evaluation phase – namely, AST. A robust agent will generate correct function call regardless of the precise request wording and of its toolkit size: "thin" (as it comes with the original benchmark), or expanded, simulating a shortlister selection.

3.2 Experimental Results

We report AST averaged over the 200 dataset examples, including three variants: (a) the original version, (b) original ("thin") toolkit + rephrased user request, (c) expanded toolkit + original user request. Table 2 (left) reports the results. Several insights can be drawn from the figures:

FC Evaluation Approach Weakness(es) A notable (and somewhat unexpected) drop occurs when evaluating the original toolkit on a rephrased request. Closer examination of errors reveals a significant weakness in the common approach to FC evaluation – specifically, in handling arguments that can accept several equivalently valid values (e.g., named entities). Consider the request: "What is the humidity level in `Miami,Florida` in the upcoming 7 days?". The expected response includes the function `weather.humidity_forecast()` and validates its `location` parameter by exact match to one of the predefined values: ["Miami", "Miami, Florida", "FL"]. When the request is rephrased as "How will the humidity levels change over the next seven days in `Miami,FL`?", agents assign the value "Miami, FL" to `location`, which does not match any of the (incompletely) listed options.

Further systematic analysis of error types distribution reveals that 70–90% of errors indeed stem from mis-match in parameter value assignment. We conclude that the majority of failures in this case can be attributed to the *evaluation approach drawback* rather than agents' sensitivity.

We argue that this issue could potentially be mitigated by applying *semantic similarity* instead of *exact match*. Indeed, recent studies adopt a more holistic approach to evaluation of a constructed function call; e.g., Zhong et al. (2025) who use multi-dimensional matching strategy, including FCs' embeddings similarity and LLM-as-a-Judge matching, ensuring a generated tool call meets its semantic requirements. We leave the exploration of this mitigation strategy in the context of BFCL evaluation framework to future work.

Agents' Sensitivity to Toolkit Expansion Evidently, expanding an agent's toolkit with a set of related functions caused performance degradation across the board (Table 2, left). Here, objective agent failures span a range of error types: wrong function selected, wrong number of functions generated (typically two instead of one), wrong parameter assignment to a correctly-selected function, parameter hallucinations, etc. As an example, in response to the request "What is the ranking of Manchester United in Premier League?", an agent with the expanded toolkit produces `football_league.ranking("premier league")`, retrieving the complete ranking table of the league, instead of the more appropriate `sports_ranking("Manchester United", "premier league")`, answering the query.

Table 2 (right) presents error breakdown for agents in this study in the expanded toolkit scenario, showing the proportion of each error type within the set of failures stemming from toolkit expansion. While no clear pattern dominates, it is evident that agents struggle with both accurate function selection and parameter assignment.

[4] `https://platform.openai.com/docs/models`
[5] `https://www.anthropic.com/claude`

	robustness evaluation			exp. toolkit + orig. query: error analysis (%)			
model (agent)	original	orig. toolkit reph. query	exp. toolkit orig. query	wrong syntax	wrong function	wrong num of functions	wrong param. assignment
Llama3.1-70B	0.965	0.825 (-15%)	0.925 (-4%)	0.00	0.45	0.10	0.45
Llama3.3-70B	0.945	0.785 (-17%)	0.905 (-4%)	0.00	0.23	0.46	0.31
DeepSeek-V2.5	0.965	0.835 (-14%)	0.950 (-2%)	0.00	0.56	0.00	0.44
Qwen2.5-72B	0.975	0.850 (-13%)	**0.965 (-1%)**	0.00	0.29	0.00	0.71
Granite3.1-8B-instruct	0.945	0.770 (-19%)	0.870 (-8%)	0.09	0.50	0.18	0.23
Claude-3.5-Haiku	0.925	0.765 (-11%)	0.870 (-2%)	0.00	0.44	0.00	0.56
Claude-3.5-Sonnet	0.915	**0.845 (-8%)**	0.890 (-3%)	0.00	0.29	0.00	0.71
gpt4o-mini	0.925	0.765 (-17%)	0.870 (-6%)	0.26	0.42	0.00	0.32
o1-mini	0.905	0.770 (-15%)	0.885 (-2%)	0.33	0.27	0.00	0.43

Table 2: Agentic FC robustness evaluation results. Models' AST performance drop is evident when rephrasing the original query, and also when using the original query with extended toolokit (left); relative percent drop is specified in brackets. Failures stemming from toolkit expansion vary mostly between wrong function selection and wrong parameter assignment (right). The best result in a column (the lowest performance drop) is boldfaced.

Finally, expanding an agent's toolkit with additional functions occasionally caused models to "repair" some of their original (baseline) failures in a few cases. Interestingly, this observations highlights the stochastic, generative nature of LLM agents, where seemingly unrelated changes to a model context may entail different output.

4 Conclusions and Future Work

We focus on two aspect of robustness, capturing input variations that can be expected in real-world agentic deployments: (1) meaning-preserving rephrasings of user requests and (2) agent's toolkit expansion to include a set of semantically related tools that are likely to be shortlisted by a selection module. We build a benchmark dataset, evaluate the robustness of several SOTA LLM agents, and discuss the breakdown of failures.

Our future work includes testing the robustness of agentic FC with additional and diverse datasets. Moreover, it has been shown that LLMs can be easily distracted by larger context (Shi et al., 2023; Levy et al., 2024). We plan to extend the set of experiments to scenarios were agent's toolkit is expanded also with non-relevant tools, to compare the performance against the current setting.

5 Limitations

While our study provides valuable insights into measuring agents' robustness in the function calling scenario, it has several limitations. First, we evaluate our approach on a single dataset, sufficient for the focused contribution of a short paper, but requiring extension to additional datasets for a broader analysis. Second, our toolkit expansion scenario relies on multiple LLMs to generate related requests and corresponding tools, a time-consuming process currently performed offline. We are actively exploring ways to streamline this pipeline for improved efficiency and usability.

6 Ethical Considerations

We use publicly available datasets to study the robustness of agentic function calling. We did not make use of AI-assisted technologies while writing this paper. We also did not hire human annotators at any stage of the research.

Acknowledgements

We are deeply grateful to Michal Jacovi for her invaluable assistance in carrying out this study. We would like to thank Guy Uziel for his feedback on earlier versions of this paper. Finally, we are thankful to our anonymous reviewers for their useful comments and constructive feedback.

References

Mahyar Abbasian, Iman Azimi, Amir M Rahmani, and Ramesh Jain. 2023. Conversational health agents: A personalized llm-powered agent framework. *arXiv preprint arXiv:2310.02374*.

Samuel Ackerman, Ella Rabinovich, Eitan Farchi, and Ateret Anaby Tavor. 2024. A novel metric for measuring the robustness of large language models in non-adversarial scenarios. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2794–2802, Miami, Florida, USA. Association for Computational Linguistics.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei

Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

Han Ding, Yinheng Li, Junhao Wang, and Hang Chen. 2024. Large language model agent in financial trading: A survey. *arXiv preprint arXiv:2408.06361*.

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

IBM Granite Team. 2024. Granite 3.0 language models.

Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. 2024. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments. *arXiv preprint arXiv:2411.02305*.

Mosh Levy, Alon Jacoby, and Yoav Goldberg. 2024. Same task, more tokens: the impact of input length on the reasoning performance of large language models. *arXiv preprint arXiv:2402.14848*.

Yuan Li, Bingqiao Luo, Qian Wang, Nuo Chen, Xu Liu, and Bingsheng He. 2024. CryptoTrade: A reflective LLM-based agent to guide zero-shot cryptocurrency trading. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1094–1106, Miami, Florida, USA. Association for Computational Linguistics.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. 2024. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. 2024. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*.

Nikita Mehandru, Brenda Y Miao, Eduardo Rodriguez Almaraz, Madhumita Sushil, Atul J Butte, and Ahmed Alaa. 2024. Evaluating large language models as agents in the clinic. *NPJ digital medicine*, 7(1):84.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

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

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

Ella Rabinovich, Samuel Ackerman, Orna Raz, Eitan Farchi, and Ateret Anaby Tavor. 2023. Predicting question-answering performance of large language models through semantic consistency. In *Proceedings of the Third Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*, pages 138–154.

Harsh Raj, Domenic Rosati, and Subhabrata Majumdar. 2023. Measuring reliability of large language models through semantic consistency. In *Proceedings of the ML Safety Workshop, NuerIPS*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Scott Rome, Tianwen Chen, Raphael Tang, Luwei Zhou, and Ferhan Ture. 2024. "ask me anything": How comcast uses llms to assist agents in real time. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2827–2831.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR.

Yijia Xiao, Edward Sun, Di Luo, and Wei Wang. 2024. Tradingagents: Multi-agents llm financial trading framework. *arXiv preprint arXiv:2412.20138*.

Songlin Xu, Xinyu Zhang, and Lianhui Qin. 2024. Eduagent: Generative student agents in learning. *arXiv preprint arXiv:2404.07963*.

Kaiqi Yang, Yucheng Chu, Taylor Darwin, Ahreum Han, Hang Li, Hongzhi Wen, Yasemin Copur-Gencturk, Jiliang Tang, and Hui Liu. 2024. Content knowledge identification with multi-agent large language models (llms). In *International Conference on Artificial Intelligence in Education*, pages 284–292. Springer.

Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. Rotbench: a multi-level benchmark for evaluating the robustness of large language models in tool learning. *arXiv preprint arXiv:2401.08326*.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, et al. 2024. xlam: A family of large action models to empower ai agent systems. *arXiv preprint arXiv:2409.03215*.

Lucen Zhong, Zhengxiao Du, Xiaohan Zhang, Haiyi Hu, and Jie Tang. 2025. Complexfuncbench: Exploring multi-step and constrained function calling under long-context scenario. *arXiv preprint arXiv:2501.10132*.

7 Appendices

7.1 Prompt for Request Rephrasing

We used the following prompt for generating *strictly meaning-preserving* request rephrasing with the Llama3.1-70B model (Dubey et al., 2024):

```
SYSTEM: You are a helpful assistant helping rephrasing user requests, while accurately preserving their meaning, including numbers and names if exist. Do not answer the requirement, just produce another one that is identical in meaning but is phrased differently. Produce ONLY the rephrased requirement, without further thoughts or explanations. Consider the example below:

USER: Can I find the dimensions and properties of a triangle, if it is known that its three sides are 5 units, 4 units and 3 units long?

ASSISTANT: What are the dimensions and properties of a triangle whose three sides are 5, 4 and 3 units long?
```

7.2 Prompt for Similar Requests Generation

We used the following prompt for generating *closely related but different* request with the Llama3.1-70B model (Dubey et al., 2024):

```
SYSTEM: You are a helpful assistant introduced with the following user query. Create a very similar query that refers to a very similar user need and is likely to be implemented in an enterprise as part of the same project. The new query should introduce one or two additional distinct parameter types. It should differ from the original query in a sense that a function that can be used to fulfill the original query is not fully appropriate for the new one and vise versa. As an example, generating 'Book a single room for two nights at the Hilton Hotel in Chicago' per the original query 'Book a double room for three nights at the Marriott hotel near OHare Airport in Chicago', is not sufficient since both queries can be answered using the same function call, invoked with different parameters. The query should contain all information needed for its computation. For instance, 'What is the capital of Brazil?' is a good query, while 'What is the capital of a country provided by user?' is not since one cannot generate a function call and populate its arguments using the info in the query alone. Output the newly generated query only, without explanation or interpretation. Consider the examples below:

USER: I need the schedules of matches happening on February 28, 2024.

ASSISTANT: I need the schedules of the college league matches happening during the winter 2024 season.
...
```

7.3 Example of Syntactically Different but Semantically Equivalent Tools

Although rare, distinct, yet functionally equivalent tools, pose a challenge for accurate evaluation, since the "labeled" BFCL data contains only one of these functions. As an example, the tool

```
sentence.translate(sentence: string,
from: string,
to: string)
```

is functionally equivalent to

```
translate_sent(orig_language: string,
target_language: string,
sentence: string).
```

As described in Section 2, we concatenate function name and description, as well parameter names and descriptions into a tool "signature", and filter out generated tools exhibiting cosine similarity higher than a predefined threshold to the original one, aiming at a toolkit with distinct functions. The similarity threshold was set to 0.8.