Sequential API Function Calling Using GraphQL Schema

Anonymous EMNLP submission

Abstract

Function calling using Large Language Models (LLMs) is an active research area that aims to empower LLMs with the ability to execute APIs to perform real-world tasks. However, sequential function calling using LLMs with interdependence between functions is still underexplored. To this end, we introduce GraphQL-RestBench, a dataset consisting of natural language utterances paired with function call sequences representing real-world REST API calls with variable mapping between functions. In order to represent the response structure of the functions in the LLM prompt, we use the GraphQL schema of the REST APIs. We also introduce a custom evaluation framework for our dataset consisting of four specially designed metrics. We evaluate three open-source code LLMs on our dataset using few-shot Chain-of-Thought and ReAct prompting to establish a reasonable baseline.

1 Introduction

Tool use in Large Language Models (LLMs) is an active area of research that aims to overcome the limits of pretraining LLMs (which usually results in a “knowledge cutoff date”) by enabling the LLMs to fetch data that they were not trained on using tools such as web APIs and databases. In this context the idea of using LLMs for function calling has gained traction since using tools in the form of functions requires LLMs to accurately pass correct parameter values to the functions. Any web API can be encapsulated as a function which requires inputs in a predefined format and outputs a structured response object.

The idea of empowering LLMs to use tools to harness external knowledge and perform complex computational tasks was introduced by Toolformer (Schick et al., 2024). There have been several attempts to train LLMs to use tools such as APIs (Liang et al., 2023; Shen et al., 2024; Patil et al., 2023; Song et al., 2023; Patil et al., 2024).

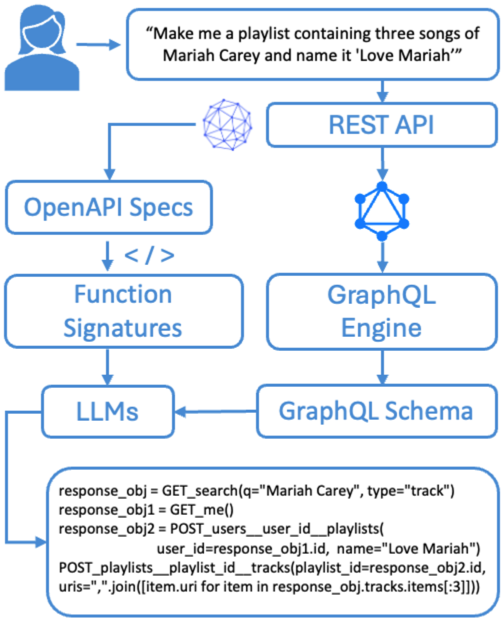

Figure 1: An example sequential function calling scenario from Spotify in GraphQLRestBench.

LLMs still do not perform well on API calling due to their inability to generate accurate input arguments and their tendency to hallucinate the wrong usage of an API call. It is essential for API-augmented LLMs to have robust planning and decision-making capabilities. Planning based approaches like ReAct (Yao et al., 2022) encounter challenges in effectively adapting API feedback and generating viable plans. RestGPT (Song et al., 2023) introduced a coarse-to-fine online planning mechanism for task decomposition and API selection, and API execution.

While methods like ReAct and RestGPT have demonstrated promising abilities for online planning and execution, they may generate incorrect APIs during the exploration phase. In contrast, Gorilla (Patil et al., 2023) focuses on the ability of the LLM to call a given API correctly. We wish to extend this approach to the sequential API execution scenario of RestGPT. While the Gorilla OpenFunctions framework (see the Berkeley Function Calling Leaderboard (Yan et al., 2024)) supports single and parallel function calls, it does not as yet support the use case of chained or sequential function calls where there exist mappings between the input and output parameters of functions.

The fundamental difficulty in calling sequential APIs in a single shot is the lack of knowledge about the response structure of APIs. While the OpenAPI specification of the API might provide some clue as to the response structure, it is often incomplete or inadequate for the purpose of defining the variable mapping in pythonic form.

GraphQL (Inc., 2015) is a query language for APIs that allows the user to easily find the useful fields and types in the API response object by inspecting the so-called GraphQL "schema" of the API using a feature called "introspection". As a solution to the above problem, we propose using the GraphQL schema of the APIs as a reliable source of information regarding their response structure. Tools like StepZen (IBM, 2024), Apollo (Apollo Graph Inc, 2024), and Hasura (Hasura, 2024) are available for automatically generating the GraphQL schema for querying RESTful APIs and databases.

In this paper, we introduce a new dataset, GraphQLRestBench which is built using the RestBench dataset introduced by RestGPT. Notably, RestBench only provides API sequences and not input-output parameter mappings between APIs. In GraphQLRestBench, we additionally add the GraphQL schema generated by StepZen for the APIs and also Python code to call the APIs in a sequence using input-output parameter mapping given the response structure of the APIs obtained from the GraphQL schema. The task is to generate the correct Python code consisting of a sequence of function calls with accurate parameter mapping between functions (see Figure 1). We introduce a custom evaluation framework for our dataset consisting of four task-specific metrics. We also evaluate three open source code LLMs on this task using Chain-of-Thought (Wei et al., 2022) and ReAct (Yao et al., 2022) style prompting as a reasonable baseline.

2 Related Work

Tool use and function calling (Mialon et al., 2023) presents a survey of augmented language models in general. Gorilla (Patil et al., 2023) introduced the idea of fine-tuning a base LLM for function calling by supplementing it with information retrieval. Toolformer (Schick et al., 2024) fine-tunes an LLM on the task of function calling with some custom built tools. (Yang et al., 2024) teaches LLMs to use such tools with self-instruction. TaskMatrix (Liang et al., 2023) studied the problem of task completion using a large number of APIs. ToolLLM (Qin et al., 2023) is a general tool-use framework encompassing data construction, model training, and evaluation over 16,000 APIs from RapidAPI Hub.

Agent-based frameworks have also been explored in this area. ReAct (Yao et al., 2022) studied the integration of reasoning and acting (by means of function calls) in LLM agents. Inspired by ReAct, RestGPT (Song et al., 2023) proposes a dual-agent planner-executor approach to connect LLMs with real-world RESTful APIs. (Song et al., 2024) introduced exploration-based trajectory optimization for open-source LLM agents by fine-tuning on the agent trajectories. AnyTool (Du et al., 2024) introduced self-reflective, hierarchical agents for API calling using the function calling ability of GPT-4 (Achiam et al., 2023). HuggingGPT (Shen et al., 2024) is an LLM-powered agent that connects various AI models in machine learning communities such as Hugging Face to solve AI tasks.

RESTful is the popular web service development standard (Li et al., 2016), which supports HTTP protocols and URIs to serve resources. OpenAPI Specification (Initiative, 2021) describes the operations, parameters, and response schemas in RESTful APIs.

Function calling datasets APIBench from Gorilla (Patil et al., 2023) consists of HuggingFace, TorchHub, and TensorHub APIs. RestBench from RestGPT (Song et al., 2023) consists of APIs from TMDB movie database and Spotify music player. ToolBench from ToolLLM (Qin et al., 2023) consists of 16,464 real-world RESTful APIs spanning 49 categories from RapidAPI Hub. AnyToolBench from AnyTool (Du et al., 2024) is similar to ToolBench but with a different evaluation protocol.

GraphQL (Wittern et al., 2018) discussed generating GraphQL wrappers for REST APIs using the OpenAPI specifications. (Farré et al., 2019) proposed automatic GraphQL schema generation for data-intensive web APIs using a semantic metamodel. Works such as (Brito and Valente, 2020) compare GraphQL and REST frameworks.

3 Methodology

In this section we explain the methodology we used to create the GraphQLRestBench dataset.

GraphQL schema Generation First we generate GraphQL schema for all the API endpoints in RestBench, except for those whose output schema is never required. We use the `import curl` command from the StepZen CLI to generate the GraphQL schema for the endpoints using appropriate dummy values for the parameters if required. The schema files thus generated are collated to form the combined schema for a given sample (sequence of API calls) in RestBench.

Function Signature Generation We programmatically generated function signatures in the OpenAI compatible format used by Gorilla OpenFunctions (Patil et al., 2023) and the Berkeley Function Calling LeaderBoard (Yan et al., 2024) by parsing the OpenAPI specifications for Spotify and TMDB available in RestBench.

API Function Calling We then manually generated the code to call the APIs, where each API is encapsulated by a function named as the Query type corresponding to the API in the GraphQL schema, and the arguments of the function are the API parameters (which may be in the path, the query string or the body of the REST API call). Some arguments are required whereas others are optional as per the OpenAPI specification. In the ground truth code that we generated, we considered only the required arguments and ignored the optional ones. The generated code is organized as a sequence of function calls along with variables to store the function outputs.

Data Organization

Each sample of GraphQLRestBench consists of (1) a natural language utterance from a sample of RestBench, (2) the function signatures of the ground truth APIs in the sample, (3) the combined GraphQL schema of these APIs, and (4) the ground truth code to call these APIs as functions.

split	overall	spotify	tmdb
train	107	38	69
val	16	6	10
test	32	12	20

Table 1: Number of samples in each data split of GraphQLRestBench

Data Splits We split both Spotify and TMDB data from GraphQLRestBench into train, validation and test splits in the ratio 7:1:2. The corresponding splits from the two domains are combined to form the overall train, validation and test splits. Statistics of the data are shown in Table 1.

4 Experiments

We report results on our test data, benchmarking multiple open source models, namely CodeLlama (Rozière et al., 2024), DeepSeek Coder (Guo et al., 2024) and Granite Code (Mishra et al., 2024). We demonstrate the capability of these models on our code generation task using (i) Chain-of-Thought style prompting (Wei et al., 2022) where the model reasons about the sequence of functions it must call as well as the parameter values it must use, generating additional code if necessary to extract the correct parameter values from API responses represented by GraphQL types, and (ii) ReAct style prompting (Yao et al., 2022) where the model generates code in a step by step fashion (one function call per step)

As in RestBench, our dataset contains real-world examples from two domains: Spotify (Spotify, 2024) and TMDB (TMDB, 2024). For each domain, we carefully select representative few-shot examples from the corresponding train splits to guide the model in understanding the sequence of function calls and parameter assignments required to generate the correct Python code.

Metrics We used the following metrics to evaluate performance of all the models on our test data. (1) *Arg Match (full)*: This metric measures the exact match of all the function arguments in the generated and ground truth code snippets post standardization of response variable names. It assigns a score of 1 if all the arguments of all the functions in the ground truth code snippet are also present in the generated code snippet and a score of 0 otherwise. The final score is the average of the scores over the code snippets. (2) *Arg Match (functions)*: This metric measures the exact match of all the function arguments per function post response variable name standardization. It assigns a score of 1 if all the arguments of a ground truth function call are also present in the generated function call and a score of 0 otherwise. The final score is the average of the scores over the functions. (3) *Seq Match (full)*: This metric measures the exact match of the sequence of functions in the generated

Model	Prompt Style	Test split	Arg Match (full)	Arg Match (functions)	Seq Match (full)	Seq Match (conn. subseq.)
codellama-34b-instruct	CoT	overall	0.6875	0.8051	0.9062	0.9375
deepseek-coder-33b-instruct	CoT	overall	0.7500	**0.8701**	**0.9687**	**1.0000**
granite-34b-code-instruct	CoT	overall	**0.7812**	**0.8701**	0.9375	0.9687
codellama-34b-instruct	ReAct	overall	0.7188	0.8182	0.9062	0.8750
deepseek-coder-33b-instruct	ReAct	overall	0.7500	0.8312	0.9375	0.8438
granite-34b-code-instruct	ReAct	overall	**0.7812**	0.8571	0.8750	0.8750
codellama-34b-instruct	CoT	spotify	**0.5833**	**0.7741**	0.9166	0.9166
deepseek-coder-33b-instruct	CoT	spotify	**0.5833**	**0.7741**	**1.0000**	**1.0000**
granite-34b-code-instruct	CoT	spotify	0.5000	0.7096	0.9166	0.9166
codellama-34b-instruct	ReAct	spotify	0.4167	0.7097	0.8333	0.7500
deepseek-coder-33b-instruct	ReAct	spotify	0.5000	0.7419	**1.0000**	0.7500
granite-34b-code-instruct	ReAct	spotify	0.5000	0.6774	0.8333	0.8333
codellama-34b-instruct	CoT	tmdb	0.7500	0.8260	0.9000	0.9500
deepseek-coder-33b-instruct	CoT	tmdb	0.8500	0.9347	0.9500	**1.0000**
granite-34b-code-instruct	CoT	tmdb	**1.0000**	**1.0000**	**1.0000**	**1.0000**
codellama-34b-instruct	ReAct	tmdb	0.9000	0.8913	0.9500	0.9500
deepseek-coder-33b-instruct	ReAct	tmdb	0.9000	0.8913	0.9000	0.9000
granite-34b-code-instruct	ReAct	tmdb	0.9500	0.9783	0.9000	0.9000

Table 2: Few-shot Chain-of-Thought (CoT) and ReAct prompting results on the test split of GraphQLRestBench.

and ground truth code snippets. It assigns a score of 1 if the two sequences match and a score of 0 otherwise. The final score is the average of the scores over the code snippets. (4) *Seq Match (connected subsequences)*: A connected subsequence is a sequence of function calls that are dependent because of input-output variable mapping. We can extract all such connected subsequences from a code snippet by matching the input and output variable names. This metric measures the exact match of these connected subsequences in the generated and ground truth code snippets. It assigns a score of 1 if all the connected subsequences match and a score of 0 otherwise. The final score is the average of the scores over the code snippets. This metric is more robust than *Seq Match (full)* since functions can be called in any order so long as they are not dependent on each other.

Models We used three open-source code LLMs available on Hugging Face, codellama-34b-instruct (Meta), deepseek-coder-33b-instruct (DeepSeek), and granite-34b-code-instruct (IBM). We also experimented with gorilla-openfunctions-v2 but the results were very poor.

Experimental Setup For the few shot learning setting, we prompt models using greedy decoding and a temperature setting of 0.05. We use 3-shot prompting for CodeLLama and DeepSeek Coder (which have 16K context length) for Chain-of-Thought and ReAct prompting. In case of Granite Code (which has 8K context length), some adjustments were needed: (i) for CoT, only 2-shot prompts were used due to limited context length, and (ii) for ReAct, the function descriptions were stripped out from the function specs (this saves context length but slightly affects performance).

Results

We compare the few-shot performance of the three LLMs in Table 2. We see that in the overall test split, Deepseek Coder is generally the best model, while Granite Code performs better for *Arg Match (full)*. CodeLLama and DeepSeek Coder perform better on Spotify data while Granite Code performs better on TMDB data. We see that *Seq Match (connected subsequences)* is generally higher than *Seq Match (full)*, indicating that models can generate independent functions in an arbitrary order, but they are less likely to generate dependent functions in the wrong order since it would result in incorrect code. We observe that ReAct performs better than Chain-of-Thought for CodeLLama on TMDB data which also affects overall scores.

Conclusion

In this paper, we introduce GraphQLRestBench, a new benchmark for evaluating sequential function calling performance of Large Language Models (LLMs). GraphQLRestBench leverages GraphQL schema for input-output variable mapping and code generation. We propose new metrics that better evaluate sequential function calling and evaluate open source code LLMs using few shot Chain-of-Thought and ReAct style prompting on this dataset.

Limitations and Ethical Statement

In this section, we briefly highlight the limitations and ethical considerations of our work. This work suffers from three major limitations:

- RestBench is a relatively small dataset, consisting only of two domains (Spotify and TMDB). Since our dataset is based on RestBench, it is also small in size. It is difficult to fine-tune LLMs effectively on this data.
- The function calls are currently not executable. In future we would like to add the execution functionality in the evaluation framework.
- We did not evaluate the performance of state of the art closed source models like GPT-4 (Achiam et al., 2023) or Claude 3 (Anthropic, 2024), preferring instead to evaluate open source models. While these open source models are quite good, they do not match the performance of the closed source models.

Ethical Considerations

In this work, we have used publicly available datasets and open source Large Language Models. There are mentions of names of people and organizations in the dataset. While this can be considered innocuous data about well known people, we do not know if the organisations that produced and released these datasets offered options for people to opt out.

Our work proposes methods to use LLMs for function calling, namely generating functions from natural language instructions given function specifications and GraphQL schema generated from REST APIs. Function calling is a well known task. Several datasets and leaderboards exist for this task. However, the potential for a malicious user or organization using this kind of work for exploiting vulnerabilities in REST APIs does exist.

Such exploitation of vulnerabilities could lead to leak of sensitive data from API services and could generally be used for distributed denial of service attacks. While such attacks can be carried out by malicious users coding themselves, LLMs could help scale such attacks. But this kind of misuse of LLMs is possible with all code models. The ability to generate code using natural language in general and our contribution here to the particular aspect of function calling can be used by malicious users but is generally useful to a much larger population who use it for good and productive reasons.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

AI Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*.

Apollo Graph Inc. 2024. Apollo GraphQL. https://www.apollographql.com/. Accessed: 2024-06-15.

Gleison Brito and Marco Tulio Valente. 2020. Rest vs graphql: A controlled experiment. In *2020 IEEE international conference on software architecture (ICSA)*, pages 81–91. IEEE.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253*.

Carles Farré, Jovan Varga, and Robert Almar. 2019. Graphql schema generation for data-intensive web apis. In *Model and Data Engineering: 9th International Conference*, pages 184–194.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence.

Hasura. 2024. Hasura graphql engine. https://hasura.io/. Accessed: 2024-06-15.

IBM. 2024. StepZen GraphQL as a Service. https://stepzen.com/. Accessed: 2024-06-15.

Facebook Inc. 2015. Draft rfc specification for graphql. http://spec.graphql.org/July2015/.

The OpenAPI Initiative. 2021. Openapi specification. https://spec.openapis.org/oas/latest.html.

Li Li, Wu Chou, Wei Zhou, and Min Luo. 2016. Design patterns and extensibility of rest api for networking applications. *IEEE Transactions on Network and Service Management*, 13(1):154–167.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2023. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*.

Grégoire Mialon, Roberto Dessi, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Roziere, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *Transactions on Machine Learning Research*.

Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, Manish Sethi, Xuan-Hong Dang, Pengyuan Li, Kun-Lung Wu, Syed Zawad, Andrew Coleman, Matthew White, Mark Lewis, Raju Pavuluri, Yan Koyfman, Boris Lublinsky, Maximilien de Bayser, Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Yi Zhou, Chris Johnson, Aanchal Goyal, Hima Patel, Yousaf Shah, Petros Zerfos, Heiko Ludwig, Asim Munawar, Maxwell Crouse, Pavan Kapanipathi, Shweta Salaria, Bob Calio, Sophia Wen, Seetharami Seelam, Brian Belgodere, Carlos Fonseca, Amith Singhee, Nirmit Desai, David D. Cox, Ruchir Puri, and Rameswar Panda. 2024. Granite code models: A family of open foundation models for code intelligence.

Shishir G Patil, Tianjun Zhang, Vivian Fang, Roy Huang, Aaron Hao, Martin Casado, Joseph E Gonzalez, Raluca Ada Popa, Ion Stoica, et al. 2024. Goex: Perspectives and designs towards a runtime for autonomous llm applications. *arXiv preprint arXiv:2404.06921*.

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*.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*.

Spotify. 2024. Spotify. http://spotify.com/. Accessed: 2024-06-15.

TMDB. 2024. The movie db. https://www.themoviedb.org/. Accessed: 2024-06-15.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Erik Wittern, Alan Cha, and Jim A Laredo. 2018. Generating graphql-wrappers for rest (-like) apis. In *International Conference on Web Engineering*, pages 65–83.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2024. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.