ToolReAGt: Tool Retrieval for LLM-based Complex Task Solution via Retrieval Augmented Generation

Norbert Braunschweiler, Rama Doddipatla, Tudor-Catalin Zorila
Toshiba Europe, Cambridge, Cambridgeshire, UK
{norbert.braunschweiler, rama.doddipatla, catalin.zorila}@toshiba.eu

Abstract

Artificial intelligence agents when deployed to solve complex problems, need to first decompose the task into smaller manageable sub-tasks, and further associate tools if one is required to solve the sub-task. If the size of the set of tools to chose from is large, a retrieval system is usually employed to narrow down the tool choices before the LLM can proceed with associating tools to the sub-tasks. This paper focuses on the retrieval problem to identify the set of relevant tools to solve a complex task given a large pool of tools to chose from using retrieval augmented generation (RAG) and we refer to it as *ToolReAGT*. The proposed approach employs *ReAct* prompting to perform the retrieval in an iterative fashion to first identify if a tool is required and then associate one or more tools for each intermediate step, also referred to as a sub-task. This deviates from conventional RAG where an n-best list of tools are identified given the complex task directly. Experiments are presented on the *UltraTool* benchmark corpus with 1000 complex tasks and over 2000 tools to select from. A conventional RAG-system is established as baseline and compared to the *ToolReAGt* approach, resulting in an 8.9% improved retrieval accuracy score recall@5.

1 Introduction

The ability of current AI systems utilizing the language understanding and reasoning capabilities of LLMs in combination with individual tools designed for solving specific tasks, has greatly expanded the application range and capacity of these systems (Parisi et al., 2022; Qin et al., 2023b; Schick et al., 2023; Qin et al., 2023a; Patil et al., 2023; Li et al., 2024; Wang et al., 2024; Kong et al., 2024; Qu et al., 2025). Tools which can provide tailored information such as up-to-date temperature measurements, compute mathematical equations or identify objects in images can each contribute to solve an overarching complex task and result in powerful AI agents. However, the efficacy of these systems in completing complex tasks heavily relies on their ability to accurately select appropriate tools for solving individual sub-tasks.

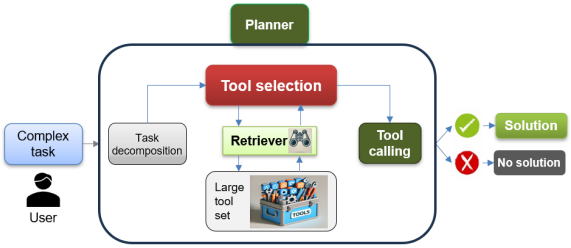

Figure 1: Importance of tool selection quality in complex task solution AI system.

Figure 1 illustrates the flow diagram of a system that can solve a complex task. The *Planner* is internally made of a *Task Decomposition* module that decomposes the complex task into smaller manageable sub-tasks, the *Tool Selection* module can dive into a database of tools and retrieve the relevant tools that can solve the sub-tasks and a *Tool Calling* module that can compose and call the retrieved tools in a specific order to solve the complex task, which we refer to as a solution. Every module can introduce dependencies that the other modules need to adhere to in deriving a solution to solve the complex task. In this paper, we focus on the *Tool Selection* module and investigate tool retrieval in depth, with the aim to support the planner in solving complex tasks. The main challenges in tool retrieval are: a) understanding the requirements of the task to be solved and formulating an adequate query to find a suitable tool, b) comprehending the functionality of a tool from its description, and c) ability to distinguish between similar tools to choose the most suitable one.

In this paper, we introduce a training-free Retrieval-Augmented Generation (Lewis et al.,

2021) architecture called *ToolReAGt*. The proposed approach employs *ReAct* prompting to enhance retrieval using iterative refinement of the prompts. We will also present investigations on the importance of context information when solving a complex task. We present our investigations using the *UltraTool* (Huang et al., 2024) benchmark corpus that has over 2000 tools to choose from and compare with traditional RAG approaches as well as more recent iterative based approaches that also involve training the retriever. We show through experiments on the *UltraTool* corpus that the proposed training free approach can outperform existing methods. The rest of the paper is organised as follows: An overview of related work is presented in the next section, followed by the description of the conventional RAG system and the *ToolReAGt* model. Then, the benchmark corpus is described and the set-up of the evaluation which is followed by the presentation of results and their discussion, and finally conclusions.

2 Related work

A simple and straight forward approach to selecting the relevant tools is to provide all the tool description in the prompt (Yuan et al., 2024; Mu et al., 2024; Du et al., 2024), which can be further combined with fine-tuned retrieval systems (Qin et al., 2023b; Hao et al., 2023; Gao et al., 2023). But a major limitation of these methods is when the pool size of the tools to chose from increases drastically, that limits to include all the tool descriptions into the prompts. A potential solution is to first build a smaller pool of relevant tools using RAG and then proceed to solving the complex task, which is one of the motivations for the method presented in this paper.

In Zhang et al. (2024), the authors propose to leverage reinforcement learning to enhance the alignment between user queries and tools in LLMs. This method focuses on retrieving n-best tools related to the query using query re-writing. It also requires training the retriever using reinforcement learning. In contrast the proposed method follows an iterative prompt refinement and is more focused on solving a complex task, where tools are retrieved in a step-by-step fashion. Also, we follow a training free approach.

An adaptive truncation of retrieval results is presented in Zheng et al. (2024) which treats seen and unseen tools differently to ensure more relevant tools are prioritized. Additionally, it introduces a hierarchy-aware reranking which refines retrieval results by concentrating them for single-tool queries and diversifying them for multi-tool queries. While the adaptive truncation method effectively manages unseen tools, our method explores the use of the *ReAct* framework that inherently performs re-ranking of the relevant tools, but also explores the use of a varied context during retrieval at (a more fine-grained) sub-task level.

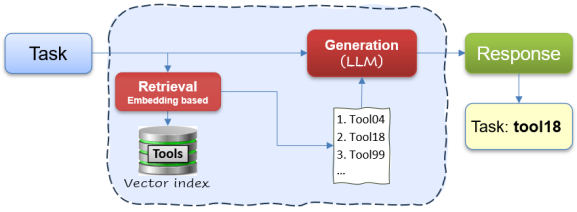

Figure 2: Conventional RAG architecture for tool selection.

An approach in which fine-tuned LLMs are used to capture relationships between user queries and tool descriptions is introduced in Qu et al. (2024). The method constructs bipartite graphs among queries, scenes, and tools, and it uses a dual-view graph collaborative learning framework to capture intricate collaborative relationships among tools. In this work, we assume that the planner is looking into the relations between tools, where the tool retriever is one of the components of the planner. This is done in a training free fashion and it should generalise to unseen tasks.

In Xu et al. (2024), authors propose iterative LLM feedback to improve tool selection, but use a trained dense retriever without the RAG-specific generation part. In our method we avoid training the retriever and the iterative refinement is done through a *ReAct-agent*.

3 Methods

We will first present the general RAG architecture and how it can be employed to perform tool retrieval, which we will refer to as *Conventional RAG* and is used as a baseline in our study. Further we introduce the proposed *ToolReAGT* method and present the design changes that are introduced contrasting with the conventional RAG.

3.1 Conventional RAG

Figure 2 illustrates the basic RAG-architecture for tool retrieval and includes two main components: the retriever and generator. Given a task as input

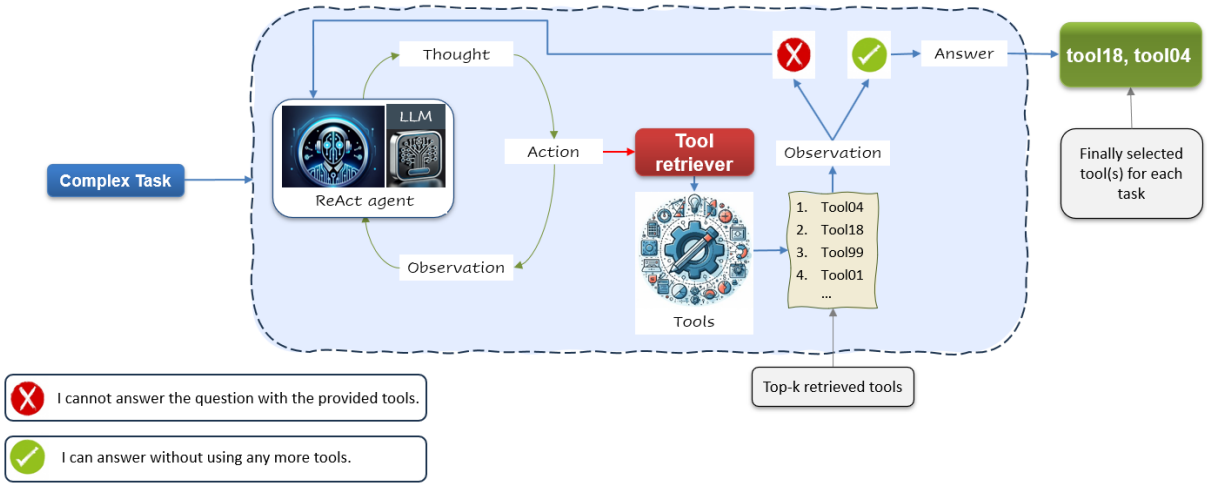

Figure 3: Block diagram of proposed *ToolReAGt* system.

query, the retriever, typically leveraging an embedding based dense retriever, retrieves a subset of tools. A distance measure (for example the cosine similarity) is computed between the query embedding and the tool embeddings to retrieve the relevant tools from the vector index. The list of retrieved tools is propagated into the prompt for the generation module (including the task and instructions) to output the final response, i.e. tool(s) selected for the given query.

3.2 ToolReAGt

Figure 3 illustrates the proposed *ToolReAGt* architecture. Input can consist of a complex task and optionally task decomposition into sub-tasks (provided by a planner which is not part of the *ToolReAGt* system). The *ReAct*-agent can flexibly operate with only the complex task as input and decompose into the intermediate steps by itself or, if provided, utilize a given task decomposition from a planner. The *ToolReAGt* system is guided by a *ReAct-agent* leveraging a sequence of *Thought->Action->Observation* steps (see Appendix B). Firstly, the system calls the tool retriever with a tailored query for the given sub-task (or internal decomposition step) and retrieves a $top\_k = \{t_1, t_2, ..., t_k\}$ list of tools. The *ReAct-agent* is instructed to always call the retriever as a mandatory tool, to provide it with a list of relevant candidates from the set of available tools. The retrieved tool list is then inspected in the *Observation* step and depending on whether the system decides that at least one of the retrieved tools is suitable to solve the sub-task (or internal decomposition step) it will proceed to give the answer (indicated by the green tick), i.e. either one tool or multiple tools for each sub-task, or otherwise enter another iteration including a new tool retrieval call.

By using the *ReAct* technique the complex prompt can be interpreted by the LLM and a more targeted question is formulated for finding the best tool for the current sub-task (or intermediate step). Conversely, the conventional RAG employs the input prompt only. Then, the *Thought->Action->Observation* loop can be executed until either a suitable tool has been found or the max number of iterations (set to 10) is reached. As such, an iterative refinement can take place which is likely to be beneficial for both tool retrieval and the final generation output.

4 Data

For evaluation, we used the *UltraTool* (Huang et al., 2024) corpus which provides a rich number of complex tasks (5824) from 22 domains (e.g. finance, travel, documents, etc.) with a large tool set of 2032 tools. The corpus comes divided into a test set (1000 tasks) and a development set (4824 tasks). For evaluation we employ the 1000 tasks test set which consists of 436 tools (TEST-436). In addition, we perform evaluation on the same test set using the full (test+dev) 2032 tool set (TEST-2032).

The test tasks include an average number of 2.4 tools per task with the following distribution: 1: 188, 2: 496, 3: 205, 4: 83 and the remainder requiring >5 tools (up to a maximum of 10 tools). A major reason to choose *UltraTool* benchmark is that, it has annotations with reference solution plans, i.e. decomposition into sub-tasks, including tool-requiring sub-tasks and their respective tools,

enabling objective evaluation via retrieval metrics. This helps us to investigate how the retrieval performance can vary when task decomposition from planner is available apriori.

Solution plans contain on average 12.1 steps which means there is a high proportion of tool-free steps. For brevity, we chose the English version of the corpus which was originally collected in Chinese. Contrary to *UltraTool's* original evaluation methods which encompass *planning*, *tool creation* and *tool usage*, we are using the corpus for evaluating tool selection performance of RAG-systems with and without using the provided sub-tasks.

5 Evaluation setup

The experiments conducted in this study employ the LlamaIndex-framework[1] for implementing the RAG pipeline. The basic workflow to create a RAG-system contains the preparation of source data from which information will be retrieved (typically in the form of documents, i.e. tool descriptions here), ingesting this data into a vector index leveraging an embedding model, defining a query agent together with an LLM, and formulating input prompts. These steps will be described next.

5.1 Tool representation

Tools provide specific functionalities such as currency conversions or getting up to date weather information and are crucial helpers in a system designed to combine their abilities for solving more complex tasks. As such, understanding tool functionalities, including their required input parameters and their generated output, is essential for successful tool selection. In *UltraTool*, tools are described in the widely used JSON-format and include "name", "description", "arguments" (type and format of input(s)) and "results" (type and format of output(s)). Considering each tool as a separate entity, each of the 2032 tool descriptions was stored in a separate file named with the tool name (e.g. "check_weather.json"). The following shows an example of a tool description for the *check_weather* tool which provides weather information such as temperature and precipitation probability, for a given location and a specific date:

"name": "check_weather", "description": "Check the weather forecast for a specified date and location", "arguments": {"type": "object","properties": {"date": } {"type": "string", "description": "Specified date"}, "location": {"type": "string", "description": "Weather query location"}}}, "results": {"type": "object", "properties": {"weather_status": {"type": "string", "description": "Weather condition"},}, "temperature": {"type": "string", "description": "Temperature"}, "precipitation": {"type": "string", "description": "Probability of precipitation"}, "weather_info": {"type": "string", "description": "Weather forecast information"}, "suggestions": {"type": "string", "description": "Suggestions based on weather conditions"}}}

5.2 Vector index

A vector index is an essential component in a RAG-system providing an efficient way of retrieving relevant information from a potentially large amount of data to enhance the responses generated by the LLM. In the current study, the vector indices were built on the tool descriptions in JSON-format. Two different vector indices were built: one based on all the 2032 tools and another using the subset of 436 tools which appear in the test set. This was intended to shed some light on the impact of tool corpus size on retrieval performance. Vector indices were created by converting tool descriptions into high-dimensional vectors of dimension 768 using the *bge-base-en-v1.5* [2] embedding model. Additional information in the form of metadata, such as data classes or file name, can be attached to each tool description which can support retrieval. For our vector indices the file name was added as metadata because it included the unique tool name which was deemed to be helpful for retrieval. For ingesting tool descriptions into the vector index, the text was split with a token text splitter using a chunk size of 512 tokens and a chunk overlap of 128 tokens.

5.3 Impact of input information

Different prompt types were created to evaluate the impact of input information on the retrieval performance. For the first one, no sub-task decomposition information is provided, while for the other ones, the various levels of information from the corpus are included:

- `plain_fulltask`: full task without sub-task decomposition
- `subtask`: only the sub-task that is annotated in the corpus without full task decomposition
- `subtask+fulltask`: `subtask` + `plain_fulltask` , but no full sub-task

[1] https://docs.llamaindex.ai/en/stable/

[2] https://huggingface.co/BAAI/bge-base-en-v1.5

Data	Input	Retrieval			
		R@1	R@2	R@5	R@10
TEST-2032	plain_fulltask	16.8	29.2	51.1	66.6
	subtask	4.6	6.8	12.6	18.2
	subtask+fulltask	14.9	28.4	44.1	57.5
	fulltask+decomp	20.7	36.5	59.4	76.8

Table 1: Impact on tool retrieval performance using Conventional RAG with varied contextual information

decomposition

- `fulltask+decomp`: `plain_fulltask` with full sub-task decomposition

Examples for each of the prompts used in the experiments are provided in Appendix A.

The LLMs used for evaluation are the 8-bit quantized GGUF-version of the Mistral-7b-instruct[3] LLM, i.e. *mistral-7b-instruct-v0.2.Q8_0* from Huggingface [4] and the 4-bit quantized GGUF-version of the *Mistral-Large-Instruct-2411*[5] model which are publicly available for research and provide a 32k and 128k tokens context window respectively. The *LamaCPP*[6] library was employed to run the LLM. Experiments were run on four NVIDIA A100 GPUs with 80GB of memory.

6 Results

6.1 Evaluation metrics

Tool retrieval accuracy can be measured either at the output of the *Retriever* or at the output of the *Generator* stages of RAG-system. *UltraTool* provides reference tools for each sub-task, making evaluation straightforward, by checking if the list of retrieved tools at different top_k values includes the reference tools.

To measure the tool retrieval performance at the output of the *Retriever*, the $recall@N$ metric was chosen (see equation 1), where $N = 1, 2, 5, 10, 20$ indicate whether the required tool was selected in the top_N retrieved tools.

$$\text{Recall@N} = \frac{\text{Number of relevant tools retrieved in top\_N}}{\text{Total number of relevant tools}} \quad (1)$$

For the *Generator* stage, we report accuracy, i.e. how often the searched for tool was actually chosen from the list of tools provided by the *Retriever*. In our evaluations, we report results where the *Generator* was forced to output only one tool for each tool requiring sub-task or it was given the freedom to choose multiple tools to be associated with the same sub-task.

To compare the performance of the proposed method with the method in (Xu et al., 2024), we also report the Normalized Discounted Cumulative Gain (NDCG@k) (Järvelin and Kekäläinen, 2002) metric.

We also checked the impact of multiple runs upon retrieval scores and found that there was no variation in retrieval scores in the baseline RAG-system when prompts were kept identical. For the variation in generation accuracy, the baseline RAG-system showed no measurable variation and the *ToolReAGt* system showed marginal variation in both retrieval (average StdDev: 0.186) and generation scores (average StdDev: 0.212).

6.2 Discussion

To remind, the *UltraTool* corpus comes with annotation of task decomposition for the complex task and has annotations about which of the sub-tasks require a tool. We will use this additional knowledge in the prompt to understand the impact on the retrieval using Conventional RAG and Mistral-7B model. We measure the retrieval scores by a) only presenting the full task (*plain_fulltask*) description without any sub-task decomposition, b) only presenting the sub-task (*subtask*) that requires a tool without any additional context c) presenting the full task description along with only a single sub-task *subtask+fulltask*, and d) presenting the full task description along with the complete task decomposition *fulltask+decomp*.

The results are presented in Table 1. One can observe that providing *subtask* information in isolation without the complete task decomposition seems to perform inferior than just using the *plain_fulltask* as input. On the other hand, when using the full task along with the complete task

[3] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

[4] https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.2-GGUF

[5] https://huggingface.co/mistralai/Mistral-Large-Instruct-2411

[6] https://github.com/ggerganov/llama.cpp

Input	Output	ConvRAG		ToolReAGt	
		R@5	Acc	R@5	Acc
plain_fulltask	by full task, single	74.2	68.4	77.4	73.5
fulltask+decomp	by sub-task, single	81.7	71.9	90.6	73.7
fulltask+decomp	by sub-task, multi	81.7	74.5	90.5	87.4

Table 2: Results for tool selection contrasting Conventional RAG and ToolReAGT on TEST-436

decomposition (*fulltask+decomp*) it seems to improve the retrieval performance. This indicates that having access to complete task decomposition on how to solve the complex task should help the retriever identify the correct tools better than when presented only with the complex task description as input, which is intuitive. One can also observe that retrieving more tools at each intermediate step can also boost the retrieval performance. These initial investigations were performed on TEST-2032. Moving forward, all the results will be presented on the actual test set of the *UltraTool* corpus and is referred to as TEST-436.

In Table 2, we contrast the performance of the *Conventional RAG* with the proposed *ToolReAGt* described in Section 3. We report both the retrieval (R@5) and generation performance in this table. Variations in the input prompts and how many tools the generator should output (either single or multiple) can further influence the retrieval performance. The *plain_fulltask* refers to providing only the complex task description as input without any decomposition. By doing so, we can measure how the LLM will handle the complex task and assign relevant tools without any additional information. This is used as a baseline to understand the impact of any variations that we might introduce either into the input prompt in the form of additional context or apply the *ReAct* prompting or change the output of the generator, which are all presented in this table. Comparing the RAG baseline and the *ToolReAGt* system for the *plain_fulltask* input shows that the *ToolReAGt* system achieves a 3.4% better R@5 and an increased accuracy in generation (+5.1%).

System	Retrieval		
	N@1	N@3	N@5
ToolRetriever(Xu et al., 2024)	48.2	47.7	53.0
Xu-et-al(Xu et al., 2024)	49.3	47.5	54.3
RAGbaseline [plain_fulltask]	54.8	59.2	66.3
ToolReAGt [plain_fulltask]	60.6	63.8	69.3

Table 3: Results using NDCG metric comparing different retrieval methods on the *UltraTool* TEST-436

The addition of contextual information by adding the decomposition of the complex task into individual sub-tasks including the information which sub-tasks require tools, increased both retrieval and generation scores for both systems, with an increase in R@5 of 7.5% for the RAG baseline and a boost of 13.2% for the *ToolReAGt* system, leading to an 8.9% absolute improvement for *ToolReAGt*, while the increases in generation accuracy were smaller, indicating that the improved retrieval scores did not directly propagate into improved generation accuracy. However, by asking the systems to select more than one tool in the generation output ("By sub-task, multi") both systems achieve higher accuracy, but the *ToolReAGt* system shows a much higher improvement than the baseline system, i.e. RAG baseline +2.6% and *ToolReAGt* +13.7%, indicating that it is capable to transfer more relevant tools also in the generation output.

Table 3 presents the retrieval results measured using Normalized Discounted Cumulative Gain (NDCG@k) (Järvelin and Kekäläinen, 2002) with $k = \{1, 3, 5\}$, comparing the retrieval method of Xu et al. (2024) in literature with the proposed *ToolReAGT*. *ToolRetriever* introduces a model that has been trained on the ToolBench corpus (Qin et al., 2023b) and corresponds to *out-of-domain* evaluation on the *UltraTool* benchmark as reported in Xu et al. (2024). For fair comparison to the results presented in Xu et al. (2024), the performance of *ToolReAGT* using only the *plain_fulltask* as input without sub-task decomposition is presented here. It is surprising that RAGbaseline already surpasses the performance of Xu et al. (2024). The *ToolReAGt* method achieves the highest NDCG scores across all k-values.

7 Conclusion

The paper presented a training free Retrieval-Augmented Generation architecture called *ToolReAGt* to improve tool retrieval performance in the framework of solving complex tasks. The proposed approach employed *ReAct* prompting to perform

an iterative and targeted retrieval of tools, is able to run with and without given task decomposition and showed that the retrieval performance improved on the *UltraTool* benchmark. It is also clearly evident that having access to the task decomposition in advance can greatly benefit the retriever in identifying the relevant tools. Results showed the advantage of the proposed approach against Conventional RAG as well as against other methods in literature that also followed an iterative approach to solving tool retrieval.

Limitations

ToolReAGt is motivated to solve complex tasks and when faced with a large tool set to choose from. The study in this paper is limited to investigate the performance of retriever in depth. It will be interesting to evaluate the task completion performance as a whole where the retriever should support the planner in deriving the correct solution. We believe this will be a natural extension and will form the course for our future work.

In terms of run time, *ToolReAGT* is slower than conventional RAG architectures due to its iterative design and reliance on a 123B-parameter LLM, which demands significantly more computational resources which might have to be taken into account for practical use cases. Investigating the reliance on a smaller LLM and efficiently terminating the iterative loop of *ReAcT* is something that has not been explored in the current work.

References

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

Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, and Jun Ma. 2023. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *AAAI Conference on Artificial Intelligence*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *ArXiv*, abs/2305.11554.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. *ArXiv*, abs/2401.17167.

Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20:422–446.

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

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Preprint*, arXiv:2005.11401.

Zhi Li, Yicheng Li, Hequan Ye, and Yin Zhang. 2024. Towards autonomous tool utilization in language models: A unified, efficient and scalable framework. In *International Conference on Language Resources and Evaluation*.

Feiteng Mu, Yong Jiang, Liwen Zhang, Chu Liu, Wenjie Li, Pengjun Xie, and Fei Huang. 2024. Query routing for homogeneous tools: An instantiation in the rag scenario. *Preprint*, arXiv:2406.12429.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. Talm: Tool augmented language models. *ArXiv*, abs/2205.12255.

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

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shi Liang, Xingyu Shen, Bokai Xu, and 22 others. 2023a. Tool learning with foundation models. *ArXiv*, abs/2304.08354.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023b. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789.

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

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

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761.

Hongru Wang, Yujia Qin, Yankai Lin, Jeff Z. Pan, and Kam-Fai Wong. 2024. Empowering large language models: Tool learning for real-world interaction. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2983–2986, New York, NY, USA. Association for Computing Machinery.

Qiancheng Xu, Yongqing Li, Heming Xia, and Wenjie Li. 2024. Enhancing tool retrieval with iterative feedback from large language models. *ArXiv*, abs/2406.17465.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *ArXiv*, abs/2401.06201.

Yuxiang Zhang, Xin Fan, Junjie Wang, Chongxian Chen, Fan Mo, Tetsuya Sakai, and Hayato Yamana. 2024. Data-efficient massive tool retrieval: A reinforcement learning approach for query-tool alignment with language models. In *Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, SIGIR-AP 2024, page 226–235, New York, NY, USA. Association for Computing Machinery.

Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. 2024. ToolRerank: Adaptive and hierarchy-aware reranking for tool retrieval. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 16263–16273, Torino, Italia. ELRA and ICCL.

A Prompts

The following lists examples of prompts used as input to the RAG-systems evaluated in this paper:

plain_fulltask Only the full task for which tools had to be retrieved and no other information. Example: "Given the following complex task: "I need you to help me create a file called 'Work_Tasks.txt' on the desktop, and then write 'Preparation for Tomorrow's Meeting' into it." and the list of tools in the context, select the best tools to solve the complex task."

sub-task Only the sub-task for which a tool had to be retrieved. Example: "Given the following task: "step": "1.2 Use file writing tool to create and write content" , select the best tool provided in the context to solve the task. For an example this could be step "1.2 Query the current exchange rate", and the response format would then be: [{"step": "1.2 Query the current exchange rate", "tool": "currency_exchange_rate"}]}. Provide your answer exactly in the same format as in the example and do not add anything else."

+fulltask sub-task plus full task, but no task decomposition. Example: "Given the following task: "I need you to help me create a file called 'Work_Tasks.txt' on the desktop, and then write 'Preparation for Tomorrow's Meeting' into it.", select the best tool provided in the context to solve the following substep: ["step": "1.2 Use file writing tool to create and write content"]. For an example this could be step "1.2 Query the current exchange rate", and the response format would then be: [{"step": "1.2 Query the current exchange rate", "tool": "currency_exchange_rate"}]}. Provide your answer exactly in the same format as in the example and do not add anything else."

(fulltask)+decomp, single sub-task, full task, and task decomposition; single tool output in generation. Example: "Given the following task: "I need you to help me create a file called 'Work_Tasks.txt' on the desktop, and then write 'Preparation for Tomorrow's Meeting' into it." and its decomposition into sub-tasks here: [{"step": "1. Create file"}, {"step": "1.1 Get file creation information (File path: Desktop/Work_Tasks.txt, File content: Preparation for Tomorrow's Meeting)"}, {"step": "1.2 Use file writing tool to create and write content", "tool": ""}, {"step": "1.3 Confirm file creation and content writing success"}] , select the best tool provided in the context to solve the following substep: ["step": "1.2 Use file writing tool to create and write content"]. For an example this could be step "1.2 Query the current exchange rate", and the response format would then be: [{"step": "1.2 Query the current exchange rate", "tool": "currency_exchange_rate"}]}. Provide your answer exactly in the same format as in the example and do not add anything else."

fulltask+decomp, multi sub-task, full task, task decomposition; allowing multiple tools in generation. Example: "You are an expert in selecting tools to solve a given task. The task is typically a sub-task of a more complex task and you are given the complex task, its decomposition into sub-tasks and the sub-task you are asked to select tools for by calling the "ultratools_json_tools" tool with a suitable query. So here is the complex task: "I need you to help me create a file called 'Work_Tasks.txt' on the

desktop, and then write 'Preparation for Tomorrow's Meeting' into it." and its decomposition into sub-tasks: [{"step": "1. Create file"}, {"step": "1.1 Get file creation information (File path: Desktop/Work_Tasks.txt, File content: Preparation for Tomorrow's Meeting)"}, {"step": "1.2 Use file writing tool to create and write content", "tool": ""}, {"step": "1.3 Confirm file creation and content writing success"}]. Given this context, and the list of tools provided to you by calling the "ultratools_json_tools"-tool, select the best tools to solve the following substep: ["step": "1.2 Use file writing tool to create and write content"]. For an example this could be step "1.2 Query the current exchange rate", and the response format would then be: [{"step": "1.2 Query the current exchange rate", "tool1": "currency_exchange_rate"}", "tool2": "currency_exchange_tool"}]. You can provide multiple tools ranked by their order of relevance when you think there are multiple tools capable to solve the task. Provide your answer exactly in the same format as in the example and do not add anything else."

B ReAct prompt template

Below is the *ReAct* prompt template provided in the LlamaIndex[7] version utilized in the experiments.

You are designed to help with a variety of tasks, from answering questions to providing summaries to other types of analyses.

Tools

You have access to a wide variety of tools. You are responsible for using the tools in any sequence you deem appropriate to complete the task at hand. This may require breaking the task into sub-tasks and using different tools to complete each sub-task.

You have access to the following tools:
{tool_desc}
{context_prompt}

Output Format

Please answer in the same language as the question and use the following format:

```
Thought: The current language of the user is: (user's language). I need to use a tool to help me answer the question. Action: tool name (one of {tool_names}) if using a tool. Action Input: the input to the tool, in a JSON format representing the kwargs (e.g. {{"input": "hello world", "num_beams": 5}})
```

Please ALWAYS start with a Thought.

NEVER surround your response with markdown code markers. You may use code markers within your response if you need to.

Please use a valid JSON format for the Action Input. Do NOT do this {{'input': 'hello world', 'num_beams': 5}}.

If this format is used, the tool will respond in the following format:

```
Observation: tool response
```

You should keep repeating the above format till you have enough information to answer the question without using any more tools. At that point, you MUST respond in one of the following two formats:

```
Thought: I can answer without using any more tools. I'll use the user's language to answer
Answer: [your answer here (In the same language as the user's question)]
```

```
Thought: I cannot answer the question with the provided tools.
Answer: [your answer here (In the same language as the user's question)]
```

Current Conversation

Below is the current conversation consisting of interleaving human and assistant messages.

[7] https://docs.llamaindex.ai/en/stable/