Self-Assist: Deliberate Tool Selection by Large Language Models

Anonymous ACL submission

Abstract

With the swift progress in tool-based learning, the number of tools available has also increased significantly. In comparison to the correct utilization of tools, the significance of precisely choosing the appropriate tool from an increasingly large selection is crucial. At present, depending solely on retrieval methods that use keywords and embeddings faces new challenges in the realm of tool selection. On one hand, it becomes difficult to distinguish between tools with similar functionalities. On the other hand, some queries require further reasoning to uncover the true tool needs, where direct matching with keywords or semantic embeddings does not yield the correct result. To address this issue, we introduce the Self-Assist method, which fully leverages the inherent knowledge and reasoning capabilities of large language models. Through a series of systematic steps, large language models actively engage in deliberate thought and select the most appropriate tool for a given query. In essence, our work champions a blend of LLMs and retrieval tools in a flexible, efficient, and universally compatible design, significantly bolstering retrieval outcomes. Evaluations on three datasets reveal superior performance over the previous approaches in retrieval accuracy and overall success.

1 Introduction

Recent advances in large language models (LLMs) introduce remarkable capabilities in natural dialogue, mathematical reasoning, and program synthesis. However, these LLMs face constraints due to their fixed weight set and limited context. By integrating specialized tools (Nakano et al., 2021b; Lazaridou et al., 2022; Paranjape et al., 2023b), LLMs can overcome several challenges and enhance the expertise across tasks beyond their original scope (Wu et al., 2023; Shen et al., 2023).

Initial efforts have been towards fine-tuning LLMs to work with specific tools (Nakano et al., 2021b; Lazaridou et al., 2022; Paranjape et al., 2023b). For example, Schick et al. (2023a) finetune a language model to solve various NLP tasks by leveraging question answering system, Wikipedia search engine, calculator, calendar, and machine translation system. Yet, this approach faces a fundamental limitation: it binds the model's ability strictly to tools learned during training, making it difficult to generalize to newer tools. As the few-shot capability of large language models intensifies, in-context-learning has emerged as one of the most promising approaches to tool learning today (Wu et al., 2023; Shen et al., 2023; Surís et al., 2023; Paranjape et al., 2023b). For instance, Wu et al. (2023) integrated ChatGPT with a variety of visual foundation models using in-context-learning.

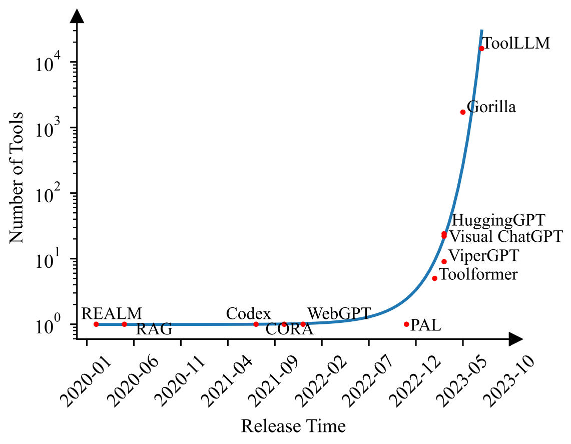

Figure 1: Temporal Evolution of Tool Invocation Counts

However, no matter finetune-based method or in-context-learning method, the prevailing focus of them lies in tool utilization, enhancing language model capacities, and addressing a myriad of specific tasks. As shown in Figure1, with an ever-expanding array of available tools and plugins, ranging from the inception of search engines to a multitude of specialized domain APIs (Lazaridou et al., 2022; Schick et al., 2023a; Patil et al., 2023; Liang et al., 2023), the selection from this burgeoning pool in response to a specific query becomes increasingly imperative. The current mainstream methodology employs an external retriever. However, common methods based on keyword and embedding retrieval can easily generate significant errors, especially when there are numerous tools with similar functionalities, or when a query requires reasoning to determine the actual tool needs, making differentiation through mere retrieval methods challenging.

In this paper, we introduce the Self-Assist method, which can fully leverages the inherent knowledge and reasoning capabilities of large language models to facilitate the tool selection. For a given query, the method undertakes the following steps: 1. Actively analyze the query to discern the requisite tool for the task. 2. Generate a general tool description tailored to address the specific query. 3. Employ the retriever using the query and tool description to search for the relevant tool. 4. Analyze the core functionalities of top-k candidate tools from the original tool documentations. 5. From the top-k candidate tools returned by the retriever, further discern the most apt match based on the query, general tool description and analysis of candidate tools. Experimental results on three public datasets demonstrate that our approach significantly outperforms the baseline in both retrieval accuracy and final success rate.

Our main contributions can be summarized as follows:

1. In addressing the issue of tool selection, to overcome the limitations of traditional retrieval methods, we propose the Self-Assist method. This approach involves large language models actively engaging in deliberate thought to select the most suitable tool, thereby fully leveraging their knowledge and reasoning prowess.

2. This method offers notable flexibility. It not only integrates smoothly as a plug-and-play feature with prevalent models, but it also prides itself on universal compatibility with a vast array of retrieval tools.

3. Empirical evaluations on three public datasets demonstrate that our method markedly enhances retrieval accuracy and the overall success rate.

2 Related work

2.1 Tool Usage in Large Language Models

Recent advancements have utilized existing tools to enhance task-specific performance across various domains. Examples include WebGPT (Nakano et al., 2021a) and ReAct (Yao et al., 2022), which improve text generation through search APIs, and PaLM-SAYCAN (Brohan et al., 2023) and PaLM-E (Driess et al., 2023), which use robotics APIs for real-world tasks. Pal (Gao et al., 2022) addresses mathematical problems using code from text inputs. ToolFormer (Schick et al., 2023b) integrates multiple APIs for NLP challenges. ART (Paranjape et al., 2023a) enhances model performance with a comprehensive toolkit for complex reasoning. Further enriching this landscape, (Patil et al., 2023) unveils APIBench, a massive dataset featuring HuggingFace, TorchHub, and TensorHub APIs. (Qin et al., 2023) collect 16,464 real-world RESTful APIs spanning 49 categories from RapidAPI Hub. Examples like AutoGPT [1] and hugginggpt (Shen et al., 2023) are at the forefront, crafting agents predicated on LLMs and thereby captivating a broad public audience and sparking vibrant discussions. Experiments with LLM-based agents are also making waves in social and gaming spheres (Wang et al., 2023; Park et al., 2023). While the highlighted techniques prioritize optimizing tool utilization for problem-solving across sectors, our focus uniquely hones in on addressing the nuances of tool selection.

2.2 Information Retrieval

Information Retrieval (IR) is now an indispensable tool in numerous applications, transitioning from traditional methods to more advanced systems. Conventional models, such as TF-IDF(Salton and Buckley, 1988), depend primarily on term matches between documents and queries. An advancement, BM25 (Robertson et al., 1995), generally yields better results than TF-IDF. However, these models grapple with issues like polysemy and synonymy. Recognizing these challenges, there's been a shift towards embeddings—dense vector representations that encapsulate word semantics. Le and Mikolov (2014) introduced unsupervised vector representations for whole texts, and works like Karpukhin et al. (2020) and Hofstätter et al. (2021) have effectively leveraged dense representa-

[1] https://github.com/Significant-Gravitas/Auto-GPT

tions for improved search and retrieval tasks. OpenAI's text-embedding-ada-002 model offers versatile embeddings suitable for a spectrum of tasks.

Furthermore, there has been a surge in attempts to harness the prowess of large language models to boost search performance, which mainly focusing on single modules such as generating pseudo-queries or ranking algorithms, for example, InPars (Bonifacio et al., 2022; Dai et al., 2022) generates pseudo-queries using large language models. Sun et al. (2023) demonstrate that guided ChatGPT and GPT-4 exhibit competitive performance of generative ranking algorithms, even outperforming supervised methods. Different from them, our approach emphasizes integrating the capabilities of large language models and retriever through in-context learning. It neither requires additional training nor is specific to a particular retrieval method, making it universally compatible for tool selection.

2.3 Self-Feedback for LLM

Recent research indicates that the LLM itself can be employed as a feedback provider (Madaan et al., 2023; Shinn et al., 2023; Gero et al., 2023). A direct approach is to evaluate the quality of its generated outputs through prompting and then harness this feedback to enhance the results. This process can be iterative, with the model persistently refining its output until it achieves a specified standard. For example, Self-Refine, as introduced by Madaan et al. (2023), offers an elegant self-correction framework. It harnesses a robust pre-trained LLM to generate outputs, provide feedback, and subsequently refine those outputs based on the received feedback. Meanwhile, Reflexion (Shinn et al., 2023) has augmented this framework by integrating the long-term memory feature. This enhancement allows the system to retain previous feedback and outputs, effectively circumventing the recurrence of past errors. In contrast to the self-feedback approaches, where the LLM iteratively refines its outputs post-generation, our method presents a novel way of leveraging large language models to enhance their problem-solving capabilities: self-assisted tool selection. Before addressing the problem, our approach prioritizes selecting the appropriate tool. On one hand, proactive decision-making proves more efficient than post-hoc correction. On the other, our Self-Assist method can be synergistically integrated with the self-feedback approach.

3 Method

Let's first agree on the abbreviations for the terms:

- T: The total tool set $T = \{t_1, t_2, \ldots, t_n\}$,
- M: The instruction about tool usage,
- q: The user-provided Query,
- R: The Retriever,
- LLM: The Large language model,
- D: The tool description generated by the LLM,
- P_q: The prompt used for query analysis and generation,
- P_f: The prompt used for analysis of candidate tools,
- P_f: The prompt used for ascertainment,
- t^*: The tool that best matches the query q,
- T_k: Top-k set of candidate tools,
- F_k: The functionalities generated by the LLM,

When faced with a multitude of tools, the contextual capabilities of language models can become constrained, thereby limiting their direct in-context learning. To address this, we have distinctly segmented the process into two phases: tool selection and tool usage. For large language models, tool selection is defined as follows: Given a query q and a collection of tools T, the method should identify and return the single most pertinent tool t^*. It's crucial to emphasize the selection of the most relevant tool over a top-k list. If the model were to return a top-k list, it would inevitably need to make an additional selection during the tool usage phase, which contradicts our goal of clear separation. To address this challenge, the current mainstream methodology utilizes an external retriever R, such as BM25.

$$t^* = R(q, T) \quad (1)$$

They uses the user's query to retrieve the tool documentations corresponding to each tool and identify the most suitable tool, which is then fed into the large language model for in-context-learning. The accuracy of this retrieval process significantly dictates the ultimate success rate. For

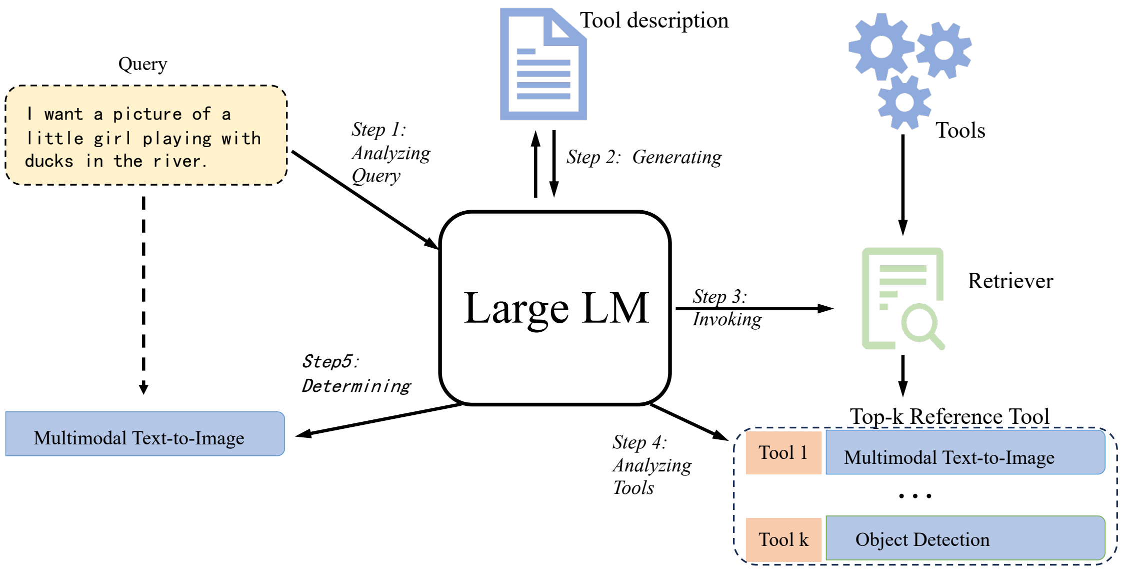

Figure 2: Systematic Workflow for Tool Selection via Large Language Model Itself. Unlike previous approaches that directly retrieve tools for a given query through a retriver, our method fully leverages the inherent knowledge and reasoning capabilities of large language models to achieve deliberate tool selection.

tasks necessitating specific tools, a failure to retrieve the correct API means even the most potent LLM cannot produce the correct outcome. Hence, enhancing retrieval accuracy is of paramount importance.

We have proposed a novel approach, self-assisted tool selection, which constructs an language agent for tool retrieval scenarios, built upon the integration of large language models and retrievers. This strategy aids the language model in achieving an autonomous tool selection process. Given a specified query, traditional paradigms often depict the language model as a passive entity, merely awaiting the feedback from the retriever to provide the pertinent tool. Contrary to this conventional approach, our framework postulates a more proactive role for the language model. Instead of being a mere recipient of the retriever's suggestions, the language model engages in an analytical contemplation, discerning the type of generalized tool that might best address the query at hand. Subsequent to this cognitive process, the model actively engages the retrieval mechanism to curate a list of suitable candidate tools. A meticulous examination of these candidates is then undertaken. Drawing upon the insights gleaned from the initial query and the subsequent analytical deliberation, the language model discerns and selects the most fitting tool from the curated set, exemplifying a synergy of active thought and dynamic retrieval in problem-solving.

Moving forward, we shall delve into a comprehensive, systematic, and formalized exposition of the methodology.

1. Query Analysis: The inception of the procedure commences with a a meticulous examination of the user's query q, facilitated by the utilization of prompt P_q. At this juncture, it is imperative for the expansive language model to comprehend the intricacies of the query and prognosticate the archetype of tool that may be requisite for the specific inquiry.

2. Tool Description Generation : Post query analysis, the emphasis shifts towards formulating a general description of the requisite tools tailored to the specific query, symbolized as D. The equation encompassing both the aforementioned stages is articulated as:

$$D = LLM(P_q, q) \tag{2}$$

3. Tool Retrieval Process: Armed with the meticulously generated tool description, the subsequent step entails invoking the retriever, denoted as R, to systematically scour for potential tools that align with both the query and the tool description. The retriever R proficiently returns a subset comprising the top-k most pertinent candidate tools, symbolically represented as T_k. The formalized representation encapsulating this phase can be mathematically articulated as:

$$T_k = R(q, D, T) \tag{3}$$

4.Analysis of Candidate Tools: Leveraging the individual documentation associated with each tool, the Large Language Model conducts a comprehensive examination of each retrieved candidate. The original API documentations might encompass specialized terminologies, detailed code snippets, among other intricacies. The primary objective of this phase is to enable the LLM to generate a succinct description of the core functionalities F_k for each candidate tool, facilitated by the utilization of prompt P_f :

$$F_k = LLM(P_f, T_k) \tag{4}$$

5.Determining of the Optimal Tool: Following the analysis F_k of the top-k candidate tools, the task gravitates towards the judicious selection of a singular, most fitting tool t^*. This decision-making process necessitates a comprehensive evaluation, factoring in both the original query, the derived tool description and the analysis of candidate tools. The process of tool selection, which facilitated by the utilization of prompt P_d, can be academically formalized as:

$$t^* = LLM(P_d, q, D, F_k) \tag{5}$$

Thus far, we have developed an langauge agent based on a large language model (LLM) that can autonomously retrieve tools. This method presents three significant advantages:

1. Leveraging the Profound Capacities of the LLM: Throughout the multifaceted stages of this paradigm, the LLM is actively engaged in discerning the nuances of both the query and candidate tools and invoking the retrieval tool. The knowledge and reasoning ability that the LLM has acquired during its pre-training phase are fully utilized.

2. Universal Retrieval Tool Compatibility: A meticulous examination of our methodology elucidates its agnostic stance towards the nature of the retriever. It manifests a harmonious interplay with retrieval mechanisms, be they anchored on traditional keyword-based methodologies or contemporary embedding techniques.

3. Plug-and-Play : Our method aligns seamlessly with the input and output of current mainstream approaches, allowing for plug-and-play integration within the entire pipeline. The responsibility for tool selection and operation can be distributed among various Large Language Models, enabling each to harness its distinct capabilities. If the LLMs designated for tool selection and tool operation are the same, we clearly can construct a more autonomous agent by appending a tool-usage step after stage 5. In this configuration, the Large Language Model, by mastering the use of a retriever tailored for tool retrieval, can markedly boost its success rate in resolving queries.

Source	Domains	Filtered Models/APIs
HuggingFace	37	925
TensorFlow Hub v2	57	696
Torch Hub	6	95

Table 1: Dataset details for APIBench.

4 Experiment

4.1 Dataset

APIBench (Patil et al., 2023) is a comprehensive benchmark built from APIs documented in HuggingFace, TorchHub, and TensorHub model cards. The benchmark collection process involved meticulously gathering and filtering online model cards to create a robust dataset of 1,645 API calls. Each API calls has a json object with the following fields: domain, framework, functionality, api_name, api_call, api_arguments, environment_requirements, example_code, performance, and description.. APIBench also contains a dataset, which is enriched with synthetic instruction data generated by the GPT-4 model, creating 10 unique instruction-API pairs per API call.

4.2 Prompts

To ensure reproducibility, we present the prompts utilized by the agent during the experiment.

- The prompt used for query analysis and generation P_q: "Kindly analyze the user's request in detail and provide a precise description of the technology they require."

- The prompt used for analysis of candidate tools P_f: "Briefly describe the core functionality based on the tool documentation."

- The prompt used for Determining of the optimal tool P_f: "Identify the technology that best fulfills the user's requirement from the given options."

4.3 Baseline

Based on (Patil et al., 2023), we explore three distinct retrieval methods as baselines: BM25, text-embedding-ada-002[2] and Sentence-BERT (Reimers and Gurevych, 2019), which is trained on the training set of APIBench. For more details, please refer to the appendix.

4.4 Evaluation Method

Considering the limited context of language models and the extensive nature of API documentation.Both the baseline method and our approach, are designed to return the most pertinent API, with accuracy gauged against the provided labels. In our approach, every API undergoes a comprehensive core functionality analysis as depicted in Equation 5. This analysis is typically much more concise than the full API documentation. Hence, we can consider the top-k candidate tools before making a final decision. By default, we set k to 5. In subsequent phases, we'll delve deeper with experiment ablation and detailed analysis. For instance, we'll compare the final success rate about a baseline method, which returns the top-k for direct input into the primary model, measures up against an agent-based approach that pinpoints the top-1 option.

For the final success rate evaluation, we feed both the query and the relevant API document into a large language model, with the default being GPT-3.5. The objective is to ascertain the correctness of the method invoked by the language model. Given the plethora of models and challenges in direct deployment, we follow (Patil et al., 2023) and evaluate our model's output by examining its functional equivalence. The solution employs the Abstract Syntax Tree (AST) tree-matching technique. Since our study zeroes in on a singular API call, the procedure becomes relatively straightforward. By examining if the AST of the proposed API call exists as a sub-tree within the benchmark API call, we can effectively pinpoint the exact API being employed from our dataset.

All the experimental results in this paper are the statistical average of different runs. Unless otherwise specified, we default to GPT-3.5 as the agent's LLM Kernel.

[2]https://openai.com/blog/new-and-improved-embedding-model

4.5 Result

The Table 2 provides a comprehensive view of the retrieval accuracy rates of various methods across three different datasets: TorchHub, HuggingFace, and TensorFlow Hub. We can derive the following insights:

1. Challenges in Retrieval from Diverse APIs: Selecting the most relevant results from the set of APIs using a specific query is inherently difficult. This can be confirmed by looking at the performance scores of the two baseline methods across the datasets. For instance, BM25 achieved a success rate of 16.13% on TorchHub, 18.03% on HuggingFace, and 42.92% on TensorFlow Hub. Both methods, as shown in the table, found it challenging to consistently retrieve relevant results. This emphasizes the complexities of developing an effective retrieval system for a wide range of API functionalities.

2. Diverse Challenges Across Datasets: The nature and quality of API documentation can differ markedly across datasets. While some datasets offer clearly discernible documentation, others present more nuanced distinctions. The contrasting success rates, particularly the disparity where BM25 fetches 42.92% on TensorFlow Hub but only 16.13% on TorchHub, underscore the unique challenges each dataset introduces, emphasizing the nuanced task of retrieval.

3. Superiority of Agent-Aided Methods: All three methods showed enhanced performance when coupled with the Agent. This points towards the efficacy of the Agent in improving the accuracy of the retrieval process. The 'Agent with BM25' and 'Agent with Ada Embedding' methods seem to outperform their standalone counterparts by a significant margin, underscoring the importance of integrating agent-based methodologies in the retrieval process. Even for advanced retrieval models such as Sentence-BERT, our agent method has also resulted in a substantial enhancement in performance.

4. Agent Benefits Increase with Baseline's Poor Performance: When the base retrieval method falters, the agent's contribution is more pronounced, improving the overall retrieval rate.

Method	TorchHub		HuggingFace		TensorFlow Hub	
	Retrieval Accuracy	Final Success	Retrieval Accuracy	Final Success	Retrieval Accuracy	Final Success
BM25	16.13	13.57	18.03	16.47	42.92	37.10
Agent with BM25	39.65	35.78	42.19	39.28	65.04	58.77
Ada Embedding	38.71	33.92	52.16	48.33	70.01	64.60
Agent with Ada Embedding	57.38	51.20	69.54	62.11	75.53	69.91
Sentence-BERT	76.44	70.35	84.15	77.49	88.01	82.62
Agent with Sentence-BERT	85.39	78.17	91.23	83.12	95.28	90.07

Table 2: Comparison of retrieval accuracy rate and final success rate for different methods across three datasets.

This can be largely attributed to the agent's design, which maximally leverages the knowledge and reasoning capabilities of the large language model.

5. Retrieval Accuracy Directly Influences Final Success: A higher retrieval accuracy rate naturally bolsters the final success rate. If a retrieval system missteps, it restricts the subsequent process's efficacy. Conversely, with apt tools, the success likelihood surges. However, the model's inherent limitations, like producing incorrect or hallucinated information, can still slightly affect its success rate. This highlights the importance of accurate retrieval systems in the whole process

In summary, the result serves as a testament to the evolving challenges in the realm of API retrieval. While baseline methods may falter in consistently delivering accurate results, the integration of specialized agents, as seen in the case of BM25 and Ada Embedding, offers a promising avenue for enhancing retrieval accuracy rates.

LLM Kernel	HuggingFace
Alpaca	26.12
Vicuna	30.78
Llama2Chat	34.10
GPT-3.5	42.19
GPT-4	48.52

Table 3: Comparison of BM-25 retrieval accuracy rate across different LLM kernels.

4.6 Different LLM kernels

The table 3 showcases the retrieval accuracy rates of agents built upon various Language Learning Model (LLM) kernels when tested on the HuggingFace dataset. As the capacity of the underlying LLM kernel increase, there is a corresponding improvement in the success rate of agent-driven information retrieval.

Experiment	HuggingFace
Agent with BM25	42.19
w/o Tool Description Generation	36.22
w/o Analysis of Candidate Tool	39.41
w/o Determining of the Optimal Tool	26.07

Table 4: Ablation study showing the impact of the three core functionalities of the agent.

4.7 Ablation Study

In the ablation study presented in Table 4,we employ GPT-3.5 as the LLM Kernel. By removing each functionality one at a time, we measure its relative contribution to the agent's overall effectiveness. The data from the table clearly shows that every functionality is pivotal to the agent's success. Notably, when "Determining of the Optimal Tool" is omitted, the performance drops substantially to 26.07%. This highlights the pivotal importance of the language model's decision-making prowess in attaining the best results. Another contributing factor to this decline is that without "Determining the Optimal Tool", the system automatically selects the first tool from the prior step's candidate list as the correct choice. This action indirectly weakens the significance of the "Analysis of Candidate Tool" feature.

4.8 Top-k Performance

Table 5 demonstrates the performance of the retrieval accuracy rate and the final success rate of both BM25 and Agent with BM25 methods on the HuggingFace dataset under different top-k values. From the data, we can draw the following conclusions:

- Growth in k-values and Accuracy: As the value of k increases, the accuracy rate of our method continues to rise. This is evident when comparing the retrieval accuracy across different top-k values. For instance, the retrieval accuracy for the BM25 method increases from 18.03% (Top-1) to 51.99% (Top-10). Similarly, the Agent with BM25 method exhibits

Method	Top-1		Top-5		Top-10	
	Retrieval Accuracy	Final Success	Retrieval Accuracy	Final Success	Retrieval Accuracy	Final Success
BM25	18.03	16.47	37.94	30.92	51.99	40.26
Agent with BM25	26.07	23.11	42.19	39.28	54.37	49.70

Table 5: Comparison of retrieval accuracy rate and final success rate for different top-k.

a rise in accuracy from 26.07% (Top-1) to 54.37% (Top-10).

- Comparison with Direct Top-k: Although directly feeding the top-k outputs from the BM25 method into a large language model for usage and then calculating the final success rate does not align with our tool selection definition given in method 3 (since the large model would have to make an implicit choice during the execution phase), it serves as a valuable comparison. The results clearly demonstrate that our method outperforms the direct BM25 top-k, which mainly benefit from three functionalities explored by ablation study.

- This disparity between retrieval accuracy and final success rate is more pronounced in the top-k method compared to ours. This suggests that selecting the most optimal tool and then providing it to the large language model can help alleviate its burden. Furthermore, this highlights the appropriateness and validity of our definition of tool selection as the most pertinent.

4.9 Chain-of-Tool

Method	HuggingFace
Agent with BM25	39.28
Agent with BM25 + context	41.21
Agent with Ada Embedding	62.11
Agent with Ada Embedding + context	63.92

Table 6: The impact of context about tool selection on final success rate.

Much like a chain-of-thought, the way we select and use tools can be viewed as a logical progression. This observation prompts a deeper inquiry: Could the contextual information present during the tool selection enhance the success rate of its subsequent use? After the agent selects the appropriate tool, the LLM kernel maintains the previous context. We directly passes the query and the tool back to this language model. Table 6 indicates that this method has improved the final success rate on the Huggingface dataset.

5 Conclusion

In this paper, we present the self-assisted tool selection method and have developed an language agent based on the the integration of large language model and retrievers. For any given query, the agent systematically analyzes the query, creates a tailored tool description, employs a retriever to find the relevant tool, examines the functionalities of the top-k tools from their original documentation, and finally, pinpoints the most suitable tool based on various analyses. On publicly available datasets, our method significantly enhances the accuracy of tool retrieval, leading to an improved success rate in problem-solving. Since APIBench consists of queries that can be resolved with just a single tool, in the future, we plan to extend our approach to queries that require the combined use of multiple tools. Some potential strategies include decomposing complex task queries into multiple single queries using large language models, and then applying our current method for tool selection.

6 Limitations

The primary limitation of our method lies in the sequential nature of the Self-Assist approach, which is susceptible to cumulative errors. For instance, inaccuracies in the initial Query Analysis could propagate through to later retrieval stages. Empirically, we have found that with advanced language models, such hallucination errors are minimal and often inconsequential. However, for less sophisticated open-source LLMs, they have an observable impact in certain scenarios. Overall, our method still yields positive gains in practice. Another potential concern is the additional token usage. In the appendix, we compare the average token usage, and despite incorporating more steps, there isn't a significant increase in the number of tokens used.

References

Luiz Henrique Bonifacio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. 2022. Inpars: Unsupervised dataset generation for information retrieval. *Proceedings of the 45th International ACM SIGIR*

Conference on Research and Development in Information Retrieval.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. 2023. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pages 287–318. PMLR.

Zhuyun Dai, Vincent Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith B. Hall, and Ming-Wei Chang. 2022. Promptagator: Few-shot dense retrieval from 8 examples. *ArXiv*, abs/2209.11755.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. PAL: program-aided language models. *CoRR*, abs/2211.10435.

Zelalem Gero, Chandan Singh, Hao Cheng, Tristan Naumann, Michel Galley, Jianfeng Gao, and Hoifung Poon. 2023. Self-verification improves few-shot clinical information extraction. *arXiv preprint arXiv:2306.00024*.

Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. 2021. Efficiently teaching an effective dense retriever with balanced topic aware sampling. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 113–122.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781.

Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. 2022. Internet-augmented language models through few-shot prompting for open-domain question answering. *CoRR*, abs/2203.05115.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR.

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*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021a. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021b. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023a. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*.

Bhargavi Paranjape, Scott M. Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Túlio Ribeiro. 2023b. ART: automatic multi-step reasoning and tool-use for large language models. *CoRR*, abs/2303.09014.

Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*.

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, 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. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Conference on Empirical Methods in Natural Language Processing*.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523.

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

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

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Pengjie Ren, Dawei Yin, and Zhaochun Ren. 2023. Is chatgpt good at search? investigating large language models as re-ranking agent. *arXiv preprint arXiv:2304.09542*.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

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

Parameter Settings

Specifically, the Self-Assist framework does not introduce any additional hyperparameters. Similarly, BM25 inherently does not have extra parameters. For the text-embedding-ada-002, we set the maximum input length to 2048. The parameters in our paper primarily relate to the Large Language Models (LLMs), which are set as follows: frequency penalty at 0, logit bias not applicable (null), a maximum of 4096 tokens, a single response generation (n=1), presence penalty at 0, no specific stop sequence (null), streaming disabled (false), temperature at 1, and top_p also at 1.

Baseline

Based on (Patil et al., 2023), we explore three distinct retrieval methods as baselines:

BM25 BM25 is a popular ranking function used in information retrieval systems, particularly in search engines. The BM25 function then evaluates the relevance of a document relative to a query based on the frequency of the query terms in the document, taking into account factors such as the length of the document and the average document length in the collection. For BM25, we consider each API as a separate document. During retrieval, we use the user's query to search the index and fetch the relevant APIs.

Embedding-based We use the text-embedding-ada-002[3]for embedding-based retriever. First, we extract the embedding representations for both the query and the API documents. We then assess their relationship using cosine similarity. All API documents are ranked according to this similarity measure with the query. Depending on the need, we either return the single document with the utmost score or the top N highest-scoring documents.

Sentence-BERT We incorporated Sentence-BERT (Reimers and Gurevych, 2019) to train a dense retriever on the training set of APIBench. This API retriever generates embeddings for both the instruction and the API document and assesses their relevance through embedding similarity.

Additional Token Usage

Table 7 in our paper shows the average tokens used for retrieving the top-5 tools and executing. De-

[3]https://openai.com/blog/new-and-improved-embedding-model

Method	HuggingFace	TensorFlow Hub	TorchHub
Baseline	2039.4	1795.2	2663.5
Ours	2186.3	1976.4	2805.8

Table 7: Comparison of average token usage.

spite our method's complexity, token use remains low, primarily for API documentation. We encouraged concise commands like "Briefly describe" to conserve tokens, detailed in our Prompts Section. Plus, the cost of token usage is decreasing due to improved inference and technology.