AI AGENT 007: TOOLING UP SUCCESS

Team 83

Abstract

A Language model L has a set of tools T, and a user query Q is given. To answer query Q, we need to use existing tools. You need to output the subset of tools to be used to answer the query, the arguments that these tools should be called with, and how to compose the tools to answer the query.

1 Introduction

The advancement of Language Models (LMs) has paved the way for numerous breakthroughs in natural language understanding and generation tasks. However, one persistent challenge remains: enabling Large Language Models (LLMs) to acquire the nuanced ability of Tool Use, specifically the capability to return a subset of tools in an appropriate format for task-solving. Addressing this challenge necessitates a comprehensive approach that combines innovative techniques to generate, fine-tune, and augment LLMs.

The problem statement at hand prompts us to develop a mechanism that empowers LLMs to learn Tool Use effectively, generating a subset of tools with the requisite format for task resolution. To tackle this intricate task, we embarked on a two-fold strategy. Initially, we employed the potent capabilities of GPT-4 to generate a diverse set of tools and curated a dataset comprising query and output pairs using GPT-4. This dataset was then employed to fine-tune various models, notably GPT-3.5.Turbo 1106.

Our approach introduces a novel methodology that leverages a two-step prompting mechanism, akin to a 2 LLM (Language Model) approach, wherein a chain of thoughts is strategically employed. This two-step prompting methodology facilitates a more nuanced and contextually rich understanding by the LLM, paving the way for improved Tool Use learning. The first step involves the generation of a set of tools through GPT-4, and in the second step, the LLM refines its understanding based on the context provided by the generated tools.

A pivotal aspect of our methodology involves the integration of Retrieval Augmented Generation (RAG) with an LLM. This amalgamation allows the LLM to benefit from a hybrid approach, combining the strengths of both retrieval-based and generation-based methods. By tuning the LLM in conjunction with the RAG approach, we seek to enhance the model's ability to grasp the intricacies of Tool Use.

In the subsequent sections, we will delve into the details of our two-step prompting methodology, elucidating how it optimally guides LLMs towards a nuanced comprehension of Tool Use. Additionally, we will expound upon the integration of RAG with LLMs and the subsequent tuning strategies employed to refine the model's performance. Through these innovations, we aim to contribute to the ongoing discourse on advancing LLMs towards more sophisticated language understanding and problem-solving capabilities.

2 Data Generation

To generate the dataset we leveraged the power of chatgpt-3.5-turbo-1106. A prompt was initially written out for the generation of data. The prompt written had consideration for $$prev[i] logic and proper formatting of JSON schema. We used the 3.5 version as it was found during experimentation that GPT 4 gave almost the same results as 3.5 but the cost per token was relatively less for 3.5. We experimented through configurations of temperature and top-k and found that a temperature of 0.3 and top-k of 1 was ideal for dataset generation using our prompt. These values of temperature and top-k were ideal for the prompt we have formatted. The prompt we created uses a dynamic set of tools for each query-output pair generated. As the number of tools initially were few in number, we created tools by ourselves to support the feature of using a dynamic set of tools for every query-output pair. Then we used the tools provided by DevRev to create the test dataset and the tools handcrafted by our team to create the training and validation datasets. This is during the training phase, we can avoid any data leakage between test datasets with training or validation datasets. Then it was found that while generating query-output pairs, certain examples with natural language were present before the query-output pair. We then created a filter to remove the language of the generated data. The flow structure is shown in Fig.1

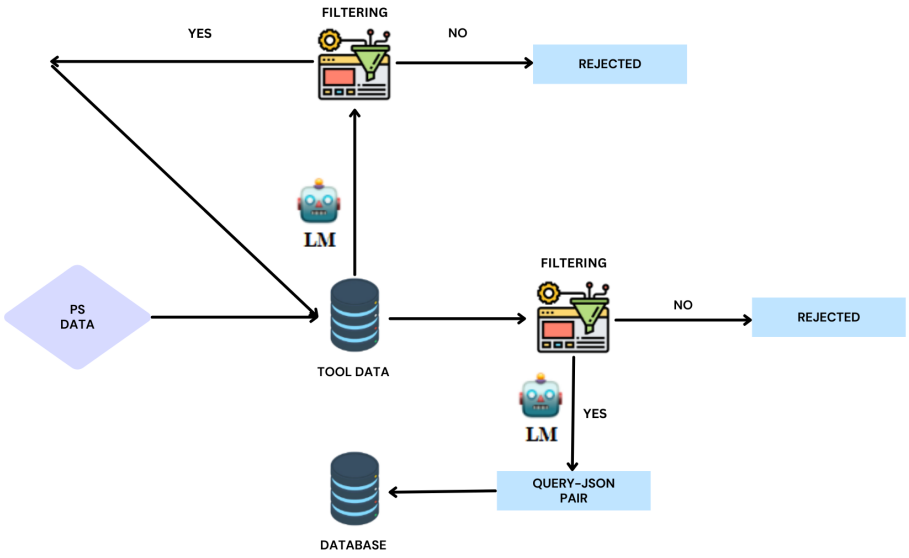

Figure 1: Data Generation Flow Structure

3 Direct Inferencing:

Our initial foray into direct inferencing involved utilizing LLama 2 7b and CodeLLama 7b on the test set without any fine-tuning, retrieval-augmented generation (RAG), or additional prompting. This approach entailed passing the tool list and querying the output, yielding suboptimal results. The discrepancies between the generated output and the expected results were pronounced, indicating the inadequacy of this method in addressing our specific problem. Transitioning to GPT-3.5 for inference showed improvements, yet it fell short in providing a comprehensive set of tools required for the query, rendering it suboptimal for our purposes.

4 Finetuning

4.1 GPT-3.5

For initial experiments, we decided to fine-tune gpt-3.5-turbo-1106 using our training and validation datasets. We set the number of epochs to be 3 for the training process. We then generated outputs for our test dataset which consisted of examples consisting of tools only from the DevRev dataset. It was found that the outputs almost matched perfectly with the output which is supposed to be predicted.

4.2 LLama and CodeLLama

Turning our attention to LLama-7b and CodeLLama-7b (utilizing 8-bit quantization) with PEFT(Parameter Efficient Fine tuning) as LORA, we embarked on fine-tuning endeavors. Notably, these models demonstrated exceptional performance on the validation set. However, challenges emerged when applying these fine-tuned models to the test set, featuring entirely novel tools. The models exhibited a tendency to hallucinate, generating inaccurate outputs in the presence of unseen tools. This phenomenon emphasized the need for additional measures to address the generalization challenges posed by unseen tools in the test set.

In summary, while direct inferencing revealed limitations, fine-tuning efforts exhibited promising outcomes on familiar data. However, the fine-tuned models faced challenges when confronted with unseen tools, necessitating further exploration of techniques to enhance the models' adaptability to novel inputs. Subsequent sections will delve into our innovative approaches, such as retrieval-augmented generation and two-step prompting, aimed at overcoming these challenges and refining the models for optimal performance.

5 2-Model Approach

Following the creation of the dataset, our objective is to enhance the Large Language Models (LLM) performance in generating an accurate set of tools based on the arguments present within it. Departing from the conventional practice of passing prompts directly to the LLM, as exemplified by GPT-3.5-turbo 1106 in our experimentation, we devised a two-step process for improved efficacy.

In the initial step, using GPT-4, we presented only the query, tool names, and descriptions along with a prompt. The task assigned to GPT-4 was to generate a chain of thoughts suitable for the subsequent LLM, facilitating the production of the correct JSON Schema with the corresponding output. In summary, this initial prompt sought to establish the contextual groundwork for the subsequent tool generation.

Subsequently, the second prompt incorporated the chain of thoughts generated by GPT-4 in the preceding step, along with the complete set of tools, their names, descriptions, arguments, and their descriptions. Additionally, essential constraints for the final output JSON Schema format were included in this comprehensive prompt. Remarkably, the results obtained on the Validation set were exceptional, as the second LLM (GPT-3.5-turbo 1106 in this instance) accurately predicted the complete format, arguments, and their respective descriptions without any fine-tuning requirements.

Directly passing queries with tool lists to the LLM, even employing GPT-3.5-turbo-1106, exhibited instances where only 2-3 tools were generated as output in JSON Schema. In contrast, employing

our two-step procedure demonstrated a marked improvement, with the LLM consistently generating 6-7 tools, all pertinent to the query without any instances of hallucinations. Thus, the 2-Step prompting approach revealed notable enhancements compared to direct prompting to the LLMs. This methodology not only proved effective in preventing under-generation but also ensured the generation of relevant tools, showcasing its superiority over direct prompting methods.

6 Our Custom Metric

In our pursuit of devising a metric aligned with human-like evaluation processes, we introduce the Jaccard Similarity metric. This metric operates in a step-by-step manner, emulating how humans might systematically assess the structural similarity between expected and generated JSON outputs.
Example
Correct JSON:
{
"tool_name": "works_list",
"arguments": [
{"argument_name": "applies_to_part", "argument_value": ["FEAT-123", "ENH-123"]}
]
}
Generated JSON:
{
"tool_name": "works_list",
"arguments": [
{"argument_name": "applies_to_part", "argument_value": ["FEAT-123", "ENH-456"]}
]
}
Step 1: JSON Schema Validation
Define a JSON schema to represent the expected structure. For simplicity, let's consider a schema where "tool_name" is a required string, and "arguments" is an array of objects with "argument_name" and "argument_value."
Step 2: Token-Level Comparison
Tokenize both JSONs into individual elements (keys, values, brackets) and compare corresponding tokens:
Correct JSON Tokens: ["{", "tool_name", ":", "works_list", ...]
Generated JSON Tokens: ["{", "tool_name", ":", "works_list", ...]
Step 3: Jaccard Similarity for Sets
Treat sets of keys as mathematical sets:
Correct JSON Keys Set: {"tool_name", "arguments", "argument_name", "argument_value"}
Generated JSON Keys Set: "tool_name", "arguments", "argument_name", "argument_value"
Jaccard Similarity:
J =| {"tool_name", "arguments", "argument_name",
"argument_value"}∩{"tool_name", "arguments", "argument_name", "argument_value"} |
÷ | {"tool_name", "arguments", "argument_name", "argument_value"} ∪ {"tool_name", "arguments", "argument_name", "argument_value"} |

Step 4 Structural Loss Function
Define a loss function that penalizes structural deviations. For example, penalize differences in "argument_value":
loss = weight_missing_keys * calculate_missing_keys() + weight_extra_keys * calculate_extra_keys() + weight_argument_value_difference * calculate_argument_value_difference()

Step 5: Hierarchical Comparison Recursively compare substructures. In this case, compare the "arguments" array:
Correct JSON Arguments: [{"argument_name": "applies_to_part", "argument_value": ["FEAT-123", "ENH-123"]}]
Generated JSON Arguments: [{"argument_name": "applies_to_part", "argument_value": ["FEAT-123", "ENH-456"]}]

7 Retrieval-Augmented Generation (RAG) Strategies for Tool Retrieval:

Retrieval-augmented generation (RAG) constitutes a pivotal aspect of our methodology, serving as an indispensable framework to enhance the quality of Large Language Model (LLM)-generated responses. This framework supplements the internal representation of information within the LLM by grounding it on external sources of knowledge. Our exploration of RAG involved various retrieval techniques tailored to the context of tool generation.

7.1 ChromaDB Integration

For efficient vector storage and retrieval, we utilized ChromaDB as the underlying vector database. This choice contributed to the robustness of our retrieval mechanism, ensuring high-performance tool retrieval for improved LLM-generated responses.
ChromaDB is an open-source vector database, which has low latency. So we used this vectorDB to minimize the overall latency of the application

7.2 Initial Retrieval Attempts

Our first endeavor involved the implementation of a retriever designed to retrieve documents based on Tool Name, Description, Argument Name, and Description. However, this approach exhibited suboptimal performance, particularly when matching tool and argument names resulted in unintended retrievals. This limitation underscored the need for a more nuanced retrieval strategy.

7.3 Document and Metadata Separation

Another strategy involved the segregation of document name and metadata, with document names containing tool and argument descriptions, while metadata contained tool and argument names. This separation aimed to prioritize retrieval based on description rather than name, yielding notable improvements. However, occasional retrieval failures persisted, prompting further investigation.

7.4 Integration of Dense Passage Retriever

To address the shortcomings identified in our initial approach, we employed the Dense Passage Retriever, training the embedding model using the Haystack library. This adjustment aimed to enhance the contextual understanding of the queries and improve the relevance of retrieved tools. Despite advancements, challenges persisted, motivating further refinements.

7.5 Chain of Thought for Enhanced Retrieval

Recognizing the limitations of previous approaches, we introduced a novel method employing a chain of thought (COT). By transforming the prompt into sub-queries using an LLM, we facilitated more refined retrieval. Subsequently, tools were retrieved for each sub-query, duplicates were removed, and the results were concatenated. This innovative approach addressed challenges associated with self-referencing tools, such as the "whoami" tool, which were previously elusive.

```
Query:
{user_query}

Here's a Query. Transform the given query into distinct sentences, following the provided examples.ONLY OUTPUT AS JSOI

Example 1.:
Query: Prioritize my P0 issues and add them to the current sprint.

Output:
{{
"sentence1": "Address my specific issue.",
"sentence2": "Emphasize the prioritization of P0 issues.",
"sentence3": "Add listed issues to the current sprint."
}}

Example 2.:
Query: Summarize high severity tickets from the customer UltimateCustomer

Output:
{{
    "sentence1" : "search for customer",
    "sentence2" : "List high severity tickets Coming from 'UltimateCustomer'",
    "sentence2" : "Summarise them"
}}
```

Figure 2: Visualization of the COT-driven retrieval process

8 Final Approach:

The final pipeline we prepared first rewrites the query using chatGPT in a way that is optimal for the retrieval augmented generation and then we get a list of useful tools according to the query. This list along with the query is then passed to a fine-tuned chatgpt3.5 which outputs the JSON schema. Apart from this approach, we tried an approach in which we bifurcated the task in two parts:- a)ordering of tools in which they will be used and b)generation of proper tool-argument JSON schema.Both these tasks were done by prompting LLMs. This method gave better results than simply prompting an LLM would give. This approach was inspired by Chain Of Thought and task splitting as referenced in RestGPT.

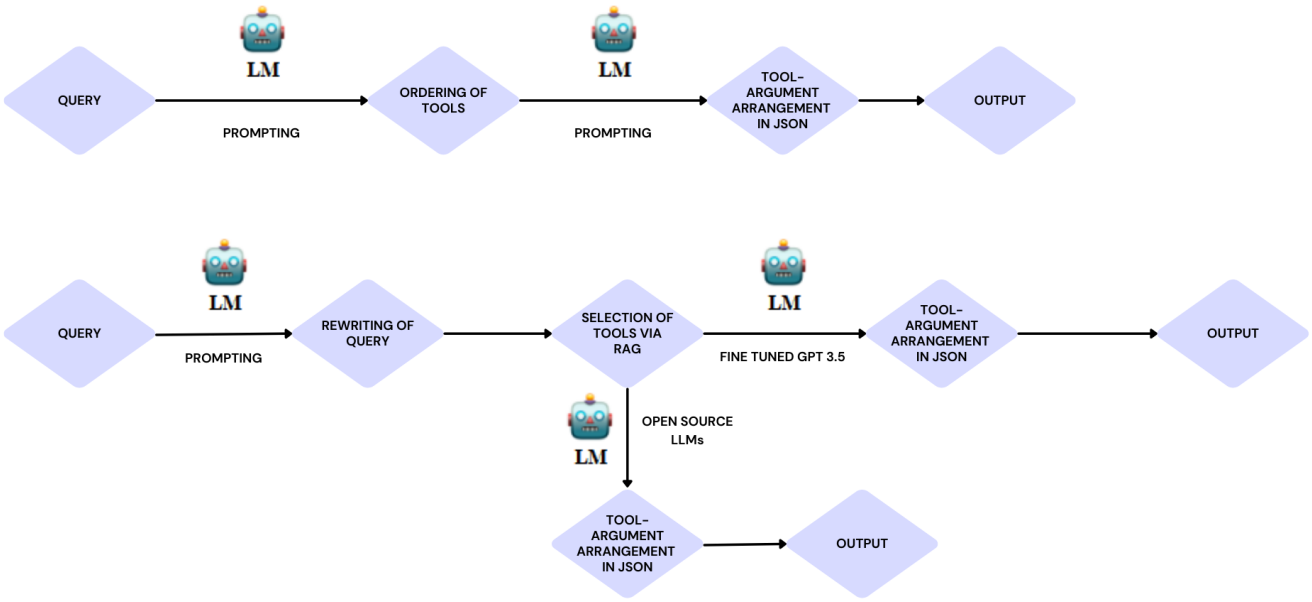

Figure 3: Final Pipeline

9 Future Scope :

9.1 Addressing Hallucination Challenges:

To mitigate the hallucination phenomenon in future models, there is a need for a more comprehensive understanding of the underlying causes. Exploring advanced techniques for data augmentation, diverse training scenarios, and tailored prompt engineering will be crucial. Striking a balance between the model's generative capacity and its capacity to adhere to the specifics of the task requires further investigation.

9.2 Expanding Data Resources:

The role of data in model performance is evident, and future works should explore avenues for augmenting existing datasets. Efforts to diversify the dataset with a broader spectrum of tools and scenarios could contribute to enhanced model robustness and a reduction in hallucination tendencies. Additionally, investigating the potential of transfer learning from related domains may prove beneficial.

9.3 Contextualized Prompting Strategies:

Refining the prompting mechanism is a key component in tackling hallucination. Future research could delve into developing advanced prompting strategies that provide the necessary context and constraints, steering models towards more accurate tool generation. The optimization of prompting for each specific Llama model, considering its unique characteristics, is a promising avenue.

9.4 Evaluating Model Generalization:

Understanding how well models generalize to diverse sets of tools is vital. Future research should explore the impact of introducing novel tools during training and fine-tuning stages. Strategies to encourage adaptability and prevent hallucination when encountering previously unseen tools could lead to more robust and versatile Llama models.

References

[1] Code Llama: Open Foundation Models For Code Baptiste Rozière , Jonas Gehring , Fabian Gloeckle , Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, 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, Gabriel Synnaeve

[2] Llama 2: Open Foundation And Fine-Tuned Chat Models : Hugo Touvron Louis Martin Kevin Stone† Peter Albert Amjad Almahairi Yasmine Babaei Nikolay Bashlykov Soumya Batra Prajjwal Bhargava Shruti Bhosale Dan Bikel Lukas Blecher Cristian Canton Ferrer Moya Chen Guillem Cucurull David Esiobu Jude Fernandes Jeremy Fu Wenyin Fu Brian Fuller Cynthia Gao Vedanuj Goswami Naman Goyal Anthony Hartshorn Saghar Hosseini Rui Hou Hakan Inan Marcin Kardas Viktor Kerkez Madian Khabsa Isabel Kloumann Artem Korenev Punit Singh Koura Marie-Anne Lachaux Thibaut Lavril Jenya Lee Diana Liskovich Yinghai Lu Yuning Mao Xavier Martinet Todor Mihaylov Pushkar Mishra Igor Molybog Yixin Nie Andrew Poulton Jeremy Reizenstein Rashi Rungta Kalyan Saladi Alan Schelten Ruan Silva Eric Michael Smith Ranjan Subramanian Xiaoqing Ellen Tan Binh Tang Ross Taylor Adina Williams Jian Xiang Kuan Puxin Xu Zheng Yan Iliyan Zarov Yuchen Zhang Angela Fan Melanie Kambadur Sharan Narang Aurelien Rodriguez Robert Stojnic Sergey Edunov Thomas Scialom

[3] Gpt-4 Technical Report - Openai

[4] Retrieval-Augmented Generation For Knowledge-Intensive Nlp Tasks Patrick Lewis, Ethan Perez , Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin† , Naman Goyal , Heinrich Küttler† , Mike Lewis† , Wen-Tau Yih , Tim Rocktäschel, Sebastian Riedel, Douwe Kiela

[5] Chain-Of-Thought Prompting Elicits Reasoning In Large Language Models Jason Wei Xuezhi Wang Dale Schuurmans Maarten Bosma Brian Ichter Fei Xia Ed H. Chi Quoc V. Le Denny Zhou

[6] RestGPT: Connecting Large Language Models with Real-World RESTful APIs Yifan Song , Weimin Xiong , Dawei Zhu , Wenhao Wu , Han Qian , Mingbo Song Hailiang Huang , Cheng Li , Ke Wang , Rong Yao , Ye Tian , Sujian Li1

[7] LoRA: Low-Rank Adaptation of Large Language Models Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen

[8] QLoRA: Efficient Finetuning of Quantized LLMs Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, Luke Zettlemoyer

[9] Dense Passage Retrieval for Open-Domain Question Answering Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, Wen-tau Yih

[10] Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods G.V.Cormack,C.L.A.Clarke,Stefan B¨uttcher