**Mid-Term Report**

**Introduction**

In this report, we present our solution for resolving the problem of domain-specific querying with the help of provided tools. The task included resolving user queries with the help of a model to output a JSON formatted text containing all the APIs called in sequence along with their correct set of arguments. Our study of several existing tools and techniques available for similar problems and careful evaluation of some of them for their accuracy and performance has led us towards the final approach. Our custom dataset generation pipeline for examples on input/output and BERT-based tool filtering approach has also helped to improve the efficiency of our overall approach. In all, this report delves into the challenges of orchestrating these elements effectively to improve overall accuracy and navigate the intricacies of domain-specific queries.

**Literature Review**

Querying techniques like ReAct [1] and ART [2] build on CoT querying to seamlessly generate both reasoning traces and task-specific actions in an interleaved fashion. The integration of reasoning traces assists the model in the induction, tracking, and dynamic updating of action plans, including the handling of exceptions. Concurrently, the generated actions enable the model to interact effectively with external sources, such as knowledge bases or environments, facilitating the acquisition of additional information.

Self-Instruct [3] is a framework for improving the instruction-following capabilities of pre-trained language models by bootstrapping off training data generated from the LLMs themselves. It features a 3-step cycle starting with a small set of seed tasks as the task pool. Random tasks are sampled from the task pool and used to prompt an LLM to generate instructions and tasks, followed by filtering low-quality or similar generations. The filtered outputs are then added back to the initial repository of tasks, which can finally be used for the instruction tuning of the LLM. ToolQA [4] uses a similar approach utilising Human-Guided Question generation and Programmatic Answer Generation to generate accurate answers to the generated questions.

Gorilla [5] uses the paradigm of self-instruct fine-tuning and retrieval to enable LLMs to accurately select from a large, overlapping, and changing set of tools expressed using their APIs and API documentation. They fine-tuned the LLaMA-7B model on the generated instruction to get Gorilla.

ToolLLM [6] leverages the conversational capabilities of ChatGPT to develop a novel depth-first search-based decision tree algorithm to evaluate multiple decision paths and to either retract steps or proceed along a promising path. Multi-step solutions are cast as a multi-round conversation with ChatGPT in a manner similar to ReAct but improved for better error propagation and solution exploration.

**Techniques Evaluated**

We tested multiple approaches to solve the problem, evaluating each on the metrics of token length, solution accuracy and response time. To ensure the correctness of the solution provided, we made a robust similarity function that factors in the possible differences in the orders of the generated tools used.

**Langchain and Naiive ReAct Querying**

We have integrated Langchain, a robust AI application development framework, to facilitate the use of the ReAct querying paradigm. We extended LangChains base output parsers to ensure the generated responses conform to the requested JSON output format. We further use the built-in tool handling functionality of LangChain to dynamically generate Tool classes, which allows GPT-3.5 to utilise the ReAct paradigm, allowing it to perform both reasoning traces and task-specific actions in an interleaved manner.

In our initial approach, we queried the GPT 3.5 API using the ReAct querying paradigm. The off-the-shelf GPT-3.5 model was used in this case without any fine-tuning. We construct a prompt that sends all the available tools and some example instances followed by the task we wish to complete to test the one-shot response generation. The prompt also ensures that the LM returns an empty sequence in case there are no available tools for the given task.

This method provides promising results with our prompt. However, it requires a large amount of tokens to generate a response.

**BERT-Based tool filtering**

In the ReAct approach, it has been observed that combining ReAct with a few-shot examples can result in lengthy instructions that sometimes may exceed context length restrictions. To address this issue, in ReAct, we incorporated BERT to send only the relevant tools in the given context.

The process involves using BERT to calculate the cosine-similarity between BERT embeddings from the tool descriptions and embeddings of the current user prompt. We then selectively choose to include a tool in the prompt only if the calculated similarity surpasses a certain threshold. This approach effectively reduces the number of tokens and, consequently, the Language Model (LLM) cost.

**Fine-Tuning GPT-3.5 and**

**Training Dataset Generation**

We generated a dataset containing artificially generated tasks and the correct ordering of tools that need to be used to complete the task. To accomplish this, we opted for an approach similar to self-instruct [3]. We selected a random subset of tools from the complete set and queried GPT-3.5 to generate a set of tasks and their corresponding solution paths that necessarily used these tools. These results were then added to the original set, and the process was repeated. This allowed us to have good coverage over all the tools available in the provided examples.

We used this process to generate a training set of 50 tasks and their correct solution using the example queries provided in the dataset to bootstrap the process. For the process of incorporating new tools, we devised a systematic strategy. For each added tool, we curated subsets that inherently involved the use of the new tool within the given tasks. Subsequently, we queried the fine-tuned GPT-3.5 model to generate corresponding tasks and their respective tool sets, effectively tailoring the model to the unique contributions of the introduced tools.

We fine-tuned the GPT-3.5 Turbo model on the generated dataset and used the resulting model for inference. The purpose is to reduce the number of tokens used per query and, consequently, the cost. We also made the model conversational so that it may ask the user for additional input whenever it comes across some data that is unavailable through the given set of tools.

While these approaches gave us promising results, it was observed that certain queries, especially those involving intricate multi-step tasks, may yield suboptimal results. And it is required to decompose complex queries into simpler subqueries, allowing for more accurate identification of relevant tools.

**Final Approach**

We decided to use our fine tuned GPT-3.5 model with our BERT Based tool filtering approach to perform inference. In addition we decided to incorporate query decomposition to ensure robustness even in mutistep queries. To do this we aim to do the following:

1. Multi-stage Query Decomposition:

This involves breaking down complex queries into distinct subqueries, each targeting specific components of the overall task. This multi-stage decomposition facilitates a more nuanced understanding of user intent. For instance, when confronted with a query like "Prioritize my P0 issues and add them to the current sprint," our model breaks it into two subqueries: "Prioritize my P0 issues" and "Add them to the current sprint."

2. Tool Identification and Aggregation:

Upon decomposing the query, the model processes each subquery independently, identifying relevant tools for the specific task encapsulated in that subquery. This enables a more granular understanding of user requirements. In the example provided, the first subquery may prompt the inclusion of the "whoami" tool, which might be overlooked in the holistic analysis of the original query.

This approach is expected to produce better results and our ongoing efforts focus on expanding and refining this methodology.

**Future Work**

Several efficient models, each with unique strengths can be further evaluated. LLaMA 2 being much faster and efficient than other LLMs, requires far less computing power and resources to test new approaches. This efficiency positions LLaMA 2 as a valuable asset for quick experimentation and development. Another model Claude AI excels in resource utilization and provides an increased token limit.

**TODO: add tables, graphs related to metrics; add numbers**

For future attempts, we may consider using non-LLM-based approaches to analyse the task and gauge the dependencies between the entities to generate a solution path. Coreference analysis [7] and Named Entity Recognition can be used here to preprocess the data and perhaps even generate an algorithmic solution.

**References:**

[1] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models.

[2] Paranjape, B., Lundberg, S., Singh, S., Hajishirzi, H., Zettlemoyer, L., & Ribeiro, M. T. (2023). ART: Automatic multi-step reasoning and tool-use for large language models.

[3] Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Khashabi, D., & Hajishirzi, H. (2022). Self-Instruct: Aligning Language Model with Self Generated Instructions. In arXiv preprint arXiv:2212.10560.

[4] Zhuang, Y., Yu, Y., Wang, K., Sun, H., & Zhang, C. (2023). ToolQA: A Dataset for LLM Question Answering with External Tools.

[5] Patil, S. G., Zhang, T., Wang, X., & Gonzalez, J. E. (2023). Gorilla: Large Language Model Connected with Massive APIs. ArXiv Preprint ArXiv:2305.15334.

[6] Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., & Sun, M. (2023). ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs.

[7] Sukthanker, R., Poria, S., Cambria, E., & Thirunavukarasu, R. (2020). Anaphora and coreference resolution: A review. Information Fusion, 59, 139–162. https://doi.org/https://doi.org/10.1016/j.inffus.2020.01.010