DevRev's AI Agent 007: Tooling up for Success

Team 15

Abstract

We aim to develop a Large Language Model (LLM) powered chatbot, augmented by a set of tools, each accompanied by its detailed description. The chatbot should intelligently recommend a subset of these tools, specifying the arguments for their utilization, and providing guidance on how to combine them effectively to address user queries. Additionally, our solution should incorporate features facilitating the seamless addition and modification of tools within our toolset.

In the following sections, we present the current direction of our work which is backed by an extensive literature review and experimentation.

1. Literature Review

The challenge associated with this problem statement necessitates the strategic specialization of Large Language Models (LLMs) within specific domains. This targeted approach ensures optimal utilization of tools, enabling comprehensive responses to a diverse range of queries. This has been addressed in the literature under terms such as Tool Ordering, Task/Tool Planning, Task/Tool Scheduling, Tool Learning, Tool Augmented Large Language Models, Tool Integration with LLMs, Autonomous Agents backed by LLMs, and other similar terms. We also explore how we can incorporate the work done in the field of Domain Specific Question Answering using LLMs.

In the following subsections, we aim to narrow down our focus to a specific section of this extensive literature. Here, we elaborate on the part of the domain that aligns with our interests and pertains to the particular task at hand.

1.1. API Interaction

Majority of the existing literature in this field revolves around actually utilizing the APIs or external tools Qin et al. (2023), Patil et al. (2023), Schick et al. (2023), Yang et al. (2023) and integrating the results from these interactions as observations into the context.

However, our unique problem statement necessitates working solely with API descriptions, excluding the actual execution of API calls. Notable works that share a similar approach include the Reverse Chain method proposed by Zhang et al. (2023b) and the task planning step in HuggingGPT introduced by Shen et al. (2023).

1.2. Singular vs Multiple Tool Integration

A lot of the early literature in the tool learning domain was based on mostly single tool usage like Toolformer by (Schick et al., 2023) and Gorilla by (Patil et al., 2023). However, our task involves queries that might require multiple API calls. The solution may require multiple iterations with a tool, and there can be multiple tools involved, each with its own set of parameters.

1.3. Open Source vs Closed Source LLMs

The research community suggests that there is a significant gap between open-source LLMs and closed-source LLMs when it comes to tool-use/ tool planning capabilities Qin et al. (2023), Xu et al. (2023b).

While closed-source LLMs may initially perform adequately "off the shelf" for this task, they lack the inherent advantages offered by open-source alternatives, such as cost-effectiveness, flexibility, security, and reduced dependency on external providers and therefore, it is crucial to engage in extensive experimentation with open-source LLMs as well.

We are looking into several approaches that can help us align open-source LLMs to this task, possibly surpassing the capabilities of closed-source LLMs while giving us full control over the model. We are also looking into ways that would help make model serving and inference fast and cheap to justify the viability of deploying a model of our own (Discussed in Latency Section 5).

1.4. Number of Trainable Parameters

Based on the count of trainable parameters, our methodologies can be broadly categorized into three groups: Hard Prompting, which involves no trainable parameters; Parameter Efficient Fine Tuning (PEFT), where only a subset of parameters is tuned compared to the entire set; and Full Fine Tuning. Presently, our primary focus has been on maximizing efficiency through Hard Prompting, the most resource-friendly category among the three.

Concurrently, recognizing the constraints on resources, we

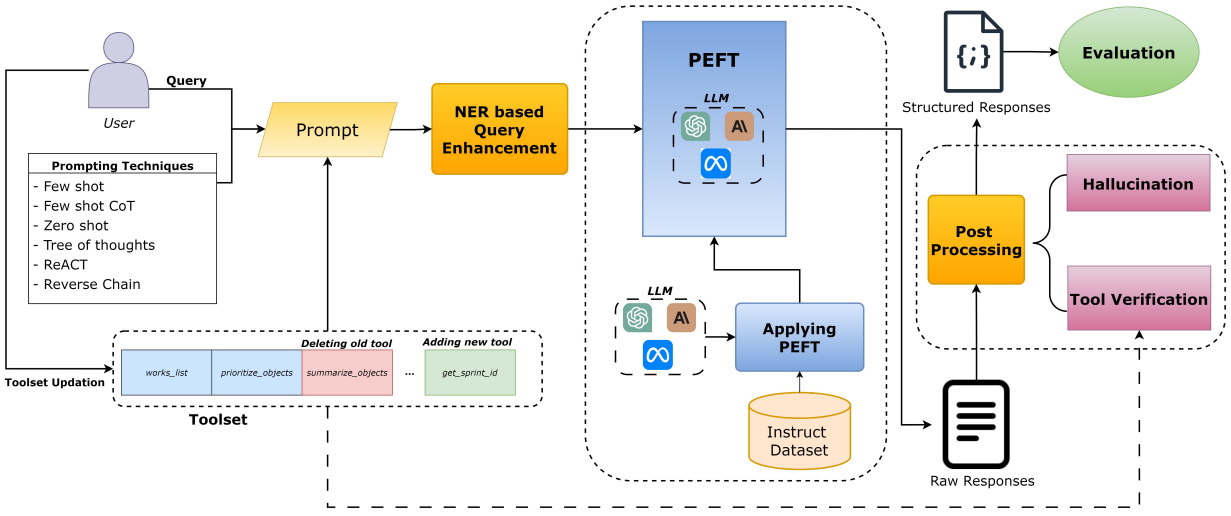

Figure 1. An overview of the inference pipeline augmented with synthetic data generation based on a human crafted seed set, instruct fine tuning using a scaled-up version of the seed set, post-processing of the raw outputs for tool verification and to check for hallucinations and an interface for tool updation.

are actively engaged in two key initiatives. Firstly, we are dedicated to constructing an instruct dataset Wang et al. (2022), Zhuang et al. (2023) Qin et al. (2023) (details to be discussed later). Secondly, we are exploring various PEFT techniques Li and Liang (2021), Jain et al. (2023), Dettmers et al. (2023), Hao et al. (2023), Zhang et al. (2023a) to enhance the alignment of our model, particularly smaller open source Language Model Models (LLMs), with the designated task. The optimization of smaller LLMs for this task not only contributes to resource conservation during deployment but also facilitates quicker inference.

1.5. Domain Specific Question Answering

In cases where the toolset is extensive and the documentation exceeds the capacity of in-context learning methods, researchers have experimented with diverse retrieval techniques Qin et al. (2023), Yuan et al. (2023), Patil et al. (2023). These methods involve retrieving pertinent tools for user queries from a tool database and integrating them into the chatbot's context using Retrieval Augmented Generation (RAG) pipelines to enhance their chatbots. However, following our discussion with the company, it appears that, at present, there is no immediate necessity for the incorporation of retrieval techniques. Most of the needed domain knowledge can be added in context.

2. Experimentation

In this section, we discuss details of our current experimentation and the corresponding evaluations. We have highlighted some findings in the report, with additional detailed results available in the accompanying deliverables.

2.1. Evaluation Metrics

In our current evaluation scheme, we focus on aspects such as API Selection, Argument Completion, API Ordering, Hallucinations and Structure of the Output to assess the quality of the output and token usage and chain latency to assess efficiency.

API selection aims to assess the ability of the model to correctly identify the necessary APIs required to solve the user query.

Argument Completion refers to the ability of the model to provide the correct argument names and values for each API being called.

API Ordering is the ability of the model to arrange the selected APIs in the correct order.

We also try to test for aspects like the model's tendency to Hallucinate, how the output is being structured, etc.

2.2. Hard Prompting Techniques

Hard Prompting, also referred to as Prompt Engineering, involves manually handcrafting text prompts with discrete input tokens. Developing effective prompts for any task requires extensive literature review and an iterative prompt development process following the guidelines of good prompt engineering.

The key advantages of Hard Prompting are:

- No training is involved.
- Allows for easy addition and modification of tools in the toolset (either directly in context via the system

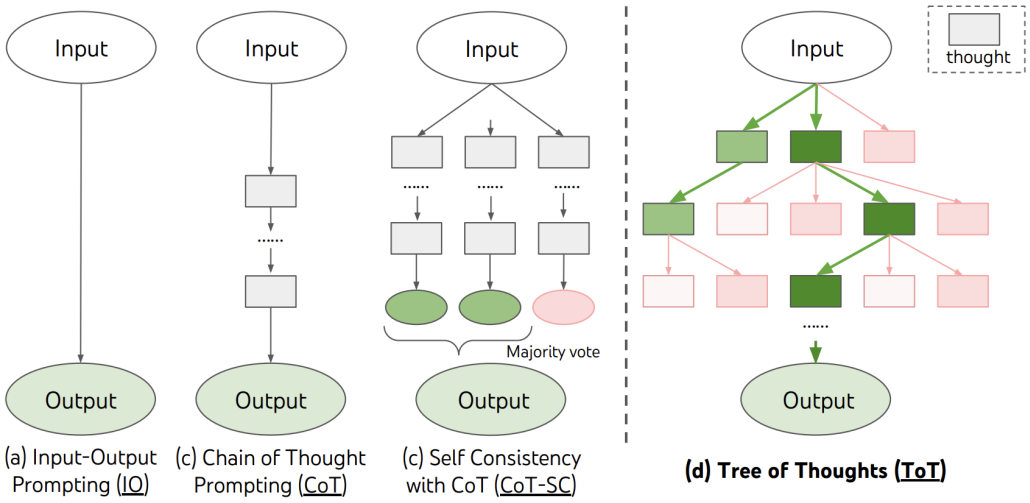

Figure 2. Overview of some prompting techniques Yao et al. (2023a). Here we have compared the structure of Tree of Thoughts prompting with Zero-Shot and Chain of Thought prompting

prompt or in the database in case of a large toolset).

Some disadvantages of Hard Prompting are:

- Limitation of context length (can be mitigated by using retrieval techniques)
- Mastering such a complex task which involves learning new and more complex tools is challenging.
- Model's existing vocabulary and knowledge are not aligned with the task at hand.

We have tried the following prompting techniques to tackle this task:

2.2.1. Zero Shot

Large LLMs today are tuned to follow instructions and are trained on large amounts of data; so they are capable of performing some tasks "zero-shot." As expected, for a complex task such as this, there are definite limitations in Zero Shot Prompting.

We noticed that there was a lot of variability in responses, hallucinations, and also issues with output structuring. The outputs were also very vulnerable to small changes in the prompts.

We tried multiple variations of zero-shot approaches which also served as baselines for our other approaches such as Single Prompt, Two Prompt (System prompt + JSON structuring prompt) with different variations of the structuring prompt, Single Prompt + Few Shot structuring prompt, Subtask decomposition + Subtask Answering (+ Structuring), Subtask decomposition + Subtask Answering + Follow up Structure Prompt, etc.

Instruction tuning has been shown to improve zero-shot learning Wei et al. (2022) and will be discussed in further sections.

2.2.2. Few Shot

Since zero-shot results showed that GPT-3.5 could not internalize the usage of APIs with no arguments and also confused words like 'summarize' within queries with natural language text, providing it with in-context demonstrations covering as many tools as possible was necessary.

Thus, we tried a few-shot prompting approach, which trains the model to execute tasks with minimal instances and is, therefore, a more flexible and adaptive approach than zero-shot prompting. The evaluation of the performance of the few-shot learning-based approach is based on limited exposure, with the assumption that the model's capacity to generalize from a few examples reflects its overall capability.

In our few-shot approach, we tested various prompts for output API list generation and structuring. However, we encountered issues with the model using natural text instead of indexing outputs from previous API tools. To address this, we had to explicitly instruct the model in the output structuring prompt. The model also tended to miss certain methods, such as creating a summary before generating actionable objects. In longer queries, it often fails to complete all tasks, providing an incomplete list of APIs.

2.2.3. Chain of Thought (CoT)

Chain of Thought prompting enables complex reasoning capabilities through intermediate reasoning steps. It allows

language models to break down complex problems into manageable intermediate steps, allocating additional computation where more reasoning is required.

The transparency of CoT provides insights into the model's decision-making process, aiding in debugging and understanding how specific answers are reached. This approach is versatile and potentially applicable to various tasks requiring complex reasoning, including this one.

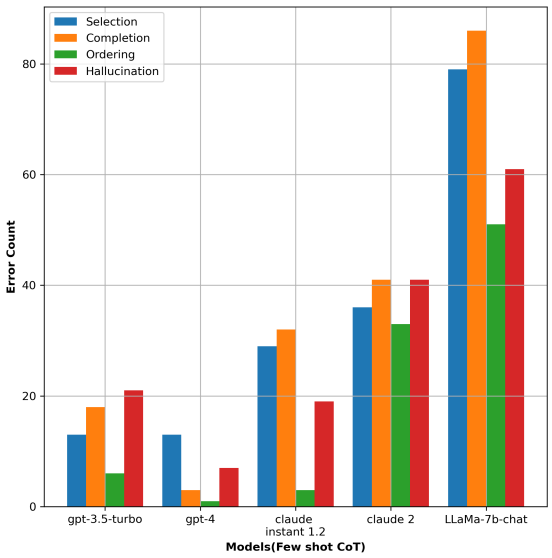

Figure 3. Few Shot CoT Error Statistics. GPT-4 performs the best while Llama has the worst performance.

Experimental Setup: Two distinct CoT prompting techniques were explored: Zero-Shot CoT and Few-Shot CoT.

1. Zero-Shot CoT Prompting:
    - Leveraging the approach outlined by Kojima et al. (2022), the "Let's think step by step" prompt was appended to the original query.
    - Trials were conducted on a self-curated dataset involving prompts that included lists of APIs with descriptions, explanations of each API, and the desired solution structure.
    - While results were generally optimal, a notable failure occurred in a query involving the "whoami" tool name, indicating a need for additional examples to aid the model's reasoning.
2. Few-Shot CoT Prompting:
    - For Few-Shot CoT trials, examples with reasoning were manually handcrafted. The prompt was similar to the one in zero-shot CoT, except at the end, two examples were given. From our experiments, Two-shot CoT seemed to perform well while One-Shot CoT struggled.
    - We ensured that the few shot examples were in a specific format so as to keep some uniformity and make it easier to add more examples or edit existing ones.

Exploration into advanced CoT techniques revealed a promising contender: Plan and Solve Prompting. This approach introduces a dual-phase methodology, requiring the formulation of a strategic plan to break down the overarching task into smaller, actionable subtasks, followed by the systematic execution of these subtasks in alignment with the devised plan. Notably, experimental results showcased its superiority over zero-shot CoT and demonstrated comparable performance to Few-Shot CoT. Given these encouraging findings, there is a strong motivation to conduct more in-depth experiments to unlock the full capabilities of this approach.

Model Name	API Selection	Argument Completion	API Ordering	Hallucination
gpt-3.5-turbo	13/100	18/100	6/100	21/100
gpt-4	13/100	3/100	1/100	7/100
Claude instant 1.2	29/100	32/100	3/100	19/100
Claude 2	36/100	41/100	33/100	41/100
LLaMa-7b-chat	79/100	86/100	51/100	61/100

Table 1. Error Statistics for Few Shot CoT

2.2.4. Tree of Thoughts

The adoption of the Tree of Thoughts technique is motivated by the limitations of linear chaining. ToT enables collaborative decision-making and iterative refinement, allowing agents to consider diverse reasoning paths and backtrack when necessary. This approach is crucial for handling complex queries that may demand multiple API calls, intricate logic, and conditional reasoning.

Model Name	API Selection	Argument Completion	API Ordering	Hallucination
gpt-3.5-turbo	29/100	19/100	7/100	16/100
gpt-4	14/100	6/100	3/100	5/100
Claude instant 1.2	36/100	39/100	8/100	23/100
Claude 2	17/100	34/100	7/100	9/100
LLaMa-7b-chat	76/100	89/100	56/100	79/100

Table 2. Error Statistics for ToT

The experiment utilizes a carefully crafted prompt that simulates a collaborative effort among three customer service agents. Equipped with exceptional logical thinking skills, these agents respond to a customer query using the Tree of

Thoughts method, detailing the utilization of a set of listed APIs at each step.

Performance Analysis : The Tree of Thoughts methodology inherently integrates an error-aware approach. At any stage, an agent may conclude that the given query cannot be resolved using the provided APIs. This allows for self-evaluation, acknowledgment of errors, and a thoughtful decision-making process. However, this approach was prone to overthinking as the model sometimes gave additional tools alongside those necessary to answer the query.

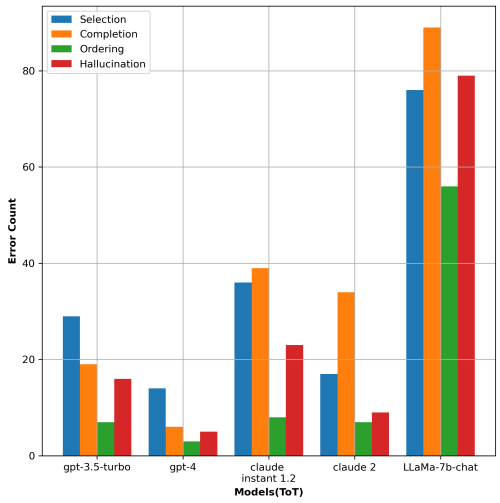

Figure 4. ToT Error Statistics

2.2.5. REACT

ReAct Yao et al. (2023b) overcomes prevalent issues of hallucination and error propagation in chain-of-thought reasoning by interacting with simple APIs and generating human-like ask-solving trajectories that are more interpretable than baselines without reasoning traces. ReAct prompts LLMs to generate both verbal reasoning traces and actions pertaining to a task in an interleaved manner, which allows the model to perform reasoning to create, maintain, and adjust plans for acting (reason to act), while also interacting with the external environments(tools) to incorporate additional information into reasoning (act to reason).

Experimental Setup:

We instructed the model to approach the query in an iterative manner while interleaving thought-action-observation steps in each iteration.

So in each iteration, the model performed the following steps:

1. **Thought:**
   - What is the task according to previous observations?
   - What you want to accomplish at this step.
   - What information is needed?
2. **Observation:**
   - What new information did you gain after this step?
3. **Action:**
   - **Pick:** Pick from the APIs that will work for you.
   - **Call:** Identify the correct arguments for the selected API.
   - **Finish:** Finish if you have all the APIs needed, and the query can be answered.

We manually composed ReAct-format trajectories to use as few-shot exemplars in the prompts. Each trajectory consisted of multiple thought-action-observation steps.

Model Name	API Selection	Argument Completion	API Ordering	Hallucination
gpt-3.5-turbo	21/100	17/100	11/100	8/100
gpt-4	12/100	6/100	5/100	6/100
Claude instant 1.2	16/100	16/100	6/100	12/100
Claude 2	12/100	14/100	6/100	14/100
LLaMa-7b-chat	74/100	82/100	53/100	65/100

Table 3. Error Statistics for ReAct

2.2.6. REVERSE CHAIN

Reverse Chain Zhang et al. (2023b) is a target-driven prompting approach designed for the purpose of multi-API planning using LLMs. This method decomposes the multi-API planning task into API selection and argument completion based on the API descriptions, specifically, the Reverse Chain performs planning in a reverse manner starting from the final API and then each preceding task is inferred backwards.

Given a user query and a set of APIs along with their descriptions, the first step of the Reverse Chain is to ask the LLM to select an appropriate API to address the query. The selected API will be the final API used in the proposed solution to the query.

The second step is to provide the arguments and argument descriptions of the selected API to the language model and ask it to complete the arguments by either fetching an appropriate argument value from the user query or by specifying another API whose output can complete the argument value or the model can leave the argument empty if it's not needed.

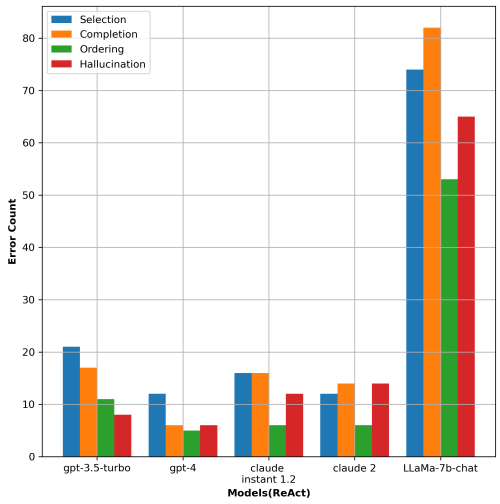

Figure 5. ReAct Error Statistics

The argument completion step is repeated for APIs needed to complete an argument.

This method breaks down a complex task into simpler sub-tasks, and since each step is inferred backwards from the target, this method is less prone to deviation from the intended path. However, since each step is inferred backwards, any error at any step will lead to a completely incorrect solution path.

On paper, this prompting technique seems to have the most potential but we have encountered some difficulties in tuning it to get its most optimal implementation.

2.2.7. Decomposed Prompting

Decomposed Prompting Khot et al. (2023) is an approach that involves breaking down intricate tasks into more manageable sub-tasks, and assigning them to a shared library of tools designed for prompting. The modular structure of Decomposed Prompting allows for optimization, further decomposition, and the easy replacement of prompts or functions.

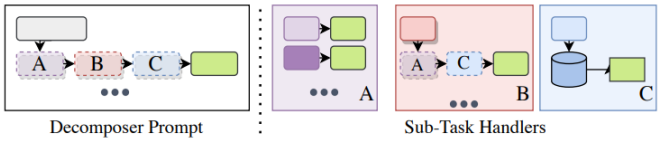

Figure 6. **Decomposed Prompting**, a method for solving complex tasks by breaking them down into simpler sub-tasks. Involves decomposing the original query into sub-queries and then combining the results.

Notably, the flexibility of this method surpasses previous attempts with GPT-3.5 in few-shot prompting. In tasks involving symbolic reasoning, sub-tasks that pose challenges for LLMs are subjected to further decomposition. When faced with complexity due to input length, tasks are recursively broken down into smaller inputs.

On the other hand, GPT-3.5 encounters difficulties when executing decomposed prompting in zero-shot scenarios. To overcome this limitation, a shift towards a few-shot decomposed prompting approach is deemed essential.

In this methodology, explicit examples of decomposed queries derived from the parent query are provided, resulting in improved outcomes.

The model exhibits inconsistent output formats, presenting decomposed queries in varying structural formats with each iteration. Additionally, the model tends to display few-shot examples alongside the primary output.

2.2.8. ExpertPrompting

ExpertPrompting Xu et al. (2023a) is an augmented strategy for instructing LLMs. For each specific instruction, ExpertPrompting first envisions a distinguished expert agent that is best suited for the instruction, and then asks the LLMs to answer the instruction conditioned on such expert identity.

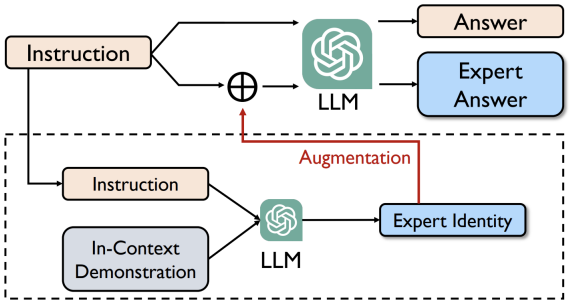

Figure 7. **Expert Prompting**, an augmented strategy for instructing LLMs. For each specific instruction, ExpertPrompting first envisions a distinguished expert agent that is best suited for the instruction, and then asks the LLMs to answer the instruction conditioned on such expert identity.

The expert identity is produced with In-Context Learning. Each expert identity is defined at very delicate granularity using a detailed and elaborate description. ExpertPrompting is also simple to implement, requiring no sophisticated crafting of prompt templates or iterative processes.

We employ an expert software engineer persona with ExpertPrompting. Our trials show that language models show improved performance with the inclusion of ExpertPrompting for our task of addressing user queries.

2.3. Output Structuring

- **Using JSON Mode in GPT**: The GPT model can structure its outputs in JSON format. This mode can be activated either through explicit instructions in the initial prompt or via follow-up prompts during the interaction.
- **Few-Shot Demonstrations**: To ensure that the output is structured correctly, it's beneficial to provide few-shot demonstrations. These examples guide the model in formatting its responses in the desired JSON structure.
- **Post Processing**: After receiving the response from the model, the output string can be parsed to extract individual JSON objects.
- **Creating a Fixed JSON Schema based on model response**: By analyzing the count and arguments of various APIs involved, a fixed JSON schema can be developed. This schema facilitates the use of standard processing techniques, such as JSONFormer, to handle the structured data efficiently.
- **Handling Complex Logic Queries**: The output structure for complex logic queries requires a different approach, which will be discussed in a later section. These queries often involve more intricate data relationships and may necessitate a more flexible or elaborate JSON structure.

2.4. Addition and Modification of Tools

- **Real-time Integration within System Prompt**: If the tool addition or modification is an integral part of the system prompt, regex methods could be applied to identify and manipulate tool-related patterns in the text.
- **In-context learning**: In-context learning would imply that the model learns to recognize and generate tools based on examples and demonstrations provided in the training data.
- **ToolkenGPT**: The central concept of ToolkenGPT involves representing each tool as a unique token. Each tool is represented as a "toolken," and its embedding is inserted into the LLM head, solving the problem of hallucination.
- **Retrieval-based Modification**: Employing a database for storing and retrieving tools. First, given an input prompt, it uses a retriever to fetch relevant documents from a corpus of knowledge. Then, it feeds these documents, along with the original prompt, to an LLM which generates a response.
- **Separate Interface for Tool Management**: Developing a dedicated interface for tool addition and modification. This interface provides a controlled environment for adding, modifying, or removing tools, allowing users to make changes systematically.

2.5. NER for Query Enhancement

Named entity recognition (NER) is a component of natural language processing (NLP) that identifies predefined categories of objects in a body of text.

We employ NER to identify a carefully curated set of entities from the user query. Our entity set currently includes :

- Rev_Org
- User
- Source_Channel
- Issue
- Stage
- Part

Our prompt instructs the language model to perform named entity recognition (NER) prior to answering the query. The query, tagged with the identified entities, is then utilized by the Language Model to adeptly address the user query.

The inclusion of named entity recognition (NER) increases the robustness of the model by preventing important information from being overlooked, thus improving overall model performance. Furthermore, our observations indicate that without instruction to perform NER, the model does not consistently utilize the "who_am_i" tool when necessary. However, NER addresses this issue as it identifies the current user as an entity, enabling the model to utilize this key information when answering the query.

2.6. Tool Documentation over description

In our prompting techniques, providing a few shot examples seemed to work best but even they resulted in a lot of errors with undesirable biased usage. As queries grew more complex, the selection search grew combinatorially and the demonstrations provided failed to generalize to these more complex tasks. Thus we used an alternative to these: tool documentation. We observed that zero-shot prompts with only tool documentation achieved performance on par with few-shot CoT. Tool documentation is significantly more valuable than zero-shot demonstrations, generalizing well to more complex tasks.

2.7. Post Processing

We explore further post-processing the model's response, to deal with the following issues:

- **Output Structuring**: Our experiments show that even when instructed LLMs may struggle to produce output in the desired format. This issue can be solved by

processing the model's output to the desired JSON schema.

- **Hallucinations**: Language Models have the tendency to hallucinate tools and their arguments when a tool apt to address the user query does not exist. Hallucinations can be tackled in post-processing where any tool that the LLM has hallucinated can be removed. This approach also allows us to handle cases where the response should be an empty list.

2.8. Complex Queries

We experimented with the model using a diverse set of queries, some of which cannot be potentially solved by taking the composition of available functions; and might need some additional logic around combining the outputs of those functions.

Notably, certain queries demanded the model to execute complex iterative processes to reach the desired output. One illustrative example was the instruction: "Begin by listing the first 15 work items owned by Eve in the Testing stage. Then, iteratively filter the list to include only those of type issue and task separately, and summarize each subset."

This particular scenario demanded the model to engage in multiple tool calls, sequentially filtering the list based on specific criteria and summarizing the results at each step. However, the model struggled to recognize the iterative nature of the task and consequently performed sub-optimally in such cases.

Conversely, the model demonstrated competence in queries involving BOOLEAN logic. These queries leveraged the model's ability to comprehend and process logical conditions effectively.

This discrepancy in performance highlights the need for refining the model's understanding of iterative processes and complex logic, enabling it to recognize and execute multi-step tasks more efficiently. Further fine-tuning using diverse examples featuring intricate logic could enhance the model's performance at handling a broader scope of complicated queries.

3. Dataset Generation

Dataset Generation is essential for this problem statement, for evaluation, and for fine-tuning. We tried multiple preliminary approaches to automate the generation of queries in zero-shot, few-shot, and using other prompting techniques, especially those that have an intermediate query generation process. However, we quickly realized the need for a small, manually or semi-manually curated seed dataset before scaling it up to provide diversity and complexity.

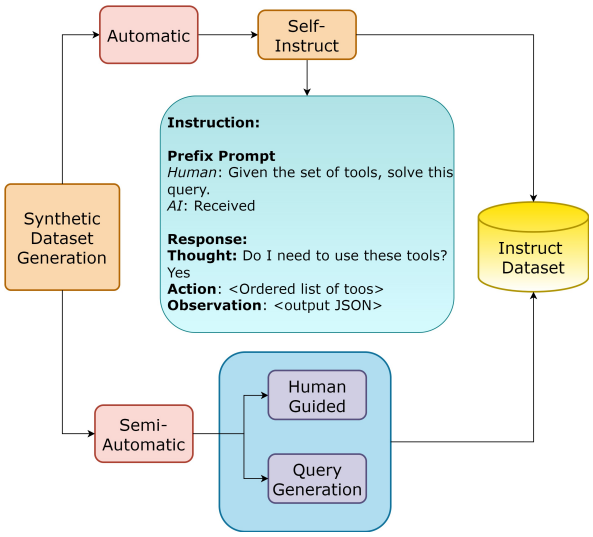

Figure 8. Dataset Generation Pipeline. Creating Instruct Dataset using Automatic: Self-Instruct, Semi-Automatic: Human Guided and Query Generation using LLMs.

3.1. Seed Dataset Generation

We provided the model with a set of APIs and corresponding examples to generate additional queries using a few-shot learning approach. The queries generated, while acceptable, lacked the desired complexity and diversity.

To enhance the output, we directed the model to explore unused tools, embody a customer persona for a chatbot interface, and implement complex logic, including iterative and conditional structure. While the model produced more human-sounding responses, there were challenges in interpreting them. The model especially gave queries that incorporated time-related terms. This issue was effectively resolved by directing the model to avoid such terms. This comprehensive approach aimed to enrich the diversity and complexity of the generated queries.

Additionally, a curated selection of well-formed queries produced by the model was added to the few-shot examples for improved performance.

Ensuring the complexity of the generated queries is crucial. We've categorized the dataset generation seed into three levels: Low, Medium, and High. This categorization is determined by the complexity associated with the number of tasks within each query. Initially, the zero-shot generated queries were brief and typically focused on a single task. However, through the implementation of more sophisticated prompting methods, the number of tasks per query increased. Therefore, the complexity assessment is solely based on the quantity of tasks involved.

3.2. Human-Guided Query Generation

While the input from human experts yields high-quality queries, the process is laborious and time-intensive. To expedite and scale the dataset creation, we are looking into utilizing LLMs like GPT-4 to synthesize instructional data for fine-tuning. However, relying solely on language models can present challenges, such as generating unanswerable or factually incorrect questions, and posing overly simplistic queries answerable through the model's internal knowledge.

We propose a human-guided approach combining human expertise with automated language model generation to tackle these challenges. These queries are used to generate query templates using GPT-4 which are further enhanced by automatically populating them with sampled values from the reference data, amplifying the dataset size.

3.3. Tools Synthesis

Our dataset expansion involved synthesizing tools by feeding documentation inputs through Langchain into GPT-4. This process allowed the model to create new tools based on the structured data from the documentation. Additionally, we manually generated diverse queries using these newly created tools. Hence, our goal is to broaden our array of tools and expand the dataset's scope.

Method	Number of Tools
Zero-Shot	20
Few-Shot	15
CoT	13

Table 4. Tool Dataset Statistics

3.4. Self-Instruct Dataset

In the Self-Instruct framework Wang et al. (2022), the process starts with a compact seed set of manually crafted queries from prior approaches. This set acts as the task pool, randomly selecting queries to stimulate the language model (LM). The LM generates new queries and their corresponding outcomes. The generated outputs go through a careful filtering process to eliminate low-quality or redundant content. The refined data seamlessly integrates back into the initial repository. This iterative bootstrapping algorithm progressively sharpens the LM's ability to follow query patterns, establishing a self-improving cycle that leverages its own generated data for subsequent fine-tuning. This approach is expected to outperform the programmatic generation of new queries from predefined templates, presenting possibilities for a more diverse dataset.

4. Parameter Efficient Fine Tuning

As we test various LLMs on their capabilities to achieve the task, it is clear that fine-tuning the model is a must to get more robust and reliable results. Tuning all parameters of a model is not at all feasible due to resource constraints and is not always an effective approach since it can degrade the general reasoning abilities of the original model and lead them to situations like catastrophic forgetting Lin et al. (2023).

Hence, we leverage Parameter Efficient Fine Tuning Methods (PEFT), to only fine-tune a small subset of parameters of the LLM while freezing most of its parameters. It is important to recognize which part of parameters to tune to get optimal results, so, we plan to explore some different techniques that may prove effective for this particular task.

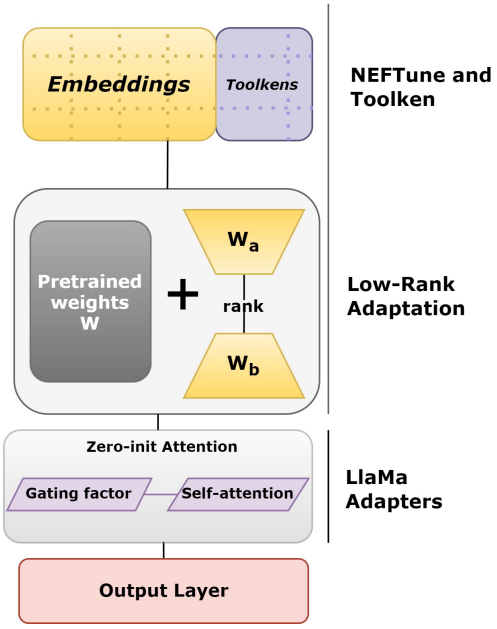

Figure 9. PEFT techniques for finetuning embedding space using NEFTune and Toolkens, updating inner layer parameters using LoRA, and adding additional layers of LlaMa Adapters.

4.1. LLM inner layers finetuning

4.1.1. LoRA

Low-rank adaptation, or LoRA, updates the parameters of specific dense layers by injecting trainable rank decomposition matrices into the architecture. Common practice is to keep the ratio lora_r / lora_alpha = 1:1, so as not to overpower the base model, but we can try to deviate from this approach, our task being very specific to developer queries. LoRA technique fine-tuning is orthogonal to many fine-tuning methods which means it can be combined with many other fine-tuning techniques as well, and, it introduces no in-

Method	Number of Queries	Tools Utilized	Arguments Utilized	Complexity (Low / Med / High)	Executable
Manual with Few Shot Seed	90	3	4	22 / 56 / 12	81
Few Shot	50	2	3	27 / 15 / 8	48
Few shot with CoT	25	3	5	8 / 12 / 5	23

Table 5. Query Dataset Statistics

ference latency compared to a fully fine-tuned model, which is one of the important factors for this task.

LORA has proven to be effective in learning tool capabilities as evident from works by Qiao et al. (2023) and Yang et al. (2023), making this technique worth exploring.

4.1.2. PREFIX-TUNING

Since hard-prompting plays a big role in correctly addressing the input queries by the model, prefix-tuning by Li and Liang (2021) allows us to optimize even the soft-prompts by adding learnable task-specific prefixes to them, which may enhance the model's ability to interpret different tasks in the user query. It is possible that by learning only 0.1% of the parameters prefix-tuning can obtain reasonable performance and is able to extrapolate it equally to unseen queries. We recognize that newly added parameters might introduce inference latency, but testing speed-quality trade-off is something worth investigating for this PEFT technique.

4.1.3. LLAMA ADAPTERS

Extending the ideas of prefix tuning and the original adapter method by Houlsby et al. (2019), researchers proposed LLaMA-Adapter Zhang et al. (2023a), a model-agnostic fine-tuning method, which provides an additional advantage due to zero-init attention which makes the initial tuning more stable without disrupting model's linguistic knowledge. We would want our base model to retain its reasoning abilities while improving its tool-learning capabilities, we can exploit this method to try to achieve this. Also, this method adds additional layers to only a few top and bottom layers, playing to our advantage when it comes to inference latency.

4.2. LLM Embedding Layer Finetuning

4.2.1. NEFTUNE

Noisy Embedding Improve Instruction Finetuning(NEFTune) PEFT method by Jain et al. (2023) is fairly recent and shows that augmentation, as simple as adding noise to embedding vectors, can considerably improve model finetuning. Our model must be able to handle conversational queries while preserving its technical abilities and correctly identifying arguments to give formatted output. The research shows this method to be effective in improving the quality of chatbots and works well with other PEFT methods like QLoRA(Quantized LoRA) by Dettmers et al. (2023). NEFTune avoids model overfitting to the specifics of the instruction-tuning dataset, especially when the dataset is small, and adds no inference latency because no new parameter layers are introduced.

4.2.2. TOOLKENGPT

Fine-tuning model embeddings to learn separate tokens for tools is an approach utilized in ToolkenGPT by Hao et al. (2023). Like NEFTune, this fine-tuning method also focuses on the embeddings. This technique arms the model with special vocabulary for available tools by adding special tokens called "toolkens" to the embedding space, expanding its understanding of the tools available. New tools can be added to the toolset easily, and performance can be improved by fine-tuning the model on tool-specific tasks while keeping inference latency unchanged and updated parameters less. This technique introduces a way to handle newly introduced tools while performing well overall.

4.3. Additional Methods

If the time and resources allow us, we will explore further testing methods like Prompt Tuning Lester et al. (2021), Diff-Pruning Guo et al. (2021), BitFit Zaken et al. (2022), IA3 Liu et al. (2022). All these PEFT methods have not been used for the task exactly like ours, hence it is also possible we would have to apply a variation of the techniques or even come up with a different one to get the results we need. It would be interesting to see which of these would impact the model in what way, whether to induce new capabilities or bring out the already existing ones in our base model.

Technique	% of parameters
In-Context Learning	0 %
Prefix Tuning	0.1 %
LoRA	0.1 to 1 %
LlaMa Adapter	0.01 to 0.1 %
Adapter Tuning	3 to 4 %
Tuning top k layers	20 %
Full Fine Tuning	100 %

Table 6. Parameter percentages for different techniques

5. Latency and Inference

When serving user queries, latency and response times are especially important factors that impact the user experience. We are exploring various approaches aimed at reducing latency, accelerating inference speeds, and reducing memory consumption like precision reduction (float16 or bfloat16) to speed up the model, 8-bit or 4-bit quantization to reduce memory consumption by 2x or 3x, fine-tuning with adapters (LoRA, QLoRA) to improve prediction accuracy on your data which works well in combination with quantization afterward, tensor parallelism for faster inference on multiple GPUs to run large models.

We are also looking at libraries for LLM inference and serving, such as Text Generation Inference, DeepSpeed, or vLLM. These already include various optimization techniques: tensor parallelism, quantization, continuous batching of incoming requests, optimized CUDA kernels, and more.

We are tracking the chain latency and token usage associated with our experiments to evaluate different models. We have tabulated some of those results in the experiments section of our deliverables.

6. Resources

We conducted our assessment of various language models using several conversational AI platforms including claude.ai, the free version of ChatGPT, the LMSYS chat interface, and Forefront.ai. We also leveraged Langchain to implement various complex conversational chains and analyze the performance of multiple pre-trained language models. In addition, we utilized the OpenAI API to access their language models for evaluation.

Forefront.ai hosts the GPT-3.5 and Claude Instant 1.2 models with free public access. A key benefit is the ability to adjust the response temperature setting, improving the reproducibility of test results. Specialized assistants are available on this platform to assist with unique domains, helping maximize the benefits of ExpertPrompting Xu et al. (2023a). Leveraging the software engineer assistant allows for further refinement in handling user queries.

Evaluation on ChatGPT lacked robustness and reproducibility as the platform does not permit customizing the temperature parameter. Similar constraints with Claude.ai have made thorough testing of Claude 2 challenging, highlighting the need for access to Anthropic API to facilitate more robust evaluation (We are yet to receive access).

The LMSYS Chat interface provides access to various language models but usability is impacted by a character limit for prompts. Multi-part prompts are necessary and thus consistency across responses cannot be guaranteed.

We have also recently secured access to ChatGPT Plus which would facilitate the dataset generation process as well as allow us to try some AI-based evaluation techniques.

Limited compute resources have posed challenges for thorough testing methodologies. Due to computing constraints, testing on GPT-4 was restricted in scope. We have additionally applied for access to alternative APIs, including Anthropic, which would expand our capabilities for evaluation. Broader access to application programming interfaces (APIs) and obtaining additional computational resources for dataset generation and training would facilitate a more robust assessment and allow for a more comprehensive examination of approaches.

7. Deliverables

Drive link for experimentation, prompts and seed dataset: Team 15 Mid Eval.

References

T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

D. Guo, A. M. Rush, and Y. Kim. Parameter-efficient transfer learning with diff pruning, 2021.

S. Hao, T. Liu, Z. Wang, and Z. Hu. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings, 2023.

N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. de Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp, 2019.

N. Jain, P. yeh Chiang, Y. Wen, J. Kirchenbauer, H.-M. Chu, G. Somepalli, B. R. Bartoldson, B. Kailkhura, A. Schwarzschild, A. Saha, M. Goldblum, J. Geiping, and T. Goldstein. Neftune: Noisy embeddings improve instruction finetuning, 2023.

T. Khot, H. Trivedi, M. Finlayson, Y. Fu, K. Richardson, P. Clark, and A. Sabharwal. Decomposed prompting: A modular approach for solving complex tasks, 2023.

T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213, 2022.

B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning, 2021.

X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation, 2021.

Y. Lin, L. Tan, H. Lin, Z. Zheng, R. Pi, J. Zhang, S. Diao, H. Wang, H. Zhao, Y. Yao, and T. Zhang. Speciality vs generality: An empirical study on catastrophic forgetting in fine-tuning foundation models, 2023.

H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022.

S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023.

S. Qiao, H. Gui, H. Chen, and N. Zhang. Making language models better tool learners with execution feedback, 2023.

Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, R. Tian, R. Xie, J. Zhou, M. Gerstein, D. Li, Z. Liu, and M. Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face, 2023.

Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. Self-instruct: Aligning language model with self generated instructions, 2022.

J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. Finetuned language models are zero-shot learners, 2022.

B. Xu, A. Yang, J. Lin, Q. Wang, C. Zhou, Y. Zhang, and Z. Mao. Expertprompting: Instructing large language models to be distinguished experts, 2023a.

Q. Xu, F. Hong, B. Li, C. Hu, Z. Chen, and J. Zhang. On the tool manipulation capability of open-source large language models, 2023b.

R. Yang, L. Song, Y. Li, S. Zhao, Y. Ge, X. Li, and Y. Shan. Gpt4tools: Teaching large language model to use tools via self-instruction, 2023.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023a.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models, 2023b.

L. Yuan, Y. Chen, X. Wang, Y. R. Fung, H. Peng, and H. Ji. Craft: Customizing llms by creating and retrieving from specialized toolsets, 2023.

E. B. Zaken, S. Ravfogel, and Y. Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2022.

R. Zhang, J. Han, C. Liu, P. Gao, A. Zhou, X. Hu, S. Yan, P. Lu, H. Li, and Y. Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention, 2023a.

Y. Zhang, H. Cai, Y. Chen, R. Sun, and J. Zheng. Reverse chain: A generic-rule for llms to master multi-api planning, 2023b.

Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang. Toolqa: A dataset for llm question answering with external tools, 2023.