Mid-Term Report

Team 35

Abstract

As Large Language Models have become an integral part of our lives, aiding us in nearly every sphere of our everyday routines. From ChatGPT to LLaMa and Bard, numerous models have been established that are breaking limits in multiple fields. As the developments made using LLMs are breaking ground at light year speed, the improvements to LLMs are not very far behind. As a part of the High Prep Statement: AI Agent 007: Tooling up for Success organized by the Indian Institute of Technology, Madras, in collaboration with Dev Rev, we explore another improvement to Large Language Models, arming the models with the power to choose from a toolbox consisting of specific tools to help answer queries. It has been shown that many of the tool-augmented LLMs still mostly rely on closed LLM APIs. We attempt to build on open-source LLMs with minimal human intervention to achieve performance which is competitive with closed LLMs.

1. Introduction

In the domain of domain-specific answering queries, we wish to answer queries of users in the best manner possible. For this purpose, we arm ourselves with language models, along with a toolbox of specific tools. The aim is to enable users to interact with DevRev using natural language. The idea is to understand the queries, map them to the tools necessary along with the relevant arguments, and then execute the tasks.
It has been shown in multiple previous works that many of the tool-augmented LLMs still mostly rely on closed LLM APIs. As we reach a stage where there is a lot of exposure of information and workflows to the closed LLMs, there are increasing security and robustness risks. Finally, we attempt to build on open-source LLMs with minimal human intervention to achieve performance which is competitive with closed LLMs.

2. Problem Statement

A Language Model L has a set of tools T and is given a user query Q. The model is supposed to answer Q using the existing tools, the relevant arguments for the tools chosen, and the composition of the tools. Moreover, the set of the tools T is dynamic, and new tools could be added, or existing tools could be modified or deleted. The model should be able to handle all such cases gracefully.

3. Literature Review

The domain of domain-specific question answering and the orchestration of tools to address user queries represent a challenging area that has garnered considerable attention. Several research efforts and practical implementations have explored similar facets, providing insights into effective strategies and methodologies.

3.1. LLM

LLMs, or Large Language Models, are sophisticated artificial intelligence models designed to understand and generate human-like text. These models, such as GPT-3.5, Bard, etc., leverage extensive training on diverse language data to exhibit advanced language understanding, allowing them to perform tasks like text generation, translation, and comprehension with remarkable proficiency.

3.1.1. GPT-4 [21]

GPT-4 is a powerful multimodal model excelling in language tasks and image processing. It surpasses GPT-3.5, achieving human-level performance on exams like the Uniform Bar Exam. Improvements include safety enhancements and reduced hallucinations. Limitations include no real-time learning, occasional reasoning errors, and vulnerability to adversarial inputs. Emphasis is on predictable scaling and infrastructure development. Success in multiple languages and improved visual capabilities are noted. Caution is advised in high-stakes contexts due to potential limitations. Ongoing efforts for transparency, safety interventions, and independent auditing are highlighted.

3.1.2. LLaMa [29]

This introduces the LLaMA series of Large Language Models (LLMs) ranging from 7B to 65B parameters, challenging the notion that larger models necessarily lead to better performance. LLaMA models, despite being smaller, outperform GPT-3 and compete with much larger models on various benchmarks. The training approach involves diverse data sources, transformer architecture modifications, and efficient implementation. LLaMA excels in tasks like common sense reasoning, question answering, reading comprehension, mathematical reasoning, and code generation. This also addresses potential issues such as toxicity and bias, evaluates carbon footprint, and emphasizes the importance of open access for advancing research.

3.2. Augmented Language Models

[20] talks about how Augmented Language Models (ALMs) address limitations in Large Language Models (LLMs) by combining reasoning and external tools, categorized across three axes. The paper explores diverse reasoning methods and tools integration, emphasizing an iterative LM calling approach and information retrieval strategies. Efforts to teach ALMs include few-shot prompting, fine-tuning, and reinforcement learning, with acknowledged challenges like limited supervision and model overfitting. Potential benefits encompass truthfulness, uncertainty reduction, interpretability, and enhanced capabilities, while ethical concerns revolve around the trustworthiness of ALM predictions using external tools.

3.3. Data Augmentation using LLMs

Large Language Models are now being used extensively for augmenting data and producing synthetically generated examples. What works in their favour is they can simulate multiple scenarios and can purposely introduce minor errors, anomalies, and perturbations to the dataset, and their data paraphrasing abilities. Multiple papers such as [4] and [15] report this in more detail.

3.4. Retriever Augmented Generation

[3] explains Retriever Augmented Generation or RAG as an AI framework for improving the quality of LLM-generated responses by grounding the model on external sources of knowledge to supplement the LLM's internal representation of information. LLMs only know how to relate words by their statistical properties, and not their meanings. RAG It ensures that the model has access to the most current, reliable facts, and that users have access to the model's sources, ensuring that its claims can be checked for accuracy and ultimately trusted. It also reduced hallucinations.

3.5. Prompt Engineering

In the early stages of language model development, prompts were straightforward queries provided to the model to elicit desired responses. However, researchers and practitioners soon recognized the potential for enhancing model performance through carefully crafted prompts. The process evolved from simple queries to more nuanced instructions that guide the model towards the desired output.

3.5.1. Zero Shot Setting

Zero-shot setting involves training a model on a task and then evaluating its performance on a related but unseen task without any additional training. In this context, prompt engineering is crucial for conveying the task or generating context that allows the model to generalize effectively. Researchers have explored ways to design prompts that enable zero-shot transfer learning across diverse domains, showcasing the adaptability of language models to novel tasks.

3.5.2. One Shot Setting

One-shot setting represents a paradigm where a model is evaluated on a task with only a single example per class. Effective prompt engineering in this setting is vital as it guides the model to comprehend and generalize effectively from a solitary instance. Striking a balance between specificity and generality in prompts is crucial, allowing the model to extract meaningful patterns and make accurate predictions with minimal exposure to task-specific examples.

3.5.3. Few Shot Setting

Few-shot setting extends this idea by providing the model with a small amount of task-specific examples during evaluation. This setting requires prompts that efficiently leverage the provided examples to make accurate predictions. Researchers have delved into crafting prompts that strike the right balance between specificity and generality, allowing models to grasp task nuances with minimal examples.

3.6. LLMs for API Calls

The approach involves orchestrating sub-tasks through APIs. It employs various large language models (LLMs) to strategize and map these sub-tasks to specific actions, where the actions correspond to API calls. This allows for efficient planning and distribution of tasks, leveraging the capabilities of diverse LLMs to handle different aspects of the overall problem. One approach involves utilizing a large language model (LLM) as a planner, drawing inspiration from [27]. We can also employs latent language representations for skill induction and planning in a different domain, mapping tasks to relevant functions, with a focus on API selection in the given context as discussed in [26].

3.6.1. Language Models as Zero-shot Planners [11]

This paper investigates how high level tasks can be broken down into a series of specific steps that are actionable. They achieve the same without any training. They use two evaluation criterion: Execution, which checks for if the actionable plan satisfies common-sense and should be parsable, and Correctness, which can be determined based on human evaluations. Using one example high-level task and its annotated action plan from the demonstration set, they obtain text completion results using temperature or nucleus sampling. They then ensure that each environment action has a corresponding predicted action phrase using cosine similarity, which is followed by autoregressive trajectory correction.

3.6.2. API-Bank [16]

This paper introduces Multi-agent, an automated method using five agents to generate cost-effective training data for tool-augmented Large Language Models (LLMs). This eliminates the need for human labelers, saving 98% compared to manual annotation. The proposed approach addresses challenges in training data creation by simulating dependencies among domains, APIs, queries, and abilities. Additionally, the paper presents API-Bank, a comprehensive benchmark for evaluating tool-augmented LLMs. It comprises 1,008 domains, 2,211 APIs, and 2,202 dialogues. API-Bank is designed to assess LLMs' abilities in planning, retrieving, and calling APIs. The proposed Multi-agent method is compared favorably to existing benchmarks, emphasizing its diversity, realism, and coverage.

3.6.3. Gorilla [23]

Gorilla is a system enhancing large language models (LLMs) by enabling them to interact with external APIs, transforming LLMs into primary interfaces for dynamic knowledge bases. It introduces a novel self-instruct finetuning and retrieval paradigm for accurate API selection, demonstrated through the APIBench benchmark. Gorilla's retrieval-aware training adapts to API documentation changes, excelling in accuracy and hallucination reduction compared to GPT-4. Gorilla outperforms GPT-4 in API functionality accuracy and hallucination reduction, demonstrating adaptability to test-time API changes. It emphasizes the importance of LLMs using tools, introduces APIBench for evaluation, and highlights Gorilla's superiority in various scenarios, contributing to systematic evaluation and training methods.

3.6.4. ToolLLM [25]

ToolLLM uses a three step process which involves the collection of APIs, generating instructions using ChatGPT for subsets of APIs that involve both single-tool and multi-tool scenarios, and then generating a solution path using a depth-first search baed decision tree. They also develop an automatic evaluator which comprises of two metrics, *pass rate*, which measures the ability of the LLM to execute an instruction, and *win rate*, which compares the quality and usefulness of two solution paths. They also use a neural retriever for the APIs, which eliminates the need for manual selection of APIs.

3.7. Frameworks

3.7.1. ToolAlpaca [28]

ToolAlpaca is a framework that equips compact language models with generalized tool-use abilities. It constructs a diverse toolset using large language models (LLMs) to generate structured documentation for various tools. The multi-agent simulation framework involves user, assistant, and tool executor agents represented by LLMs to automatically generate tool-use instances. The resulting corpus contains 426 tools from 50 categories with diverse instances. Quality evaluation indicates high proficiency, and experiments demonstrate that fine-tuning on the ToolAlpaca corpus empowers compact language models for generalized tool-use, outperforming other models. The study highlights the effectiveness of ToolAlpaca in handling diverse scenarios and the importance of toolset diversity for generalized learning.

3.7.2. LangChain [1]

LangChain is a sophisticated framework designed for the development of applications utilizing language models. This framework facilitates the creation of applications that are context-sensitive, such as prompt instructions, specific examples, and relevant content, and reasoning-enabled.

3.7.3. TaskMatrix.AI [17]

TaskMatrix.AI envisions an AI ecosystem linking foundation models with diverse off-the-shelf models and systems via APIs for versatile task completion. The architecture includes a Multimodal Conversational Foundation Model (MCFM) for user communication and code generation, an API Platform storing millions of APIs, an API Selector recommending relevant APIs, and an Action Executor executing generated codes. MCFM learns multimodal inputs, proposes solution outlines, and leverages feedback for continual improvement. The API Platform uses a unified documentation schema, and the API Selector employs module strategies to find suitable APIs. Reinforcement Learning with Human Feedback (RLHF) optimizes MCFM and API Selector. User feedback is relayed to API developers, fostering iterative improvements. TaskMatrix.AI aims to seamlessly integrate neural and symbolic systems, bridging digital and physical task domains.

3.8. QA Approaches

The proposed approach involves treating the problem as a Question-Answering (QA) task. The language model formulates precise questions directed at an external retriever or semantic model, seeking information on which tools and their corresponding functionalities to use. For instance, questions like ”which API for finding the sum of 2 numbers?” guide the model to dynamically adapt to changes in the set of tools. This strategy ensures flexibility and scalability, allowing the model to gracefully handle modifications or additions to the toolset while responding to user queries in a targeted and effective manner. This approach is discussed in [30]. Another approach involves breaking down a complex question into sub-questions sequentially. Each sub-question is generated based on the original query and the QA pairs of preceding sub-questions as discussed in [32] . Subsequently, follow-up questions are posed to an external system or API using the generated sub-questions. The answers obtained from these follow-up questions then contribute to refining the model's understanding, ultimately leading to an answer for the initial complex question as discussed in [24].

3.8.1. Text Modular Networks [13]

This introduces Text Modular Networks (TMNs), a framework for complex question-answering by decomposing questions into sub-questions answered by existing QA models. TMNs include a next-question generator trained with distant supervision. They present MODULARQA, a system implementing TMNs, achieving competitive performance on DROP and HotpotQA. The method demonstrates versatility, robustness, sample efficiency, and interpretable reasoning. The approach involves training sub-task question models and generating training decompositions with distant supervision hints. MODULARQA's cross-dataset performance showcases its effectiveness.

3.8.2. Successive Prompting for Decomposing Complex Questions [9]

Successive Prompting method is proposed for tackling complex reasoning tasks, particularly demonstrated on the DROP dataset. They use a modular approach involving question decomposition (QD) and answering (QA) stages. In their experiments, they compare this approach with chain-of-thought prompting and observe a performance improvement. They employ in-context learning with a large language model and fine-tuning using annotations from DROP. The results show that Successive Prompting outperforms other methods, achieving a 5.1% F1 improvement over the state-of-the-art model. The authors discuss the strengths and weaknesses of their approach through qualitative examples and analysis. They emphasize the effectiveness of modular systems that decompose and delegate tasks to specialized models.

3.9. External Tools as Problem Solvers

The mentioned papers adopt a coding paradigm for problem-solving by utilizing external tools (APIs) as analogous to coding functions. [19] focuses on adaptive reasoning, synthesizing programs to sequentially execute tools for diverse queries. [22] aims at enabling large language models to perform automatic multi-step reasoning using a variety of tools. [7] disentangles computation from reasoning in numerical tasks, treating them as coding challenges. Common to all is the emphasis on adaptability, automation of reasoning through sequential tool-use, and the ability to accommodate new tools or modifications.

3.9.1. Chameleon [19]

The Chameleon model uses an LLM-based planner to assemble a sequence of tools to compose various tools for accomplishing complex reasoning tasks. It generates a plan given an input query, along with the planning task instructions, descriptions of the modules, some demonstration examples, and some constraints. Using the plan generated, the modules for each step are executed sequentially, using the input for the current module and all the cached information, which is then updated at each time step.

3.10. Fine-Tuning Approaches

With recent developments in LLM, researchers explored different directions to make model specialist with generalization capabilities and methods to finetune models in parameter efficient way without having access to massive computing.

3.10.1. In-context Learning (ICL) [5]

This paper aims to alleviate task-specific fine-tuning by describing "in-context learning", which conditions the model on a natural language instruction and/or a few demonstrations of the task and the model is then expected to complete further instances of the task simply by predicting what comes next. They believe that while it shows inferior performance with respect to fine-tuning, it can show significant improvement with scale.

3.10.2. Parameter-efficient Fine-Tuning (PEFT) [18]

This paper compares PEFT and ICL, bringing out the drawbacks of ICL such as significant computational costs involved in processing all input-target pairs and inferior performance to fine-tuned models. PEFT involves minimal updates to a few added or selected parameters. PEFT methods also allow mixed-task batches, for example, concatenating different prompt embeddings to each example for several different tasks. While PEFT also requires fine-tuning and some computational power for inference, the paper claims that PEFT is much more computationally efficient when considering both fine-tuning and inference in comparision to ICL.

In synthesizing these diverse but interconnected strands of research, AI Agent 007 can draw inspiration and guidance for its mission to adeptly handle domain-specific queries through intelligent tool orchestration. The ensuing sections will build upon these foundations to articulate the design and implementation of a robust solution tailored to DevRev's objectives.

3.11. Performance Improvement Approaches

The proposed approach involves enhancing confidence and reliability in the generated sequence of API calls through a majority voting mechanism that considers multiple approaches. This draws motivation from the concept of [12] where aggregating outputs from diverse methods can lead to more robust outcomes. Additionally, the strategy incorporates a verifier that assesses whether the sequence of sub-tasks (API calls) accomplished thus far effectively achieves the desired task. This idea is inspired by the concept of [8], emphasizing the importance of validating the generated sequences for accuracy and task fulfillment.

3.12. Bonus Task Approaches

The [6] approach involves leveraging large language models to create a logic layer or program atop retrieved APIs for query handling. This not only facilitates the generation of API sequences but also allows the incorporation of logic, offering potential solutions for the bonus section. [14] adopts a modular strategy to solve complex tasks by breaking them down, with the capability to employ recursion, which aligns with one of the proposed approaches for the bonus section. The idea is to propose a program on the generated API call sequence, introducing logic such as error handling (e.g., try-except in Python) to address exceptions that may arise from API calls, a relevant consideration for the bonus section.

4. Experimental Setup

4.1. Dataset

We utilize GPT-3.5 and GPT-4 for data augmentation along with manual human intervention to generate tools for smoother functionality. We further created additional data points with different levels of utilization in terms of the tools and arguments that will be used to resolve each query. Table 2 describes the number of examples available utilizing each tool and Table 1 contains the dataset statistics for tools and user query and API call pair.

Type of Data	**Count**
New Tools Added	8
Old Tools	9
Total Tools	**17**
Single Tool Single Argument	113
Single Tool Multiple Arguments	44
Multiple Tools	101
Tool with No Arguments	5
Empty Answers	1
Total Examples	**264**

Table 1. Number of data samples generated for various types of queries

Tool Name	**Count**
works_list	185
summarize_objects	23
prioritize_objects	16
get_similar_work_items	13
add_work_items_to_sprint	22
get_sprint_id	19
search_object_by_name	12
create_actionable_tasks_from_text	10
who_am_i	10
create_project	5
create_sprint	5
create_work_item	11
test_execution_report	5
code_review_feedback	4
update_work_item_status	10
assign_work_item	4
automate_workflow	4

Table 2. Number of data samples generated for each of the tools

4.2. Evaluation Metric

The following are the metrics which we have used to measure the correctness or performance of our methods:

4.2.1. Exact Match

For each answer, we check if the predicted JSON is the same as the ground truth JSON. This returns 1 if it is the same and 0, if they are not.

While checking the similarity, we check if the order of the tools used in the prediction and the ground truth are the same. Arguments are evaluated in order invariant way for each tool.

4.2.2. F_1 Score

To evaluate the similarities between two API calls we introduce weighted F1 scores inspired by F1 score for context question answering task. Before we define the overall F_1 Score, we define the following,

$$\text{Precision}(x) = \frac{\text{Total correct } x}{\text{Total predicted } x}$$

$$\text{Recall}(x) = \frac{\text{Total correct } x}{\text{Total required } x}$$

$$F_1(x) = \frac{2\ \text{Precision}(x)\ \text{Recall}(x)}{\text{Precision}(x) + \text{Recall}(x)}$$

where, $x \in \{\text{tools, arguments, values}\}$.

Now, we define the overall F_1 Score as follows,

$$F_1 = \frac{W_1 x_1 + W_2 x_2 + W_3 x_3}{W_1 + W_2 + W_3}$$

where,

$$\begin{aligned} x_1 &= F_1(\text{tools}) \\ x_2 &= F_1(\text{arguments}) \\ x_3 &= F_1(\text{values}) \\ W_1 &= 1 \\ W_2 &= \text{Recall}(\text{tools}) \\ W_3 &= \text{Recall}(\text{tools})\ \text{Recall}(\text{arguments}) \end{aligned}$$

For tools, we calculate the precision, recall and F_1 by comparing the predicted and ground truth tools without checking if the order of tools is correct or not.

For arguments, we calculate the precision, recall and F_1 by comparing the predicted and ground truth arguments of correctly predicted tools.

For values, we calculate the precision, recall and F_1 by comparing the predicted and ground truth argument values of correctly predicted arguments.

4.2.3. Hallucination Rate

We define hallucination rates individually for tools and arguments.

$$\text{HRate}(x) = \frac{\text{Total predicted } x \text{ not present in the tools list}}{\text{Total predicted } x}$$

where, $x \in \{\text{tools, arguments}\}$.

The overall hallucination rate is defined as,

$$\text{HRate} = \frac{\text{HRate(tools)} + \text{HRate(arguments)}}{2}$$

4.3. Experiments

The following are the experiments we tried:

4.3.1. Benchmarking Experiments using GPT Models

We benchmark predictions of GPT-3.5 and GPT-4 [21] using zero-shot and few-shot prompts with one, two, and three examples. The evaluation uses a mixture of the provided as well as the augmented data points. The prompt used for the experiments is mentioned in Listing 1

```
Use the following tools and answer the
    given question using given answer format

{{ API DOCUMENTATION in JSON format}}

Generated answers for any given question
    should have following format

{{ ANSWER FORMAT PROVIDED IN PROBLEM
    STATEMENT}}

Following are the examples of question-
    answer pairs for the abovementioned
    tools

EXAMPLES FOR REFERENCE in  case of FEW SHOT
     SETTING

To reference the value of the ith tool in
    the chain, use $$PREV[i] as argument
    value. i = 0, 1, .. j-1; j = current
     t o o l s  index in the array
If the query could not be answered with the
     given set of tools, output an empty
    list instead.

Empty list should look like []. Whenever
    you want to access information about
    current user to pass it further, you
    need to make a call to who_am_i api tool
    .

Answer the following question based on the
    instructions provided above.
question:
answer:{
"question":
"answer":
}

Since API documentation is part of the text
    , adding new tools or updating and
    deleting existing ones is
    straightforward, as only the prompt
    needs to be modified. Hence this prompt
    template-based setup can be generalized
    without any supervised fine-tuning.
```

Listing 1. Prompt template for benchmarking experiments on GPT models

5. Results and Discussion

We have observed that GPT-4 starts to generate convincing API calls when two examples are provided. In some cases, we can observe a mismatch between the reference tool and the predicted tool. Here, F1 refers to the F1 score. Also, the Exact Match is order-invariant for the arguments and not the order of the tools.

Model	Setting	Exact Match	F1	HR
GPT-4	0 examples	0.236	0.914	0.012
	1 example	0.250	0.908	0.006
	2 examples	0.305	0.906	0.006
GPT-3.5	0 examples	0.027	0.331	0.006
	1 example	0.236	0.826	0.007
	2 examples	0.195	0.756	0.004
	3 examples	0.285	0.691	0.006

Table 3. Results

6. Limitations of Current Approaches

Even though GPT-4 is performing second to none in few-shot settings, the current approach of few-shot prompts comes with limitations like the cost associated, and security risks for companies exposing their back-end to a third party company. Another major limitation is the context window length. None of the APIs it can handle is subjective to the context window length of the model, which can limit a simple few-shot prompt-based approach.

7. Conclusion and Future Work

For the mid-term evaluation, we worked on data augmentation for generating new API tools, and data points for the provided as well as the new tools using GPT models(GPT-3.5 and GPT-4). We further conducted benchmarking experiments using GPT models on the augmented data. As mentioned in the limitation section, the current approaches are restricted by context window size to consider more APIs, and hence, along with fine-tuning open source models like LLaMA with in-context learning, we plan to explore efficient tool retriever techniques that can provide a set of relevant tools. The retriever can help to optimize the approaches by only providing the relevant APIs. This ensures that we can reduce the prompt length significantly which directly resolves the issue of context window length in small open source LLMs like LLaMA-2. It can reduce the cost associated with input tokens in third party APIs like GPT-3.5 and 4.

8. Link to Code

Link to code can be found here

References

[1] Introduction — Langchain — python.langchain.com. `https://python.langchain.com/docs/get_started/introduction`. [Accessed 25-11-2023]. 3

[2] Okapi BM25 - Wikipedia — en.wikipedia.org. `https://en.wikipedia.org/wiki/Okapi_BM25`. [Accessed 25-11-2023].

[3] What is retrieval-augmented generation? — IBM Research Blog — research.ibm.com. `https://research.ibm.com/blog/retrieval-augmented-generation-RAG`. [Accessed 25-11-2023]. 2

[4] Luiz Bonifacio, Hugo Abonizio, Marzieh Fadaee, and Rodrigo Nogueira. Inpars: Data augmentation for information retrieval using large language models, 2022. 2

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. 4

[6] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers, 2023. 5

[7] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks, 2023. 4

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. 5

[9] Dheeru Dua, Shivanshu Gupta, Sameer Singh, and Matt Gardner. Successive prompting for decomposing complex questions, 2022. 4

[10] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023.

[11] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022. 2

[12] Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models, 2023. 5

[13] Tushar Khot, Daniel Khashabi, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Text modular networks: Learning to decompose tasks in the language of existing models, 2021. 4

[14] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks, 2023. 5

[15] Bohan Li, Yutai Hou, and Wanxiang Che. Data augmentation approaches in natural language processing: A survey. *AI Open*, 3:71–90, 2022. 2

[16] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms, 2023. 3

[17] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, Yun Wang, Linjun Shou, Ming Gong, and Nan Duan. Taskmatrix.ai: Completing tasks by connecting foundation models with millions of apis, 2023. 3

[18] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022. 4

[19] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models, 2023. 4

[20] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented language models: a survey, 2023. 2

[21] OpenAI. Gpt-4 technical report, 2023. 1, 6

[22] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models, 2023. 4

[23] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023. 3

[24] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models, 2023. 4

[25] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. 3

[26] Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language, 2022. 2

[27] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face, 2023. 2

[28] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases, 2023. 3

[29] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. 1

[30] Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. Break it down: A question understanding benchmark, 2020. 4

[31] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.

[32] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models, 2023. 4

.1. Appendix A

```
[
    {
        "question_id":"2",
        "question":"Summarize high severity
             tickets from the customer
            UltimateCustomer",
        "answer":[
            {
                "tool_name": "
                    search_object_by_name",
                "arguments": [
                    {
                    "argument_name": "query
                        ",
                    "argument_value": "
                        UltimateCustomer"
                    }
                ]
            },
            {
                "tool_name": "works_list",
                "arguments": [
                    {
                        "argument_name": "
                            ticket.rev_org",
                        "argument_value":
                            ["$$PREV[0]"]
                    },
                    {
                        "argument_name": "
                            ticket.severity
                            ",
                        "argument_value":
                            ["high"]
                    },
                    {
                        "argument_name": "
                            type",
                        "argument_value":
                            ["ticket"]
                    }
                ]
            },
            {
                "tool_name": "
                    summarize_objects",
                "arguments": [
                    {
                        "argument_name": "
                            objects",
                        "argument_value": "
                            $$PREV[1]"
                    }
                ]
            }
        ]
    }
]
```

Listing 2. An illustrative Question-Answer Pair in JSON Format

```
[
    {
        "tool_name":"
            add_work_items_to_sprint",
        "tool_description":"Adds the given
            work items to the sprint",
        "arguments":[
            {
                "name":"work_ids",
                "description":"A list of
                    work item IDs to be
                    added to the sprint",
                "type":"array of strings",
                "example":""
            },
            {
                "name":"sprint_id",
                "description":"The ID of
                    the sprint to which the
                    work items should be
                    added",
                "type":"str",
                "example":""
            }
        ]
    }
]
```

Listing 3. Representation of Provided Tools in JSON Format