AI Agent 007
MidTerm Report for the Inter IIT Tech Meet 12.0
Team 99

***Abstract*—This study aims to outline our methodologies for addressing the problem statement, AI Agent 007, specifically focusing on the utilization of tool augmentation of Language Models (LLMs). Our strategy encompasses a comprehensive grasp of the problem at hand, an extensive review of relevant literature, and a well-designed experimental plan to evaluate the efficacy of various techniques that align with our objectives. In this discussion, we briefly outline our proposed solution design and the methodology we intend to employ in order to assess its accuracy on the provided set of tools. Additionally, we incorporate our experimental technique to evaluate the adaptability of our system in dynamically accommodating new tools. In conclusion, we incorporate our proposed strategies to conduct experiments and validate predictions for prompts pertaining to the bonus task.**

I. Introduction

Although Large Language Models (LLMs) have demonstrated impressive capabilities in handling basic natural language tasks, their performance in more complex activities, such as mathematical computations, has been underwhelming. The objective of this problem statement is to provide a solution that allows an LLM (Language Model) to precisely predict the appropriate tool(s) and their accompanying parameters for invoking an API call in order to provide an accurate response to a given query.

This assignment serves as a prototypical illustration of tool augmentation in the context of Language Model-based Learning.

The issue of tool augmentation of LLMs has historically been addressed through two primary approaches:

- **Fine-tuning:** Finetuning refers to the process of training a pre-existing Language Model (LM) by providing it with annotated data specifically related to tools, such as API documentation.
- **In-context Learning:** In the realm of learning, the concept of in-context learning refers to the practice of providing a pre-trained model with illustrative instances that demonstrate the proper use of various tools. The process of fine-tuning Language Models (LLMs) using tool demonstration data can incur significant expenses and is often limited to a predetermined selection of tools. Furthermore, the incorporation of additional tools in real-time for this paradigm incurs significant costs in terms of the resources required for training.

The utilization of the in-context learning paradigm addresses these challenges. However, due to the restricted length of the context, only a limited number of demonstrations can be included, resulting in less than ideal comprehension of the tools. Furthermore, in situations when there is a wide array of tools available, the effectiveness of in-context learning may be significantly diminished.

In recent years, there has been an emergence of current studies that explore a novel method for modifying the lexicon of the LLM, with the aim of incorporating tool augmentations. The current approach is in its early stages and lacks sufficient experimentation and literature to substantiate its efficacy within this paradigm.

II. Literature Review

In this section, we will provide a comprehensive overview of the research literature that has been reviewed up until the mid evaluation, focusing on the problem statement and our comprehension of the proposed methodologies.

A. Chain of Thought (CoT) prompting

The Chain of Thought (CoT) [13] prompting method encourages the LLM to engage in a thought process before answering the question. Such an approach shows impressive performance improvements in reasoning tasks. The model can perform step-by-step reasoning by creating a table without further fine-tuning by using a table header with column names in the form of "$|step|question|response|$" as a prompt.The 2-dimensional tabular structure of TabCoT [3], which is a recent variation improving the traditional CoT approach, allows for improved unlocking of the step-by-step reasoning capabilities of LLMs, transforming the linear "chain of thought" process into a more structured one.

Least-to-most prompting: A prompting strategy which reduces a complex problem into a list of sub-questions and sequentially solves the sub questions.Each sub-question is solved with the answer to previously solved sub-questions. Zero-Shot Tab-CoT method contains two steps:

1) table generation
2) answer extraction

Table Generation Prompt: To make use of the 2-dimensional structure of the table, we replace the natural language prompt with a table-generation prompt (e.g., "$|step|question|response|$"), which serves as the header of

the table.

If each row of the table is regarded as a step, the row-by-row table generation process will become a step-by-step reasoning process. Within each step (row), we have multiple columns, each of which contributes certain detail towards the current reasoning step.

$$LLM(x, c) = \begin{bmatrix} c_1 & c_2 & \dots & c_{n-1} & c_n \\ t_{1,1} & t_{1,2} & \dots & t_{1,n-1} & t_{1,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_{m,1} & t_{m,2} & \dots & t_{m,n-1} & t_{m,n} \end{bmatrix}$$

The conventional CoT method (and the standard prompting method) strongly favors a text-based LLM under the zero-shot setting. In contrast, the TabCOT approach works well with both types of LLMs, but the code-based version can give it an additional boost in performance. Table generation resembles the code generation process – both involve structured procedures that are highly organized and follow a step-by-step process. Comparing Tab-CoT approach with conventional CoT, we can conclude that the proposed table-generation prompt is able to significantly better unlock the strong reasoning abilities within the code-based LLM.

Scheme Design : For the understanding of the current table scheme design ("$|step|subquestion|process|result|$") along with the other four variations ,by removing one of the four columns . It was noticed that removing some of them leads to a drastic decrease in overall performance.

Case studies : Code underperforms text with conventional COT but yields better result with Tab-COT "code-davinci-002". It shows better reasoning ability demonstrating a more concise and straightforward reasoning process.

Symbolic Reasoning : Tab-CoT is evaluated on two symbolic reasoning datasets: Coin Flip and Last Letter .These tasks focus on some specific problems. Table schemes are split into 3 parts :

1) General: the table scheme that can be generally applied to most text questions.
2) Domain-specific : the table scheme that can be adapted to a specific domain
3) Task-specific : the scheme that can only be adopted by a single task

Common sense Reasoning tasks do not have a fixed answering pattern. Therefore, providing chain-of-thought samples is not enough to make up for the lack of commonsense knowledge.

Limitations : This approach is applicable to language models pretrained with tables. Limited improvement in common sense reasoning tasks suggests that its effectiveness may depend on the specific task and the level of structured reasoning required. Its inability to perform better than traditional COT in "text-davinci-002" like language models is also a limitation.

B. ReAct

Large language models (LLMs) have shown promise in language understanding and interactive decision-making, but their reasoning and action plan generation abilities have been studied separately. This paper examines the interleaved generation of reasoning traces and task-specific actions using LLMs. This allows for greater synergy between the two, as reasoning traces initiate, track, and update action plans, while actions enable the model to gather additional information from external sources like knowledge bases or environments. ReAct is applied as a technique to several language and decision-making tasks, demonstrating its superiority over current baselines and enhanced human interpretability and trustworthiness. For question answering (HotpotQA [14]) and fact verification (Fever [12]), ReAct addresses hallucination and error propagation in chain-of-thought reasoning by utilizing a Wikipedia API to generate more interpretable task-solving trajectories than baselines without reasoning traces. On two interactive decision-making benchmarks (ALFWorld [11] and WebShop [16]), ReAct surpasses imitation and reinforcement learning methods by 34% and 10%, respectively, with only one or two in-context instances. [15]

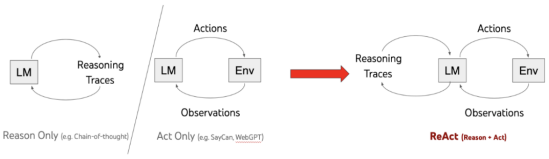

Fig. 1. Diagrammatic represenatation of the ReAct Approach

The idea of ReAct is simple:The agent's action space is augmented to $\hat{A} = A \cup L$, where L is the space of language. An action $\hat{a}_t \in L$ in the language space, which we will refer to as a thought or a reasoning trace, does not affect the external environment, thus leading to no observation feedback. Instead, a thought $\hat{a}_t$ aims to compose useful information by reasoning over the current context c_t, and update the context $c_{t+1} = (c_t, \hat{a}_t)$ to support future reasoning or acting. [15] This thought or reasoning trace does not affect the external environment i.e. wiki $\hookrightarrow$ no obs feedback. Language space is unlimited, as a result of which learning in this action space is difficult, hence the usage of a frozen LLM (examples are ToolkenGPT and PaLM API) is prompted with few shot in-context. This methodology is employed across a range of linguistic and decision-making assignments, exhibiting its efficacy when compared to leading benchmarks. Additionally, it enhances human interpretability and trustworthiness. Specifically, in question answering tasks such as HotpotQA [14] and fact verification endeavors like FEVER [12].ReAct adeptly confronts the prevalent challenges linked to hallucination and error propagation in chain-of-thought reasoning. This accomplishment is realized through its interaction with a simplistic Wikipedia API, producing task-solving trajectories that replicate human-like reasoning and outperform baseline models in terms of interpretability, all without leaving a discernible reasoning trace. In summary, ReAct demonstrates superiority over the combined approaches of HotpotQA [14] and Fever [12], which encompass question-answering and fact verification.

ReAct successfully mitigates common challenges associ-

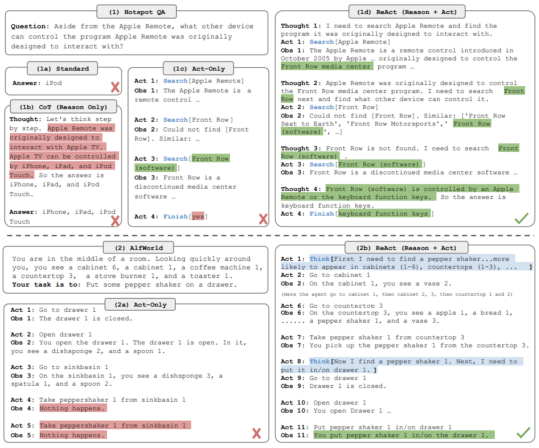

Fig. 2. Comparison of 4 prompting methods, (a) Standard, (b) Chain-of-thought (CoT, Reason Only), (c) Act-only, and (d) ReAct (Reason+Act), solving a HotpotQA [14] question. (2) Comparison of (a) Act-only and (b) ReAct prompting to solve an AlfWorld [11] game. In both domains, we omit in-context examples in the prompt, and only show task solving trajectories generated by the model (Act, Thought) and the environment

ated with hallucination and error propagation in Chain-of-Thought reasoning through its interaction with a straightforward Wikipedia API. Moreover, it generates task-solving trajectories that emulate human-like reasoning, exhibiting superior interpretability compared to baseline models devoid of reasoning traces.

In terms of performance, ReAct surpasses the limitations of reinforcement learning methods by achieving an absolute success rate improvement of 34% in the ALFWorld scenario [11]. Similarly, when evaluated in the WebShop context [16], ReAct outperforms reinforcement learning methods by an absolute success rate improvement of 10%, all while being prompted with just one or two in-context examples.

Idea and Thought Process: ReAct (Reason + Act) at first glance appears to be a combination of reason and act where sometimes a model can only do one of them, whereas ReAct can do both simultaneously. ReAct is a general paradigm to combine reasoning and acting with language models for solving diverse language reasoning and decision-making tasks. 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 dynamic reasoning to create, maintain, and adjust high-level plans for acting (reason to act), while also interact with the external environments (e.g. Wikipedia) to incorporate additional information into reasoning (act to reason).
ReAct is better and outperforms vanilla action generation models while being at par with CoT.

Why use over CoT? "Chain-of-Thought" reasoning operates as a static black box wherein the model relies on its internal representations to generate thoughts without being grounded in the external world. This characteristic imposes limitations on the model's capacity for reactive reasoning or updating its knowledge dynamically. Consequently, this inherent structure can give rise to challenges such as fact hallucination and the propagation of errors throughout the reasoning process.

Advantages: Given the integration of decision-making and reasoning capabilities within a large language model, ReAct possesses a distinctive set of features:

1) Intuitive and easy to design: Creating ReAct prompts is a straightforward process, as human annotators simply articulate their thoughts in natural language while describing the actions they have taken.
2) General and flexible: Thanks to its adaptable thought space and the occurrence format of thought-action, ReAct is applicable to a broad spectrum of tasks characterized by diverse action spaces and distinct reasoning requirements. This versatility extends to tasks such as, but not limited to, question-answering, fact verification, text games and web navigation.
3) Performant and robust: ReAct demonstrates robust generalization to new task instances, achieving consistently superior performance compared to baselines that exclusively emphasize reasoning or acting. Notably, this proficiency is exhibited even when learning from a minimal set of one to six in-context examples, underscoring its adaptability across diverse domains.
4) Human-aligned and controllable: ReAct holds the potential for providing interpretable sequential decision-making and reasoning processes, enabling human inspectors to easily scrutinize both the reasoning and factual correctness. Additionally, humans have the capability to exert real-time control or correction over the agent's behavior through thought editing, further enhancing the transparency and adaptability of the system.

The features and the method of implementation strongly point towards our implementation of the Problem Statement. Specifically how it can implement reasoning and action at the same time. It can be helpful in understanding which tools to use when presented with multiple similar tools. Reasoning is one of the stronger points to check for tools. Also, each reasoning and action can be implemented in a sequence for possible multiple-tool implementation. Being human controllable is an important context when it comes to financial issues. Robustness shows strong generalization for tools added on the fly.

C. ToolFomer

Toolformer has tried to equip LLMs with a capability of interacting with various tools via API calls [10]. The process that has been followed in the paper is as follows:

Generation of Language Model(LM) dataset: Given only a few human-written examples of API usage, the researchers had a language model (LM) annotate a vast language modeling dataset with potential API calls. They then employed a self-supervised loss to discern which API calls truly enhanced its ability to predict future tokens. Subsequently, the LM underwent fine-tuning based on the API calls it deemed beneficial. This approach, being agnostic to the dataset, allows

its application to the same dataset used for the model's initial pretraining. This ensures that the model retains its generality and language modeling capabilities. Toolformer, which is based on a pretrained GPT-J model, achieves much stronger zero-shot results, clearly outperforming a much larger GPT-3 model and other baselines on various tasks.

Approach: Given a dataset $\mathcal{C} = \left\{\mathbf{x}^1, \ldots, \mathbf{x}^{|\mathcal{C}|}\right\}$ consisting of plain texts, the researchers initially transformed this dataset into $\mathcal{C}^*$ a dataset augmented with API calls, through a three-step process. Leveraging the in-context learning capability of the model, a substantial number of potential API calls were sampled. Subsequently, these API calls are executed, and the obtained responses are assessed for their utility in predicting future tokens, serving as a filtering criterion. After the filtering process, API calls for various tools are merged, resulting in the augmented dataset $\mathcal{C}^*$. The final step involved fine-tuning the model M on this augmented dataset.

Fig. 3. Key steps in the Toolformer approach, illustrated for a question answering tool: Given an input text $\mathbf{x}$, a position i and corresponding API call candidates $c_i^1, c_i^2, \ldots, c_i^k$ are sampled. Toolformer then executes these API calls and filtesr out all calls which do not reduce the loss L_i over the next tokens. All remaining API calls are interleaved with the original text, resulting in a new text $\mathbf{x}^*$.

Evaluation:

1) **Experimental Setup:** The researchers explore the capability of the approach to enable the model to utilize tools without additional supervision, allowing it to autonomously determine when and how to employ the available tools. To assess this, a range of downstream tasks is chosen, assuming the utility of at least one of the considered tools, and performance is evaluated in zero-shot settings.
   The models which are compared in the evaluation are:
   a) GPT-J: A regular GPT-J model without any finetuning
   b) GPT-J + CC: GPT-J finetuned on C, our subset of CCNet without any API calls
   c) Toolformer: GPT-J fine tuned on C , our subset of CCNet augmented with API calls.
   d) Toolformer (disabled): The same model as Toolformer, but API calls are disabled during decoding
2) **Evaluation on downstream Tasks:** The researchers assess all models across various downstream tasks using a prompted zero-shot setup. In this configuration, models are directed to address each task using natural language, without providing any in-context examples. Opting for the more challenging zero-shot setup, the study aims to determine if Toolformer functions effectively in scenarios where users don't pre-specify which tools to use and how to use them to solve a particular problem.
   a) SQuAD [9], GoogleRE and T-REx: In each subset, the objective is to fill in a missing fact in a brief statement, such as a date or place. To accommodate variations in tokenizations and added complexity from not explicitly specifying the need for a single word, the evaluation criterion is slightly more lenient than exact match. The assessment involves verifying if the correct word is among the first five words predicted by the model. GPT-J models, when evaluated without tool use, exhibit comparable performance, while Toolformer consistently surpasses these baseline models. Notably, Toolformer outperforms OPT (66B) and GPT-3 (175B) despite their larger sizes. The superiority stems from the model's autonomous choice to employ the question answering tool for necessary information in almost all instances (98.1%), with only a small fraction using an alternative tool (0.7%) or none at all (1.2%).
   b) Math Dataset: The study evaluates mathematical reasoning proficiency using ASDiv [5], SVAMP [6], and the MAWPS [8] benchmark. In consideration of the zero-shot setup, a more lenient evaluation criterion is applied. Since the expected output is consistently a numerical value, the assessment involves verifying the accuracy of the first number predicted by the model. While GPT-J and GPT-J + CC perform about the same, Toolformer achieves stronger results even when API calls are disabled. With API calls the performance becomes more than double for all tasks, and also clearly outperforms the much larger OPT and GPT-3 models.
   c) Question answering : The study examines Web Questions, Natural Questions and TriviaQA [4]—the three question-answering datasets.In the evaluation process, the focus is on determining if the correct answer is present within the first 20 predicted words by a model, rather than necessitating an exact match.Although GPT-J and GPT-J + CC exhibit similar performance, Toolformer achieves superior results even without API calls. This suggests that the model benefits from fine-tuning on numerous examples of API calls and their results, enhancing its mathematical capabilities. Allowing the model to make API calls more than doubles performance across all tasks, surpassing the significantly larger OPT and GPT-3 models.
   d) Multilingual Question Answering: Toolformer and all baseline models undergo evaluation on MLQA, a multilingual question-answering benchmark. Each question is accompanied by a context paragraph in English, with the question itself presented in Arabic, German, Spanish, Hindi, Vietnamese, or Simplified Chinese. To successfully complete the task, the model must comprehend both the paragraph and the question, potentially benefiting from translating the question into English. The evaluation metric is the percentage of instances where the model's generation, limited to 10

words, includes the correct answer. Toolformer's performance is not consistently superior to vanilla GPT-J. This is primarily attributed to a decline in performance for certain languages due to fine-tuning on CCNet, potentially indicating a distribution shift compared to GPT-J's original pre training data. OPT and GPT-3 exhibit weak performance across all languages.

e) Temporal Dataset: To assess the effectiveness of the calendar API, they tested all models on TEMPLAMA [1] and a newly introduced dataset, DATESET. Employing the same evaluation as the original LAMA dataset for both tasks, Toolformer consistently outperforms all baseline models in both TEMPLAMA and DATESET.

Advantages :

1) Performs very well on downstream tasks except multilingual question answering even when API calls are disabled.
2) It overcomes limitations like lack of mathematical ability and a limited and older knowledge base of other language models by using external tools which can perform these tasks much better rather than relying completely on the langage models.

Disadvantages:

1) It does not perform well on tasks involving multilingual support.
2) Results show the inability of Toolformer in using tools in chains.
3) Models trained with Toolformer to often be sensitive to the exact wording of their input when deciding whether or not to call an API.
4) Toolformer currently does not take into account the tool-dependent, computational cost incurred from making an API call.

D. Gorilla

The Gorilla LLM is an finetuned language model based on LLaMA, which exhibits superior performance compared to GPT-4 in generating API calls specifically designed to address this particular difficulty [7]. When Gorilla is utilised in conjunction with a document retriever, it exhibits a robust ability to adjust to modifications in documents during testing, hence facilitating flexible updates and changes in API versions. Gorilla effectively addresses the problem of hallucination, which is frequently experienced while directly prompting LLMs. In order to assess the model's performance, Gorilla LLM utilises APIBench, an extensive dataset comprising over 1600 HuggingFace, TorchHub, and Tensorflow Hub APIs. The integration of the retrieval system with Gorilla showcases the potential of LLMs to enhance their accuracy in using tools, staying informed with regularly revised documentation, and so enhancing the dependability and relevance of their results.

The operational framework of Gorilla LLM is supported by its API augmentation. The system utilises application programming interfaces (APIs) to retrieve real-time data, so facilitating the provision of responses that are both extremely pertinent and current. The versatility of Gorilla LLM is made possible by the use of retrieval-aware fine-tuning, enabling it to effectively engage with diverse deep-learning models.

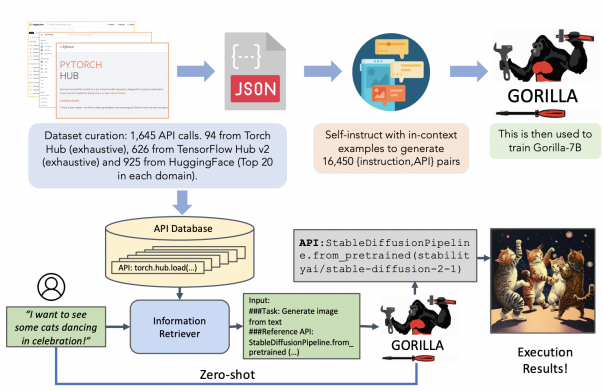

Fig. 4. Gorilla: A system for enabling LLMs to interact with APIs.During inference (lower half), Gorilla supports two modes - with retrieval, and zero-shot. In this example, it is able to suggest the right API call for generating the image from the user's natural language query.

Limitations:

1) Gorilla only supports API integeration from Hugging Face, TorchHub and TensorFlowHub. Thus it is limited in terms of the number of APIs it supports.
2) Integeration of a new API is challenging as the code is closed sourced, thus annihilating the possibility of finetuning on custom tools. The current flow of adding an API involves submitting a Pull Request with the required tool in a specified format on the Gorilla repository.
3) Gorilla is observed to consistently perform poorly on multi domain prompts. eg:"Write an Article" works fine, so does "Translate text to Japanese". But "Write an article in English and translate to Japanese" fails to provide a satisfactory response. Thus, Gorilla can handle such cases only when the API in question takes care of the multi domain nature of the query.
4) Moreover, the output generated by Gorilla is exclusively in the format of Python code. In order to obtain the intended output, it is necessary to execute the Python code provided by Gorilla LLM. However, if the hardware specs do not meet the minimal requirements for running the code, the desired output will not be achieved.

E. ToolkenGPT

ToolkenGPT aims to augment Large Language Models (LLMs) with tools in a way that is cost effective and open to integrate tools on the fly with minimal retraining [2]. Tools are represented as tokens, hence the word toolken.The model learns an embedding for each toolken, and stores these embeddings in its vocabulary. Unlike learning toolken embeddings from very extensive explanations of the tools in a text document, the authors encode toolkens from very extensive demonstrations of the tools. This helps encode the

implicit semantics of a tool, which can never be inferred from surface texts. This is what we like to call the "Show vs. Tell strategy".

During prediction, the model is initially in the reasoning mode. When a tool is predicted, it switches to a tool mode where the arguments for the specified tool are predicted. Once this is done, the model reverts back to the reasoning mode.

Advanced prompting techniques like the Chain of Thought [13] and the Tabular Chain of Thought (Tab-CoT) [3] can be used to improve the performance further.

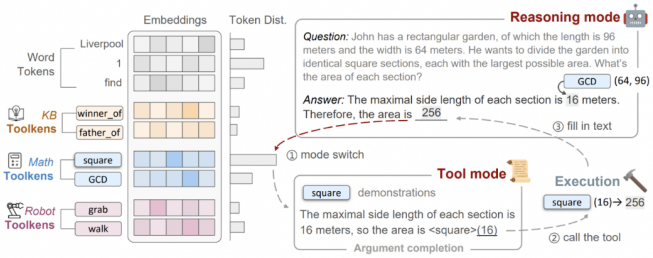

Fig. 5. Overview of ToolkenGPT framework. Toolken embeddings are appended to the language model head like regular word tokens. In the "reasoning mode" for solving the problem, the LLM generates text as usual, except that any plugged-in toolkens are also considered for the next token generation. Once a toolken is predicted, (1) the LLM switch to the "tool mode", which provides a few demonstrations of the same tool to complete the arguments. Then, (2) the tool call is executed, and (3) the result is sent back to the text to continue the reasoning mode until the final answer is generated.

Advantages:

1) Utilizes the power of in-context learning by using extensive demonstrations of a tool to learn its embedding. This is possible with minimum GPU overhead since no gradient flow is required through the parameters of the base LLM, in other words, this idea can easily be applied on a Frozen LLM.
2) Easy to add tools on the fly as it is simply a matter of extending the vocabulary and training on a new set of demonstrations.

Disadvantages:

1) Context switching between the reasoning mode and tool mode requires additional overhead.

III. Our Experimental Setup

Our experimental setup consists of 4 parts:

- M1 → An LLM to predict the Tool required for the given prompt
- M2 → An LLM to extract the arguments for the predicted tool(s)
- Prompt Designer → A python script to create a prompt of the original prompt and the predicted tool
- JSONifier → A script to bundle the tools and arguments together into a JSON output

The comprehensive sequence of events can be delineated as follows:

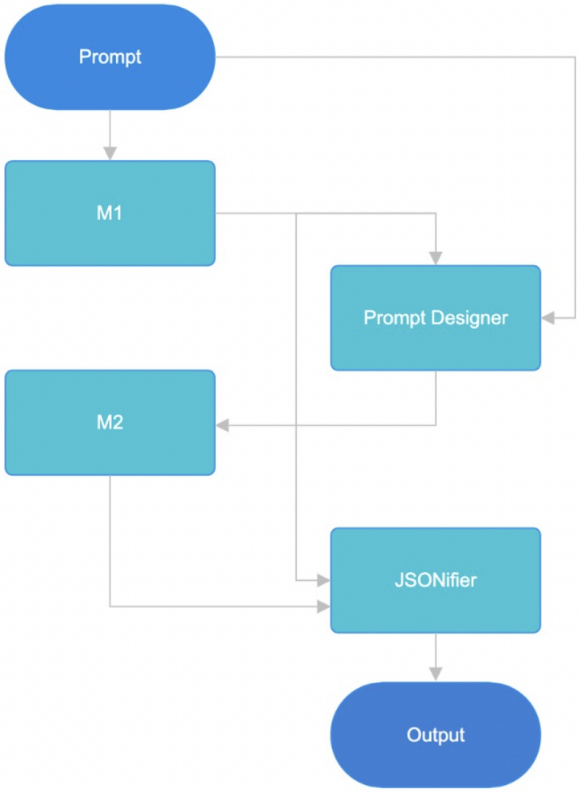

Fig. 6. Schematic representation of our Experimental Setup

- The initial input is directed to M1, a specialized model designed to forecast the necessary tool(s) for the provided query
- The anticipated tool(s) are subsequently inputed into the prompt designer, which generates a prompt tailored to the tool(s) along with illustrative instances. This process enables the one-shot prediction of arguments using M2
- The tools anticipated by M1 and the arguments predicted by M2 are subsequently inputed into the JSONifier, which combines them to provide a JSON output

IV. Experimental Methodology

The experimental methodology employed in this study is a comparison of the finetuning and in-chat prompting methodologies for the Language Models (LLMs) under investigation. The initial design consideration that arises pertains to the selection of LLMs. Commonly selected options encompass LLaMa, GPT, Falcon 7b, and Falcon 40b. The Falcon 40b is presently excluded from our experimentation due to its specific hardware prerequisites.

Our subsequent design choice entails the use of either finetuning or in-chat prompting techniques. The determination of this conclusion will rely on the outcomes derived from a series of several experiments that will be conducted in the following manner:

- M1 and M2 both left un-finetuned and rely only on in-chat prompting

- M1 is finetuned on answering for tools and M2 is unfinetuned and relies only on in-chat prompting
- M1 and M2 are both fine-tuned for tool prediction and argument extraction respectively

An additional factor to consider in the experiments will be the selection of Language Models (LLMs), as previously mentioned. This approach necessitates doing numerous rounds of the aforementioned tests, employing various options for M1 and M2, and subsequently selecting the most accurate outcome.

In order to incorporate an extra tool, our proposed approach involves providing the user with an endpoint within the finetuning and Prompt Designer script. This endpoint will be executed internally, facilitating the integration of the tool into our existing repertoire. The efficacy of the approach can solely be assessed through experimental means, hence rendering the assertion empirical.

An additional feature that will be assessed during the experimentation pertains to the bonus task. Our intention is to develop intricate inquiries that employ a blend of the existing to ols and supplementary logic, such as mathematical operations and iterative processes, in order to evaluate the effectiveness of our system. Subsequently, an examination of the setup would enable us to identify the necessary modifications for addressing the supplementary task.

It is hypothesized that the incorporation of logical operations, mathematical operations, and loops into the output of the tools can be achieved through appropriate adjustments to the prompt designer script.

V. Query Generation

In order to evaluate our system, we will employ the tools provided in the problem specification and generate artificial prompts using the accompanying examples. The aforementioned prompts will constitute the dataset that will be utilized for all of our experimental procedures. In order to assess the performance of our model in terms of incrementalism (the incorporation of new tools in real-time) and the bonus job, we will employ a range of testing methodologies.

VI. Evaluation Method

The accuracy measure will be employed to assess the JSON file produced, and a manual evaluation of said file will be conducted. In the context of tool usage, an exact match is deemed accurate if the tools consistently provide the anticipated outcome, whereas any deviation from this outcome is seen as incorrect. The accuracy of each experiment is determined using the standard formula and thereafter reported. Also, the performance metric of tokens/$ will be reported for each of the discussed approaches.

References

[1] Bhuwan Dhingra et al. "Time-aware language models as temporal knowledge bases". In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 257–273.

[2] Shibo Hao et al. *ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings*. 2023. arXiv: 2305.11554 [cs.CL].

[3] Ziqi Jin and Wei Lu. *Tab-CoT: Zero-shot Tabular Chain of Thought*. 2023. arXiv: 2305.17812 [cs.CL].

[4] Mandar Joshi et al. *TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension*. 2017. arXiv: 1705.03551 [cs.CL].

[5] Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. *A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers*. 2021. arXiv: 2106.15772 [cs.AI].

[6] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. *Are NLP Models really able to Solve Simple Math Word Problems?* 2021. arXiv: 2103.07191 [cs.CL].

[7] Shishir G. Patil et al. *Gorilla: Large Language Model Connected with Massive APIs*. 2023. arXiv: 2305.15334 [cs.CL].

[8] Syed Rifat Raiyan et al. *Math Word Problem Solving by Generating Linguistic Variants of Problem Statements*. 2023. arXiv: 2306.13899 [cs.CL].

[9] Pranav Rajpurkar et al. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. 2016. arXiv: 1606.05250 [cs.CL].

[10] Timo Schick et al. *Toolformer: Language Models Can Teach Themselves to Use Tools*. 2023. arXiv: 2302.04761 [cs.CL].

[11] Mohit Shridhar et al. *ALFWorld: Aligning Text and Embodied Environments for Interactive Learning*. 2021. arXiv: 2010.03768 [cs.CL].

[12] James Thorne et al. *The Fact Extraction and VERification (FEVER) Shared Task*. 2018. arXiv: 1811.10971 [cs.CL].

[13] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL].

[14] Zhilin Yang et al. "HotpotQA: A dataset for diverse, explainable multi-hop question answering". In: *arXiv preprint arXiv:1809.09600* (2018).

[15] Shunyu Yao et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. 2023. arXiv: 2210.03629 [cs.CL].

[16] Shunyu Yao et al. *WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents*. 2023. arXiv: 2207.01206 [cs.CL].