CREATOR: Disentangling Abstract and Concrete Reasonings of Large Language Models through Tool Creation

Cheng Qian[1], **Chi Han**[2], **Yi R. Fung**[2], **Yujia Qin**[1], **Zhiyuan Liu**[1*], **Heng Ji**[1*]
[1]Tsinghua University, [2]University of Illinois at Urbana-Champaign
qianc20@mails.tsinghua.edu.cn

arXiv:2305.14318v1 [cs.CL] 23 May 2023

Abstract

Large Language Models (LLMs) have demonstrated significant progress in utilizing external APIs as tools for various tasks. However, their tool-using ability is limited by the availability of suitable APIs and the instability of implicit reasoning, particularly when simultaneously engaging in reasoning about plans and actual calculations. To address these limitations, we propose CREATOR, a novel framework that empowers LLMs to create their own tools through documentation and code realization. CREATOR disentangles the LLM's ability into two distinct phases: abstract tool creation and concrete decision execution, which results in improved LLM performance. We evaluate CREATOR on two established benchmarks: MATH, which consists of challenging math competition problems, and TabMWP, which includes diverse tabular contents for problem-solving. Remarkably, CREATOR significantly outperforms existing chain-of-thought (CoT), program-of-thought (PoT), and tool-using baselines on these two benchmarks. Additionally, we present a new dataset, Creation Challenge, comprising 2K diverse questions, to highlight the necessity and benefits of LLMs' tool creation ability in effectively addressing these problems. Furthermore, our research reveals that leveraging LLMs as tool creators facilitates knowledge transfer, and LLMs exhibit varying levels of tool creation abilities, enabling them to flexibly tackle diverse situations. Our study represents a promising avenue for maximizing the potential of LLMs and advancing toward truly intelligent and adaptable AI systems.

1 Introduction

Recent years have witnessed remarkable progress in the development of large language models (LLMs) including GPT-3 (Brown et al., 2020), Codex (Chen et al., 2021), PaLM (Chowdhery et al., 2022), LLaMA (Touvron et al., 2023), ChatGPT (OpenAI, 2022) and the more recent GPT-4 (OpenAI, 2023). These models show excellent abilities in in-context learning, code generation, and a wide range of other NLP tasks, pushing the potential of LLMs closer towards Artificial General Intelligence (Bubeck et al., 2023). Despite these great successes, the existing LLMs still have limitations, including the inability to identify or respond to up-to-date information, frequent failures in providing clear and accurate mathematical results, and instability in reasoning over a long chain of logic (Trivedi et al., 2022; Komeili et al., 2022; Patel et al., 2021; Hendrycks et al., 2021; Lu et al., 2022b). To address these concerns, a line of research has been inspired to equip LLMs with external tools to ease their memorization burden and enhance their expertise (Qin et al., 2023). For instance, incorporating tools like a question answering (QA) system or web search engine allows LLMs to learn when and how to access external resources for problem-solving (Nakano et al., 2021; Schick et al., 2023). Recent research also incorporates additional external tools for LLMs, such as GitHub resources, neural network models (e.g. Huggingface library), code interpreters (e.g. Python interpreter), etc. (Gupta and Kembhavi, 2022; Surís et al., 2023; Shen et al., 2023; Liang et al., 2023; Lu et al., 2023). These tools require LLMs to give detailed plans before leveraging tools in solving complex problems.

However, the tool-augmented LLMs still encounter certain challenges (Chen et al., 2022; Gupta and Kembhavi, 2022; Schick et al., 2023; Surís et al., 2023), and we especially consider the following aspects: (1) Most existing work tends to concentrate on a limited number of tools, while the range of potential novel tasks remains virtually unlimited. Consequently, when faced with a new type of problem, it becomes challenging to identify an existing tool that is well-suited for its

*Corresponding author.

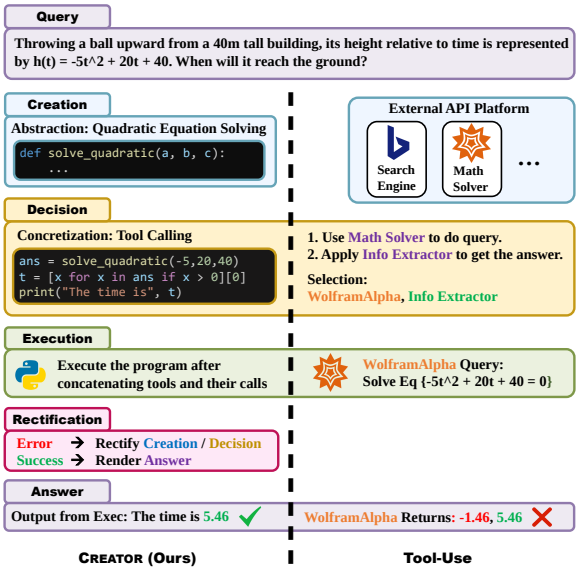

Figure 1: The difference between CREATOR and a general tool use framework.

solution. (2) The current reasoning process employed by language models for determining the optimal utilization of tools is inherently complex. It encompasses extensive planning throughout the entire task-handling process, thereby imposing a substantial cognitive burden on the models and necessitating a considerable learning cost. (3) The current pipelines about tool-using lack a specific and automatic error-handling mechanism after retrieving execution results. This leaves room for improvement in terms of the framework's accuracy and robustness.

In this paper, we aim to address these challenges from a novel perspective: instead of letting the LLMs act as the ***users*** of tools, we enable them to be the ***creators*** of tools and solve problems with higher accuracy and flexibility. Motivated by this, we present our tool creation framework, CREATOR, which exploits LLMs' ability to create tools and do rectifications based on current settings before answering the specific problem. As illustrated in Figure 1, we present the differences in pipelines between CREATOR and a general tool-using framework. The tool-using framework focuses on how to use reasonings to better select and plan the usage of APIs, while we focus on how to diversify the choice of tools, disentangle different levels of reasonings, and increase the framework's robustness as well as accuracy. Specifically, CREATOR can be divided into four stages:

- **Creation**: Create generally applicable tools through documentation and code realization, leveraging LLM's ability to do abstract reasoning based on the problem.
- **Decision**: With related tools at hand, decide when and how to use the tool to solve the problem.
- **Execution**: Execute the program, in which the LLM applies the tool to solve the problem.
- **Rectification**: Make modifications to the tools and decisions based on the results from execution.

With this design, CREATOR distinguishes itself from traditional run-time tool-using in that (1) instead of applying the given APIs of a limited number, we leverage LLMs to create tools with higher generality, reusability, and variety; (2) the tool creation module devised could offload the LLM's cognitive burden and disentangle its ability to do abstract reasonings (creation of generalizable tools) and concrete reasonings (decision making with details); (3) our framework leverages code as the medium for tool creation, which is more sensitive to errors, and enables automatic rectification of the tools and decisions based on the error tracebacks.

To assess the effectiveness of our design, we first conduct tests on CREATOR using two existing benchmarks: MATH (Hendrycks et al.) and TabMWP (Lu et al., 2022a). The MATH dataset contains challenging and diverse math competition problems, while TabMWP contains a wide range of tabular contexts used for problem-solving.' Notably, ChatGPT built on CREATOR achieves an average accuracy of 59.7% and 94.7% respectively on the MATH and TabMWP dataset, surpassing the standard chain-of-thought (CoT) (Wei et al., 2022), program-of-thought (PoT) (Chen et al., 2022) and tool-using baselines by large margins.

As existing benchmarks are not specifically designed to evaluate tool creation, we further propose the Creation Challenge dataset, which comprises novel and challenging problems inadequately solved with existing tools or code packages. By utilizing this dataset, we showcase the necessity and advantages of LLMs' tool creation ability. Moreover, we present experimental results and case studies demonstrating how tool creation also promotes knowledge transfer, and how LLMs possess different levels of tool creation ability that help them better adapt to different problem settings.

2 Related Work

Large Language Models. Large language models (LLMs) have gained significant attention in recent years due to their remarkable performance in handling various NLP tasks, following in-context demonstrations and instructions, and generating high-quality texts and codes (Brown et al., 2020; Chen et al., 2021; Chowdhery et al., 2022; Touvron et al., 2023). A line of studies focuses on how to effectively prompt the LLMs to generate reasonings to solve the problems. Wei et al. (2022) first propose chain-of-thought prompting by giving case examples, while Kojima et al. extend it to zero-shot settings. Besides, another line of studies focuses on using instructions to guide LLMs' high-quality output and align LLMs' behaviors with human expectations. Among these methods, Wang et al. (2022) and Longpre et al. (2023) design instruction-following data from existing datasets, while Chung et al. (2022); Touvron et al. (2023); Liu et al. (2023) use instruction-tuning method to boost LLMs' performances. Our work is built upon these research areas and leverages these methods as part of our framework or baseline for solving complicated problems.

Tool Using of Language Models. To solve the models' inherent limitations such as the inability to get up-to-date information (Trivedi et al., 2022; Komeili et al., 2022) and give accurate calculation results (Patel et al., 2021; Lu et al., 2022b), recent studies begin to focus on using external tools to help boost models' ability. One area of study aims to boost specific task performance by augmenting LLMs with various types of tools including scratch pad (Nye et al., 2021), search engine (Shuster et al., 2022), QA system, and calculator (Schick et al., 2023), etc. With these studies as foundations, more recent works begin to integrate LLMs' tool-using abilities into a pipeline to further showcase LLMs' potential in task planning, tool calling, and result synthesis (Wu et al., 2023; Shen et al., 2023; Liang et al., 2023). Despite the impact these attempts have brought, we are the first to take a step further by awakening the model's potential to create tools instead of merely learning and using tools to solve problems.

Reasoning and Execution with Program. Reasoning with programs (or codes) is an emerging field in NLP, whose goal is to leverage programs to do complicated computational reasoning instead of using natural language thoughts. After program reasoning, the execution results from the generated code will be returned for LLM's further use. The first work by Chen et al. (2022) shows how applying code generation in reasoning could achieve SoTA performance on various math datasets, while Gao et al. (2022) further demonstrates the potential of reasoning with programs on symbolic and algorithmic benchmarks. These efforts present a code-based chain-of-thought with linear logic but produce no enhanced tools capable of being reused or tested. As the concept of tool-using emerges, more recent studies begin to incorporate code interpreters (e.g. Python interpreter) as external tools (Lu et al., 2023). However, in CREATOR, we use code as the medium for tool creation instead of an external tool for calling. Our framework also excels over PoT as we devise the tool creation stage, code rectification stage, and disentangle the logic in complex reasonings.

3 Design of CREATOR

Distinct from previous frameworks for tool use, CREATOR leverages the tool creation ability of LLMs by incorporating special modules. Our framework comprises four stages: creation, decision, execution, and rectification, as illustrated in Figure 2. Initially, the LLMs, such as ChatGPT, receive a given query and generate generalizable tools accompanied by documentation and realization for future utilization. Next, the LLM is employed to determine the appropriate timing and methodology for employing the tool in problem-solving. Once a decision is reached, an interpreter will carry out the decision and provide feedback, which the LLM then uses to rectify the tool or the generated decision if any error occurs. To formalize CREATOR framework, we illustrate the meaning of all the notations we will use in Table 1. The following sections will present the formalization, important details, and design purpose of each stage.

3.1 Creation

During the creation stage, we provide explicit instructions and demonstration examples to guide the LLM in generating relevant tools tailored to solve the given problem. In essence, we capitalize on the LLM's capacity to follow instructions, comprehend contextual information, and engage in abstract reasoning during the initial phase of problem-solving.

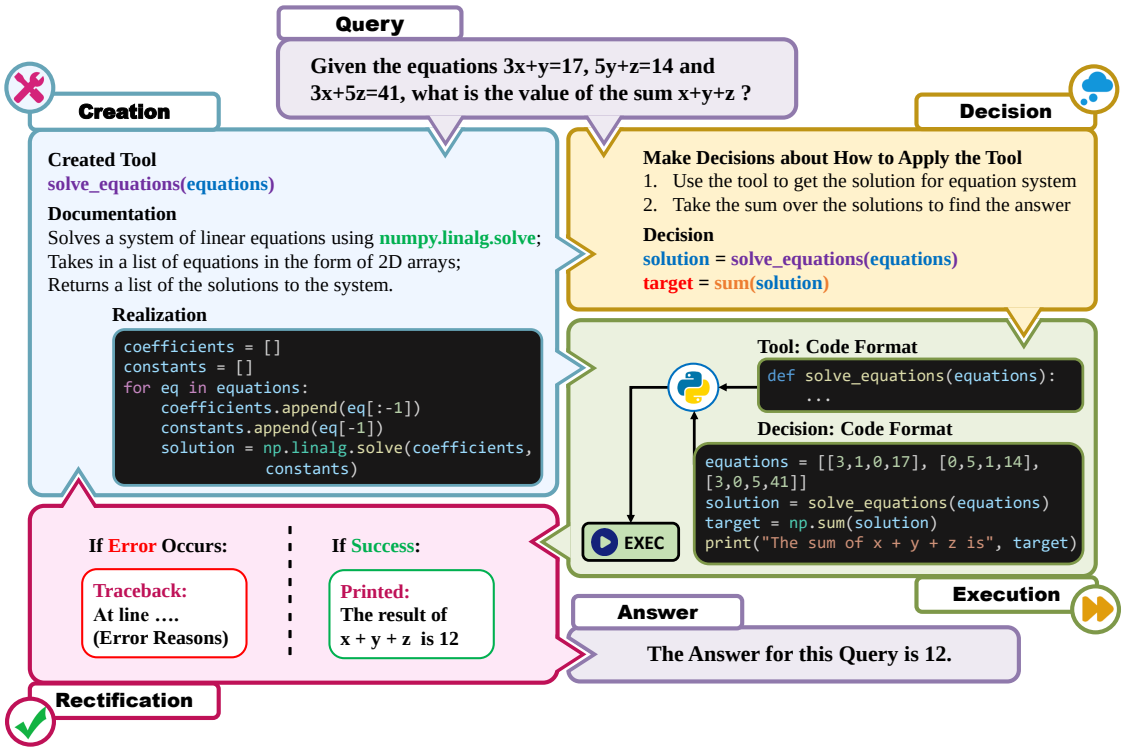

Figure 2: Overview of our CREATOR framework with four stages: Creation, Decision, Execution, and Rectification. With an LLM like ChatGPT, we successfully leverage its tool creation ability with code as the medium.

$\mathcal{S}_i$	Four Framework Stages, where $i = \{crt, dec, exc, rtf\}$
$\mathcal{Q}$	The query to be solved
$\mathcal{D}$	The documentation of a tool
$\mathcal{R}$	The realization code of a tool
$\mathcal{C}$	The decision generated by LLM
$\mathcal{E}$	The information obtained from execution
$\mathcal{A}$	The final answer extracted
k	The kth round of rectification

Table 1: The notations used in framework formalization.

Formalization. The creation stage $\mathcal{S}_{crt}$ takes the error information $\mathcal{E}^{k-1}$ from the last round of execution and the query $\mathcal{Q}$ as inputs. It produces a pair $(\mathcal{D}^k, \mathcal{R}^k)$ representing the documentation and realization of the created tool in the kth round.

$$\mathcal{S}_{crt}(\mathcal{E}^{k-1}, \mathcal{Q}) \rightarrow (\mathcal{D}^k, \mathcal{R}^k) \quad (1)$$

Documentation and Realization. The creation of a tool involves two crucial aspects: documentation and realization. Documentation provides information about the tool's utility, inputs, and outputs, while realization involves implementing the tool leveraging codes. In Figure 2, after presented the query, the LLM is instructed with demonstration examples to generate documentation about how to implement an equation solver. Next, the LLM realizes the created tool based on the documentation, with *numpy.linalg.solve* as the core.

Both documentation and code realization are essential in the process of tool creation. Documentation serves to establish the tool's relevance to a specific problem, and it is important for the LLM to refer to documentation when deciding how to utilize the tool in subsequent stages. Realization, on the other hand, is necessary for executing the tool and obtaining results.

Ability of Abstract Reasoning. The core aspect of tool creation lies in the LLM's ability to employ abstract thinking to alleviate the burden of reasoning during later stages. When LLMs create tools, they effectively use abstraction to address a particular problem type, necessitating a focus on the inherent characteristics of the problem rather than the specific numerical details.

For example, in Figure 2, the query pertains to an equation system. During the creation of the tool, the LLM disregards the numerical details and the specific expression being queried. Instead, it concentrates solely on recognizing the intrinsic nature of the problem: solving a three-variable equation system.

By having the tool readily available, LLMs can subsequently focus solely on how to utilize it, without being concerned with its implementation details. This approach significantly reduces the reasoning burden in later stages, leading to more reusable, testable tools and potentially enhanced performance.

3.2 Decision

After the tools are created, the LLM has to apply concrete reasoning to decide when and how to apply the tools, given the related information and the query.

Formalization. The decision stage $\mathcal{S}_{dec}$ takes three inputs including the query $\mathcal{Q}$, the documentation $\mathcal{D}^k$ and the realization $\mathcal{R}^k$ of the generated tool in current round k. It produces the LLM's decision $\mathcal{C}^k$ regarding all the inputs.

$$\mathcal{S}_{dec}(\mathcal{Q}, \mathcal{D}^k, \mathcal{R}^k) \rightarrow \mathcal{C}^k \tag{2}$$

Logic Chain for Decision Making. In the initial step, the LLM comprehends the tools' utility through their documentation and determines which tool(s) to use and how many times they should be applied in problem-solving. Next, the LLM takes into account the numerical value provided in the query to appropriately format the inputs for the tool Lastly, the LLM considers the relationship between the tool's outputs and the specific requirements of the problem. For instance, in the case presented in Figure 2, the solution obtained from the tool needs to be summed to derive the final answer. These three sequential steps constitute the logical chain underlying the LLM's reasoning process, with each step reliant on the clarity of the tool's documentation.

Ability of Concrete Reasoning. In contrast to the creation stage, the decision stage necessitates the LLM's meticulous attention to rules and details in order to effectively solve the problem. It capitalizes on the LLM's capacity to adhere to documentation, analyze the current context, and extract pertinent details. By separating the creation stage from the decision stage, CREATOR successfully disentangles the two phases of the LLM's abilities, facilitating a smoother elicitation of different aspects of knowledge. As demonstrated in subsequent experiments, this disentanglement contributes to improved task performance.

3.3 Execution

The execution stage takes both the information from the creation stage and the decision stage, and provides the execution results leveraging the code interpreter.

Formalization. The execution stage $\mathcal{S}_{exc}$ takes the realization of tools $\mathcal{R}^k$ from the creation stage, and the tool-calling decision $\mathcal{C}^k$ as inputs. It produces the execution results $\mathcal{E}^k$ with any printed or error information.

$$\mathcal{S}_{exc}(\mathcal{R}^k, \mathcal{C}^k) \rightarrow \mathcal{E}^k \tag{3}$$

Concatenation of Inputs. To ensure successful code execution, the created tools and the LLM's decision are standardized into a code format and concatenated. The tool is encapsulated within a function (or method), which the **decision** step can call upon to solve the problem.

Capture of Results. During the last stages, we have also instructed the LLM to print any information that may be useful for its further reference. Therefore, when the program is executed, we capture any outputs printed or errors encountered. These pieces of information serve as the inputs for subsequent stages to determine whether an answer can be obtained or if the created tool and the decision require rectification.

3.4 Rectification

During the rectification stage, CREATOR has two different options to choose from based on the information passed into it. It can either rectify the tool and decision based on error tracebacks or provide the final answer to the given problem.

Formalization. The rectification stage $\mathcal{S}_{rtf}$ takes execution results $\mathcal{E}^k$ and extracts the answer if the program runs with success. Otherwise, it passes on $\mathcal{E}^k$ to the next (k+1th) round of rectification.

$$\mathcal{S}_{rtf}(\mathcal{E}^k) \rightarrow \mathcal{A} \texttt{ if } success \tag{4}$$

Rectification of the Tool and Decision. If an error occurs, the LLM is prompted by instructions to examine the error and re-generate the tools and decisions based on the captured error tracebacks, initiating a new iteration. These tracebacks provide crucial information for the LLM to identify the error's location and causes. Armed with this guidance, the LLM can swiftly recover from previous mistakes, adjust its reasoning process, and attempt to solve the problem once again.

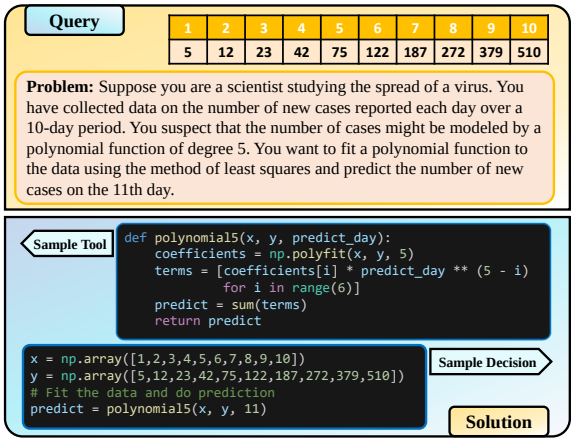

Figure 3: An example query and its solution provided in the Creation Challenge dataset.

Answer Extraction. When the execution stage completes successfully, there is no need for further rectification. In such cases, an answer is extracted from the model's printed output. To quantify the accuracy, numerical values are extracted and compared against the correct answer. Alternatively, if no numerical values are involved, the printed texts can also be considered the final output of the framework.

Subsequent experiments will demonstrate how the inclusion of the rectification stage significantly enhances the performance of CREATOR. The success of rectification also highlights the LLM's ability to recognize misconceptions and make self-corrections.

4 Experiments

To evaluate the effectiveness of CREATOR, we conduct experiments on two established benchmarks: MATH (Hendrycks et al.) and TabMWP (Lu et al., 2022a). Though CREATOR benefits various tasks requiring tool creation, we specifically consider these two datasets as representative examples to showcase our framework's strengths. Additionally, we perform experiments on our newly introduced dataset, Creation Challenge, comprising 2K diverse questions that are inadequate to solve using existing tools or code packages. This enables us to further demonstrate the necessity and advantages of the LLM's tool creation ability.

4.1 Experimental Setup

Base Model. We select ChatGPT (gpt-3.5-turbo) as the base model for CREATOR due to its exceptional capabilities in code generation, decision making, and logical reasoning. To ensure fairness, we also employ ChatGPT as the base model for all other settings including the chain-of-thought (CoT), program-of-thought (PoT), and tool-use baselines. The maximum generation length for all experiments is set to 512, and a temperature of 0.3 is chosen to encourage deterministic generations while maintaining a certain degree of diversity, particularly during the creation of tools.

Dataset. For both MATH and TabMWP datasets, we test CREATOR on all the questions in the test split with numerical value answers (e.g. integer or decimal values). This is due to other answer forms including expressions and fractions are hard to test automatically, and the outputs of programs are prone to give decimal values, which further makes the automatic evaluation complicated. Nevertheless, the test set we apply is able to cover around 80% of the questions and maintain high diversity, so we regard our results as representative.

The TabMWP dataset encompasses a wide range of table information and problems of varying difficulty levels, spanning from grade one to grade eight. In the case of the MATH dataset, it encompasses seven domains of math competition problems, including algebra, counting and probability, geometry, intermediate algebra, number theory, pre-algebra, and pre-calculus. Each domain is tested separately, and the weighted average score is computed as the final metric. For the Creation Challenge dataset, we evaluate CREATOR on all 2K data instances and explore the impact of tool creation hints on the LLM's performance.

Baselines. We compare CREATOR against three types of baselines to demonstrate its effectiveness:

Standard LLM w/ and w/o CoT: For Standard LLM with CoT setting, the LLM employs sequential reasoning to solve problems step by step. We also include the baseline Standard LLM without CoT, where the LLM directly generates the answer without employing the reasoning process.

PoT: The LLM leverages a program to reason through the problem step by step. PoT can be combined with CoT to provide the answer. To ensure a fair comparison specifically focused on the tool creation stage, we also incorporate the rectification module into PoT, making it a stronger baseline.

Tool Use: The LLM utilizes the WolframAlpha API as a tool for problem-solving. The Wolfra-

Method	Setting	Algebra	Counting & Probability	Geometry	Itmd. Algebra	Number Theory	Pre-Algebra	Pre-Calculus	Average (weighted)
Standard	w/o CoT	25.7	25.8	22.4	13.9	18.5	40.9	21.8	25.3
	w/ CoT	50.9	36.1	24.5	17.5	23.2	58.6	16.7	37.9
PoT (w/o Rectify)	w/o CoT	58.2	48.5	35.4	25.8	53.1	66.8	25.0	49.8
	w/ CoT	54.0	47.8	32.5	22.3	48.9	64.5	19.9	46.5
PoT (w/ Rectify)	w/o CoT	63.8	51.9	35.9	28.6	59.2	70.0	28.2	53.9
	w/ CoT	61.4	48.8	34.6	23.7	54.5	67.6	34.6	51.2
Tool Use	w/o CoT	47.3	35.1	27.0	20.5	30.8	56.8	31.4	39.0
	w/ CoT	55.3	37.8	28.7	20.5	34.8	61.8	26.9	43.0
Tool Create (whole)	w/o Demo	58.0	53.3	34.2	21.8	55.7	63.4	33.3	49.6
	w/o CoT	64.1	55.7	35.9	**42.7**	61.6	69.0	**37.2**	57.2
	w/ CoT	62.7	50.9	33.8	31.4	61.4	68.7	31.4	54.0
CREATOR (ours)	w/o Demo	66.6	53.6	33.8	29.4	59.8	68.7	34.6	54.9
	w/o CoT	**71.5**	55.3	**41.4**	41.9	60.4	**71.7**	35.3	**59.7**
	w/ CoT	63.1	**58.1**	34.6	35.0	**61.8**	69.7	32.1	55.7

Table 2: The accuracy (%) on the test set of MATH dataset leveraging ChatGPT. We report the results of standard QA generation, CoT, PoT, Tool Use, and our CREATOR framework under different settings.

mAlpha API is a general-purpose tool specialized in math calculations. Since all the testing data requires numerical calculations to a certain extent, enabling the LLM to leverage WolframAlpha serves as a fair baseline for LLM's external tool use.

In addition to these baselines, we introduce another approach by combining the creation stage and decision stage of CREATOR (referred to as **Tool Create - whole**). In this case, we do not disentangle the abstract tool creation and the concrete decision making. This approach can be seen as a special baseline for ablation studies.

4.2 Creation Challenge

Existing benchmarks are not originally designed to evaluate tool creation, and therefore can not fully showcase the necessity and advantages brought by the LLM's tool creation ability. Therefore, we introduce Creation Challenge to test the LLM's problem-solving skills under new scenarios, without existing tools or code packages that can be directly applied to solve the new problem. Note that although the problems could not be directly solved by existing tools, they could still be leveraged as part of the new tool created if necessary.

Dataset Construction. We begin by constructing a seed dataset that involves novel settings and unconventional reasoning processes. Subsequently, we utilize the Text-Davinci-003 model to expand the dataset in an iterative manner. By random sampling from the seed data, we encourage the variety and novelty in the problems and their reasonings. Figure 3 illustrates a query and its corresponding solution. Each data entry comprises the problem statement, a standard created tool that can be utilized (including utility, input, output, and realization), a tool-calling decision, and a final answer.

Evaluation The components of the standard created tool and the tool-calling decision can serve as valuable hints for the LLM's tool creation and problem-solving. In our experiments, we present the problem to the LLM and assess its tool creation ability with varying levels of hint utilization. The framework's performance is evaluated based on the final answer. We encourage future research to explore the dataset's potential through more flexible usage, further unveiling and testing the capabilities of LLMs.

4.3 Experimental Results

MATH. The experimental results on the MATH dataset are presented in Table 2. CREATOR achieves an average accuracy of 59.7%, surpassing the best performance of the CoT, PoT, and Tool Use methods by 21.8%, 5.8%, and 16.7% respectively. Disentangling the creation stage and decision stage also leads to generally higher accuracy, with an average performance improvement of 2.5%. We analyze the reasons for the improvement in Section 4.4.

TabMWP. The experimental results on the TabMWP dataset are presented in Table 3. For each setting, we measure both the accuracy and the successful execution rate. The successful execution

Method	Setting	Accuracy	Successful Execution
Standard	w/o CoT	68.2	99.1
	w/ CoT	75.2	99.3
PoT (w/o Rectify)	w/o CoT	80.6	98.5
	w/ CoT	80.0	91.2
PoT (w Rectify)	w/o CoT	81.2	99.7
	w/ CoT	87.3	100
Tool Use	w/o CoT	77.6	100
	w/ CoT	79.6	100
Tool Create (whole)	w/o CoT	91.6	100
	w/ CoT	93.5	99.9
CREATOR (ours)	w/o CoT	90.5	99.7
	w/ CoT	**94.7**	100

Table 3: The accuracy (%) on the test set of TabMWP dataset leveraging ChatGPT. We report the results of standard QA generation, CoT, PoT, Tool Use, and our CREATOR framework under different settings.

Method	Setting	Accuracy	Successful Execution
Standard	w/o CoT	27.9	94.9
	w/ CoT	32.7	99.1
PoT (w/o Rectify)	w/o CoT	59.2	93.5
	w/ CoT	60.7	95.7
PoT (w Rectify)	w/o CoT	61.1	98.3
	w/ CoT	62.0	98.9
Tool Create (whole, w/o CoT)	no hint	64.5	99.2
	utility hint	65.8	99.3
	all hint	75.3	99.5
CREATOR (ours, w/o CoT)	no hint	63.8	98.7
	utility hint	67.2	99.1
	all hint	**75.7**	99.5

Table 4: The accuracy (%) on the Creation Challenge test set leveraging ChatGPT. We report the results of standard QA generation, CoT, PoT, and our CREATOR framework with different levels of hints.

rate indicates whether the LLM provides a final answer within the word limit or the program yields a valid result within the rectification limit. CREATOR achieves an overall accuracy of 94.7%, outperforming the best settings of CoT, PoT, and Tool Use by 19.5%, 7.4%, and 15.1% respectively. Compared to the standard methods, our framework also achieves a higher successful execution rate. These results highlight the effectiveness of our framework in both text and tabular contexts.

Creation Challenge. The experimental results on the Creation Challenge dataset are summarized in Table 4. In addition to evaluating the LLM's performance under the CoT and PoT settings, we also assess its performance to create tools with varying levels of hints. The "*no hint*" setting represents the standard CREATOR configuration, while the "*utility hint*" provides hints about the utility of the tool, and the "*all hint*" offers additional hints about the possible inputs and outputs of the tool that the LLM may create. Our results demonstrate that even without any hints, CREATOR still outperforms PoT by 1.6%. As the number of hints provided to the LLM increases, its performance gradually improves, achieving an increase of up to 18.7% and reaching a high accuracy of 75.5%. We provide further analysis on the role of hints in Section 4.4.

4.4 Results Analysis

CoT Incompatible with Codes. Table 2 reveals a consistent decrease in the performance of the LLM on MATH problems when employing the CoT setting, regardless of whether PoT or CREATOR are utilized. In contrast, the opposite trend is observed for TabMWP. We attribute this disparity to the **inherent incompatibility between natural language reasoning and program-based reasoning when tackling challenging problems**. MATH competition problems, known for their intricate calculations, introduce higher complexity and diversify the possible reasoning paths. This often leads to conflicts between natural language and programming approaches when solving the same problem. When CoT is employed, the LLM tends to incorporate programs that follow natural language reasoning, which potentially compromises the coherence and advantages of programming itself, ultimately resulting in reduced performance. On the other hand, TabMWP also evaluates the model's capacity to extract information from tables, so it involves simpler calculations and a more constrained and straightforward reasoning path. Consequently, this promotes greater consistency between natural language and programming reasoning. Therefore, the application of CoT to such problems enhances performance.

To illustrate this phenomenon, we present three cases in Figure 4. In the two MATH examples, CoT generates complex tool creation plans that misguide the implementation of tools. Interestingly, without CoT, the adoption of brute-force algorithms and straightforward calculations actually yields higher accuracy. Conversely, for the relatively simpler TabMWP problem, the reasoning path is largely predetermined, making CoT ben-

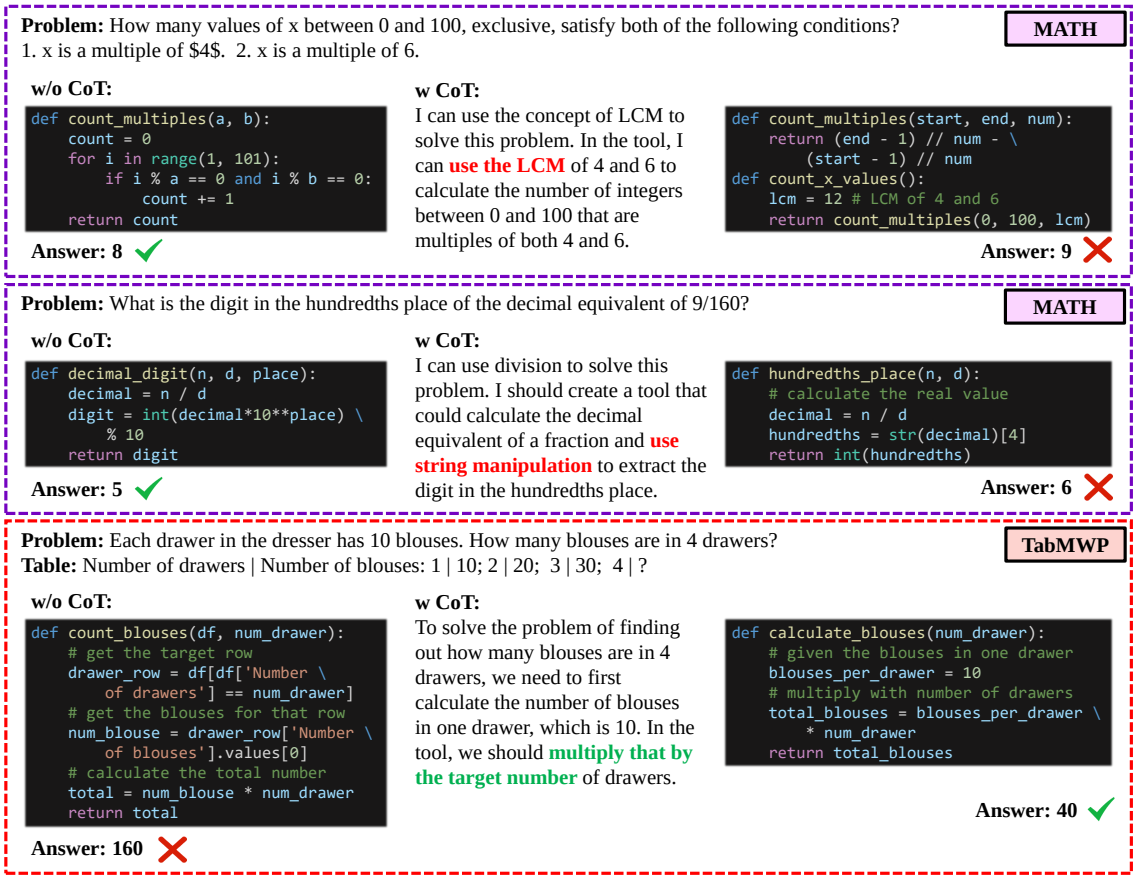

Figure 4: We present three examples from CREATOR and compare the answer given with or without CoT. Challenging problems in MATH cause conflicts between language and program reasonings, while for simpler problems in TabMWP, language and program reasonings complement each other.

eficial for tool creation by avoiding unnecessary complexities in problem-solving.

Performance Relative to Difficulty. Figure 5 illustrates the performance of the LLM in relation to the difficulty levels of the problems. We compare the results of standard methods with and without the application of CoT, as well as the best settings of PoT, Tool Use, and CREATOR. In the case of MATH, the model's performance displays a linear decline as the difficulty level of the questions increases. For TabMWP, the trend was not as pronounced as in MATH, but there is a noticeable drop in accuracy between problems of grades 2 and 3 for both of the standard methods and Tool Use baselines. CREATOR outperforms all the baselines for both tasks and achieves higher accuracy, particularly for difficult problems. This finding provides compelling evidence that **CREATOR exhibits greater resilience to challenges and more balance when dealing with problems of varying difficulty levels**.

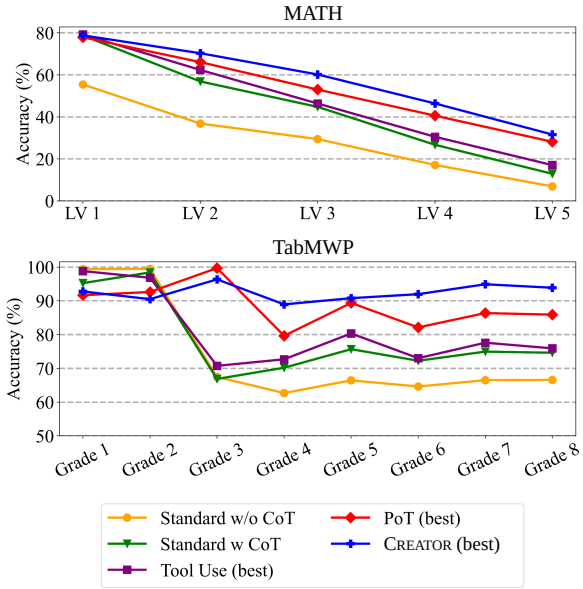

Figure 5: Comparison of the accuracy of LLM on baselines and CREATOR, relative to problem difficulty.

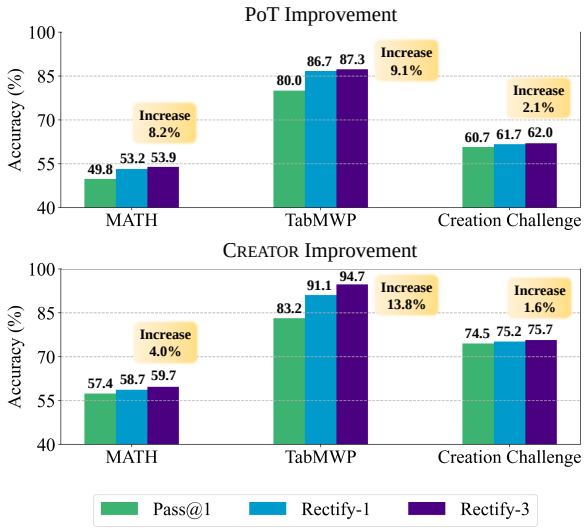

Figure 6: The improvement of the LLM's performance when the rectification stage is applied for both PoT and CREATOR.

Role of Rectification. The rectification stage is designed to enable the LLM to examine and rectify its own mistakes in case of execution errors, utilizing its intrinsic capacity to detect errors and perform self-correction based on evidence. Figure 6 demonstrates the improvement in the LLM's performance achieved through the application of the rectification stage for both PoT and CREATOR. In this figure, "*Pass@1*" indicates the LLM provides the correct answer directly in the first round without any rectification, while "*Rectify-N*" denotes the LLM retrieves the correct answer within N rounds of rectification. Our results show that **rectification can increase the accuracy of the LLM by approximately 10% of the original value**, which proves the necessity and rationality of establishing this stage.

Influential Factors of Tool Creation. Table 2, Table 3, and Table 4 highlight two crucial factors that affect the LLM's performance. (1) **Separation of Creation and Decision**: the separation of these two stages inherently represents the disentanglement of the LLM's abstract thinking and concrete thinking abilities during problem-solving. This disentanglement can isolate the LLM's distinct reasoning capacities, leading to an improvement in its performance. Our experiments show that the performance of the LLM increases by 4.4%, 1.3%, and 0.5% for MATH, TabMWP, and Creation Challenge, respectively. (2) **Availability of Hints** In practical scenarios, it is often necessary to provide guidance to LLMs during tool creation in order to better harness their behavior. Therefore, we investigate the role and impacts of hints, and demonstrate through the results of the Creation Challenge that providing more detailed hints can significantly improve the LLM's performance. This is because the utility, input, and output hints can enable the LLM to implement the desired tool much easier with success. Additionally, these hints can eliminate the LLM's uncertainty and reduce any misdirections that may present in CoT or the tool's documentation.

Set of Queries, Count	100
Data Pieces, Count	300
Tool Create Normal, Acc.	63.0%
Tool Create with Transfer, Acc.	78.3%
Increase of Acc.	**26.3%**
Sets Worse with Transfer	2 / 100
Sets Better with Transfer	**39 / 100**

Table 5: The statistical results of tool transfer experiment. The accuracy of applying tool transfer improves up to 26.3%.

5 Further Discussions

In this section, we will further show the advantages brought by the LLM's tool creation ability, and use case studies to demonstrate different aspects of this ability, which enables them to tackle challenges with more flexibility and less reasoning burden.

5.1 Facilitation of Knowledge Transfer

One of the main purposes of tool creation lies in its reusability. The implementation of tools represents the abstraction of knowledge concepts, so the creation of one tool may help solve problems of various scenarios that have the same core concept. In other words, the LLM's tool creation ability facilitates the transfer of core abstract knowledge among scenarios, and helps raise the performance of similar problems. For instance, suppose an LLM creates a tool for extracting keywords from a given text to solve a query about sentiment analysis. This tool can actually be leveraged in various other scenarios such as document categorization, topic modeling, etc. By utilizing the knowledge and logic embedded in the tool, the LLM can transfer its understanding of keyword extraction to different scenarios and apply it to solve similar problems efficiently with higher performance.

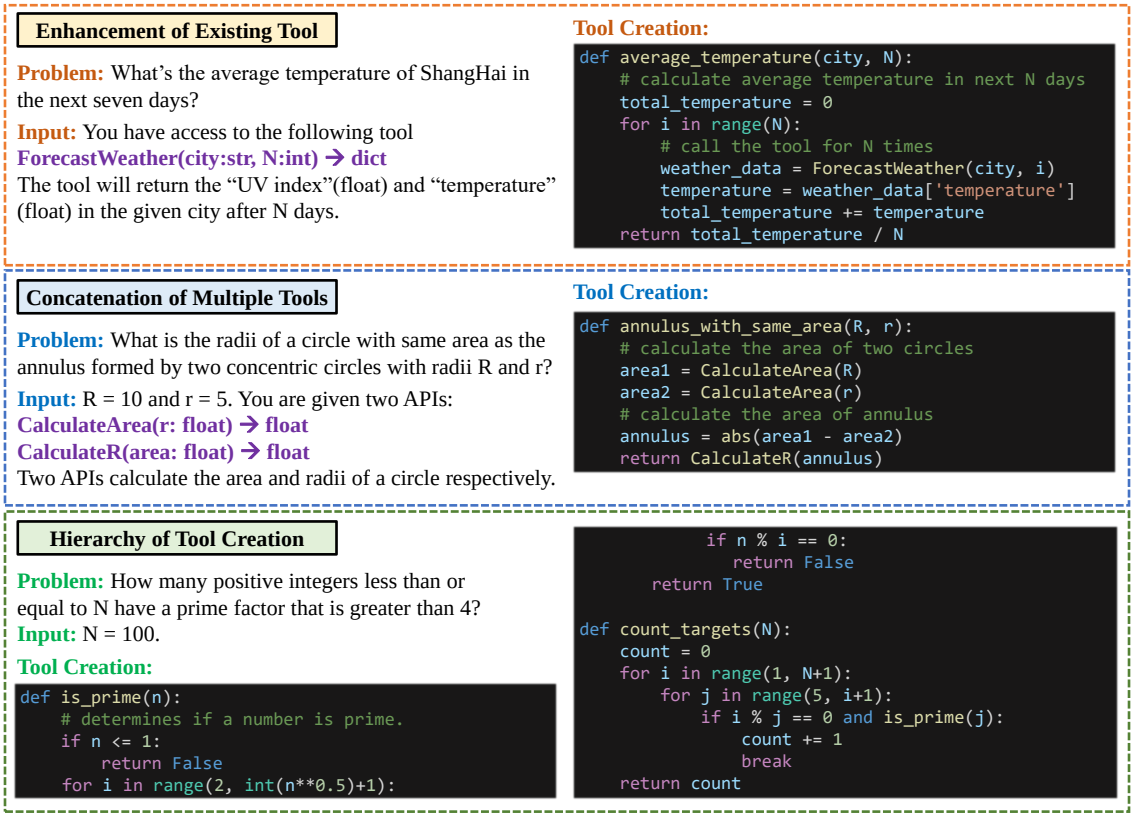

Figure 7: We show three simple cases to illustrate the idea of the LLM's tool creation from different aspects.

Data Construction. To validate our hypothesis, we construct a small set of questions with 300 data points to test the knowledge transfer ability of the LLM's tool creation. We divide 300 data points into 100 sets, where all three queries in one set share the same core knowledge concept. Each set of queries also contains three corresponding answers, one standard tool that could be applied in all three scenarios to solve the problem, and three decisions about how to use the tool respectively. Similar to Creation Challenge, we manually write the seed data, which includes five sets of queries used as examples to show the format of each data point, sample demonstration examples from these seeds, and leverage the Text-Davinci-003 to create more data iteratively.

Setup. Similar to previous experiments, we use ChatGPT as the base LLM and keep all the detailed settings the same as before. We first test all the problems under the normal CREATOR framework respectively. Then, we test if the correct tool created in one of the problem scenarios could be applied to the other two, and again test the LLM's performance. For simplicity and fairness, we do not include the rectification stage in this experiment and only compute the accuracy based on "*Pass@1*".

Results Analysis. The statistical results are presented in Table 5. Our experiment consists of 100 sets of queries with 300 data points in total. Through the application of transferred tools, we were able to demonstrate that accuracy can be improved from 63.0% to 78.3% for 300 data points. This highlights the potential benefits of using transferred tools in improving the LLM's performance.

Further analysis shows that the LLM's performance is positively influenced by the transfer of tools it creates. Specifically, 39 sets of queries have higher accuracy as a result of tool transfer. This finding suggests the tool creation ability of the LLM can facilitate knowledge transfer, leading to better performance on clusters of problems that share similar core concepts.

5.2 Different Levels of LLM's Tool Creation

As a preliminary study on LLMs' tool creation, in this section, we propose to categorize tool creation into 3 different levels, so as to provide guidance and reference to future development. These levels demonstrate how the LLM can leverage existing tools and APIs to create new tools that cater to

diverse needs.

Enhancement of Existing Tool. At the first level, LLMs demonstrate their capability to enhance existing tools. By encapsulating an existing tool or API, LLMs can re-purpose it to serve different needs. The first case in Figure 7 exemplifies this level, where an existing weather query API is wrapped into a new tool that calculates the average temperature. This level highlights LLMs' ability to adapt and enhance tools to meet various requirements.

Concatenation of Multiple Tools. The second level involves the concatenation of multiple tools. LLMs create new tools by organizing multiple APIs into a pipeline, enabling them to fulfill specific purposes. The second case in Figure 7 serves as an example, where the LLM creates a tool that calls two existing APIs three times to solve the given problem. This case also demonstrates the flexibility of the tool created by the LLM, as it can be generalized for different sets of input parameters R and r.

Hierarchy of Tool Creation. The third level of our framework involves organizing tools hierarchically during the tool creation process. This hierarchy establishes clear caller-callee relationships among the tools. This hierarchical structure is demonstrated in detail in the third case presented in Figure 7. In this case, the first tool serves as the callee, while the second tool acts as the primary tool responsible for solving the problem. By creating tools hierarchically, the LLM can reduce repetitive reasoning and enhance performance by applying clearer logic. This hierarchical organization of tools contributes to improved efficiency and effectiveness in problem-solving.

Overall, the presented case studies provide valuable insights into the tool creation abilities of LLMs. However, it is important to acknowledge that these studies offer only a glimpse into the vast potential of LLMs in this domain. We encourage future research to explore and harness the full extent of LLMs' tool creation capabilities, further pushing the boundaries of what can be achieved.

6 Conclusions and Future Work

In conclusion, we propose the concept of tool creation and empirically devise a tool creation framework, CREATOR, that successfully harnesses the capabilities of LLMs as tool creators to address various problem-solving scenarios. By disentangling the abstract and concrete reasoning abilities of LLMs, CREATOR enables clearer logic and enhances overall performance. Through comprehensive evaluations on established benchmarks as well as a newly developed Creation Challenge test set, we have demonstrated the superiority and indispensability of CREATOR compared to existing approaches including CoT, PoT, and tool-using methods.

This study serves as an initial exploration into the domain of tool creation, providing a preliminary understanding of its potential. To advance this field, future research can delve deeper into the tool creation abilities of LLMs by conducting experiments encompassing a wider range of tasks and complex scenarios. While our study has focused on unraveling the disentanglement of reasonings behind LLMs' tool creation, there are other essential aspects of their tool creation capabilities that warrant further investigation. Additionally, there is a need to investigate methods for enhancing the efficiency of the tool creation process while aligning it more closely with users' intentions.

By embarking on these future research directions, we anticipate that our study will establish a solid foundation, inspiring the development of more sophisticated AI systems that harness the tool creation ability of LLMs. Moreover, this work represents a significant step forward in pushing the boundaries of LLM capabilities and unlocking their full potential.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large

language models trained on code. *arXiv preprint arXiv:2107.03374*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Tanmay Gupta and Aniruddha Kembhavi. 2022. Visual programming: Compositional visual reasoning without training. *arXiv preprint arXiv:2211.11559*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *Sort*, 2(4):0–6.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*.

Mojtaba Komeili, Kurt Shuster, and Jason Weston. 2022. Internet-augmented dialogue generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8460–8478.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2023. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2022a. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. *arXiv preprint arXiv:2209.14610*.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2022b. A survey of deep learning for mathematical reasoning. *arXiv preprint arXiv:2212.10535*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.

OpenAI. 2022. Chatgpt.

OpenAI. 2023. Gpt-4 technical report.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023. Tool learning with foundation models.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Kurt Shuster, Jing Xu, Mojtaba Komeili, Da Ju, Eric Michael Smith, Stephen Roller, Megan Ung, Moya Chen, Kushal Arora, Joshua Lane, et al. 2022. Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage. *arXiv preprint arXiv:2208.03188*.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5085–5109.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed H Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.