Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models

Pan Lu[1], **Baolin Peng**[2], **Hao Cheng**[2], **Michel Galley**[2]
Kai-Wei Chang[1], **Ying Nian Wu**[1], **Song-Chun Zhu**[1], **Jianfeng Gao**[2]
[1]University of California, Los Angeles [2]Microsoft Research, Redmond
lupantech@gmail.com {bapeng, chehao, mgalley, jfgao}@microsoft.com

https://chameleon-llm.github.io

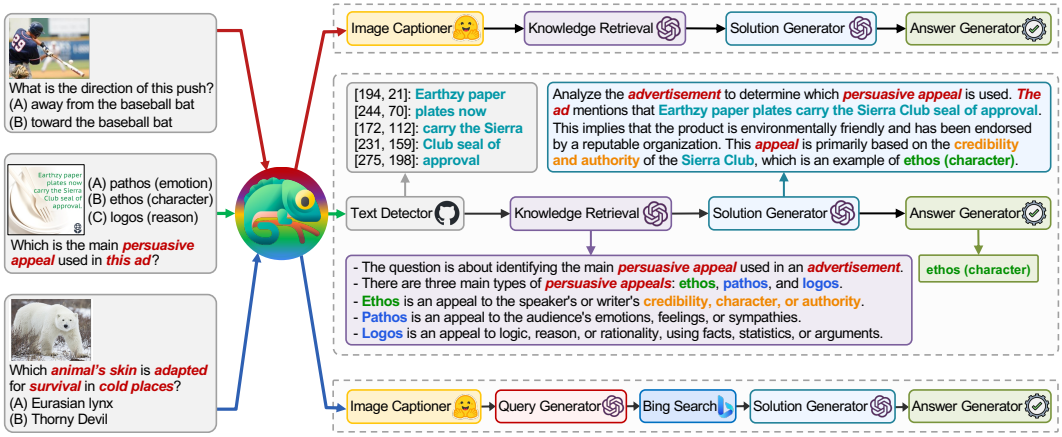


Figure 1: Examples from our Chameleon with GPT-4 on ScienceQA [27], a multi-modal question answering benchmark in scientific domains. Chameleon is adaptive to different queries by synthesizing programs to compose various tools and executing them sequentially to get final answers.

Abstract

Large language models (LLMs) have achieved remarkable progress in various natural language processing tasks with emergent abilities. However, they face inherent limitations, such as an inability to access up-to-date information, utilize external tools, or perform precise mathematical reasoning. In this paper, we introduce Chameleon, a *plug-and-play compositional* reasoning framework that augments LLMs to help address these challenges. Chameleon synthesizes programs to compose various tools, including LLM models, off-the-shelf vision models, web search engines, Python functions, and rule-based modules tailored to user interests. Built on top of an LLM as a natural language planner, Chameleon infers the appropriate sequence of tools to compose and execute in order to generate a final response. We showcase the adaptability and effectiveness of Chameleon on two tasks: ScienceQA and TabMWP. Notably, Chameleon with GPT-4 achieves an 86.54% accuracy on ScienceQA, significantly improving upon the best published few-shot model by 11.37%; using GPT-4 as the underlying LLM, Chameleon achieves a 17.8% increase over the state-of-the-art model, leading to a 98.78% overall accuracy on TabMWP. Further studies suggest that using GPT-4 as a planner exhibits more consistent and rational tool selection and is able to infer potential constraints given the instructions, compared to other LLMs like ChatGPT.

This title draws inspiration from the *chameleon*'s ability to adapt and blend into its surroundings, which parallels the adaptability and versatility of large language models in compositional reasoning tasks with external tools.

1 Introduction

Remarkable progress has been observed in recent large language models (LLMs) for various natural language processing tasks, with prominent examples such as GPT-3 [4], PaLM [7], LLaMA [56], ChatGPT [35], and the more recently proposed GPT-4 [36]. LLMs have demonstrated emergent abilities, including in-context learning, mathematical reasoning, and commonsense reasoning [48]. These models are capable of solving diverse tasks in zero-shot settings [20] or with the aid of a few examples [49], and they show great potential in planning and decision-making akin to human beings [14]. However, LLMs have inherent limitations, such as an inability to access up-to-date information [21], utilize external tools [43], or perform precise mathematical reasoning [38, 30].

To address these limitations of LLMs, an active research direction involves augmenting language models with access to external tools and resources, as well as exploring the integration of external tools and plug-and-play modular approaches. For example, aided by web search engines and external knowledge resources, LLMs are able to access real-time information and leverage domain-specific knowledge [34]. To enhance mathematical reasoning abilities, recent work uses LLMs [5] to generate complex programs to exploit powerful computational resources, and execute logical reasoning tasks more effectively [47, 9, 6, 33, 15, 37]. Another line of recent work, such as ViperGPT [45], Visual ChatGPT [51], VisProg [11], and HuggingGPT [44] incorporates a collection of foundation computer vision models to equip LLMs with the abilities to perform visual reasoning tasks.

Despite significant progress, current tool-augmented LLMs still face critical challenges when addressing real-world queries. Most existing approaches are either limited to a small number of tools [33, 6, 47, 15, 37, 43] or rely on domain-specific tools [34, 52, 11, 51, 45], and thus are not easy to generalize to diverse queries. Consider the example in Figure 1: "*Which is the main persuasive appeal used in this ad?*". To answer this question, one needs to: 1) infer that there is an ad image containing text context and call a text decoder to understand the semantics; 2) retrieve background knowledge to clarify the definition of "*persuasive appeal*" and the differences among various types; 3) generate a solution based on the clues from the input query and intermediate results from previous steps; and 4) ultimately produce the answer in a task-specific format. On the other hand, when answering "*Which animal's skin is adapted for survival in cold places*", one might need to call other modules, such as an image captioner to decipher image information and a web search engine to retrieve domain knowledge to understand scientific terminologies.

To address these challenges, we introduce **Chameleon**, a *plug-and-play compositional* reasoning framework that harnesses the capabilities of large language models. **Chameleon** is capable of synthesizing programs to compose various tools to tackle a broad range of queries. While it does not require any training, **Chameleon** uses the in-context learning capabilities of LLMs to generate these programs. In contrast with existing approaches [43, 34, 52, 11, 51, 45], it employs diverse types of tools, including LLMs, off-the-shelf computer vision models, web search engines, Python functions, and rule-based modules tailored to specifically customized tasks. **Chameleon** builds on top of an LLM as a natural language planner. Prompted by descriptions of each tool and examples of tool usage, the planner can infer an appropriate sequence of tools to compose and execute in order to generate the final response for a user query. Unlike previous work that generates domain-specific programs [34, 45, 11], **Chameleon** generates natural-language-like programs, which are less error-prone, easy to debug, user-friendly for those with limited programming experience, and efficiently extendable to using new modules. During the execution of each module in the program, the module processes the query and caches context, returns a result determined by the module itself, and updates the query and cached context for subsequent module execution. Composing modules as a sequential program, the execution of subsequent modules can leverage the prior cached context and updated queries.

We showcase the adaptability and effectiveness of **Chameleon** on two tasks: ScienceQA [27] and TabMWP [28]. ScienceQA is a multi-modal question answering benchmark spanning multiple context formats and various scientific topics, while TabMWP is a mathematical benchmark involving diverse tabular contexts. These two benchmarks serve as a good testbed to evaluate **Chameleon**'s ability to coordinate diverse tools across different types and domains. Notably, **Chameleon** with GPT-4 achieves an 86.54% accuracy on ScienceQA, significantly improving upon the best published few-shot model by 11.37%. On TabMWP, using GPT-4 as the underlying LLM, **Chameleon** achieves an improvement of 7.97% over CoT GPT-4 and a 17.8% increase over the state-of-the-art model, leading to a 98.78% overall accuracy on TabMWP. Further studies suggest that using GPT-4 as

a planner exhibits more consistent and rational tool selection and is able to infer potential constraints given the instructions, compared to other LLMs like ChatGPT.

In summary, our contributions are as follows: (1) We develop a plug-and-play compositional reasoning framework, **Chameleon**, that effectively leverages large language models to address their inherent limitations and tackle a broad range of reasoning tasks. (2) We successfully integrate various tools, including LLMs, off-the-shelf vision models, web search engines, Python functions, and rule-based modules, to build a versatile and adaptable AI system to answer real-world queries. (3)) We showcase the framework's adaptability and effectiveness on two diverse benchmarks, ScienceQA and TabMWP, significantly lifting the state of the art.

2 Related Work

2.1 Large Language Models

In recent years, the development of large language models (LLMs) has made tremendous progress, as evidenced by models such as BLOOM [42], PaLM [7], Flan-T5 [8], LLaMA [46], GPT-3 [4], ChatGPT [35], and GPT-4 [36]. These LLMs exhibit emergent abilities, including in-context learning, mathematical reasoning, and commonsense reasoning. LLMs like GPT-3 and GPT-4 are capable of solving various tasks in zero-shot settings or with the aid of a few examples. The chain-of-thought (CoT) approach [49] instructs LLMs to decompose complex problems by generating intermediate reasoning steps before arriving at the final answer. Another active research area is to incorporate external verification [39, 32] and human feedback [35] to allow LLMs to perform more reliably and align better with human preferences. In addition, a recent line of work focuses on instruction learning, either via generating high-quality instructions data [40] or by boosting the language models with instructions, such as Flan-T5 [8], LLaMA [46], LLaMA-Adapter [56], and LLaVA [24]. Our proposed **Chameleon** builds on these lines of work, introducing a general approach that enables cross-task generalization, leverages in-context learning from demonstrations, and integrates various capabilities of external tools to solve complex reasoning problems.

2.2 Compositional Reasoning

Neural modular and compositional approaches have been explored to automatically perform desired sub-task decomposition, enhancing interpretability and adaptability across various reasoning tasks. Early work [2, 3] posits that complex reasoning tasks are fundamentally compositional and proposes neural module networks (NMN) to decompose them into subtasks. However, these methods rely on brittle off-the-shelf parsers and are limited by module configurations. Some later work [16, 13, 12] takes a step further by predicting instance-specific network layouts in an end-to-end manner, without relying on parsers, using reinforcement learning [50] and weak supervised learning. In visual reasoning, models comprising a program generator and an execution engine have been proposed to combine deep representation learning and symbolic program execution [16, 53]. In the domain of mathematical reasoning, an interpretable problem solver has been developed to incorporate theorem knowledge as conditional rules and perform symbolic reasoning step by step [26].

Our work takes inspiration from neural module networks, yet it offers several distinct advantages. First, **Chameleon** does not require expensive supervision in the form of task-specific programs. Instead, it generates sequential programs, consisting of modules, that are easy to generalize to various domains and tasks, allowing for the extension to new modules in a plug-and-play manner. Second, **Chameleon** does not require any training, but uses the in-context learning capabilities of LLMs to generate programs prompted by natural language instruction and demonstrations.

2.3 Tool-Augmented Language Models

Despite impressive performance of LLMs, they suffer from inherent limitations, such as the inability to access up-to-date information [21], utilize external tools [43], or perform precise mathematical reasoning [38, 30]. To address these shortcomings, there has been a growing interest in exploring the use of external tools and modular approaches to augment LLMs. These augmented LLMs can access real-time information aided by web search engines [34] and leverage domain-specific knowledge from external resources [54]. Some work leverages the Python interpreter to generate complex programs to employ powerful computational resources, and execute logical reasoning tasks

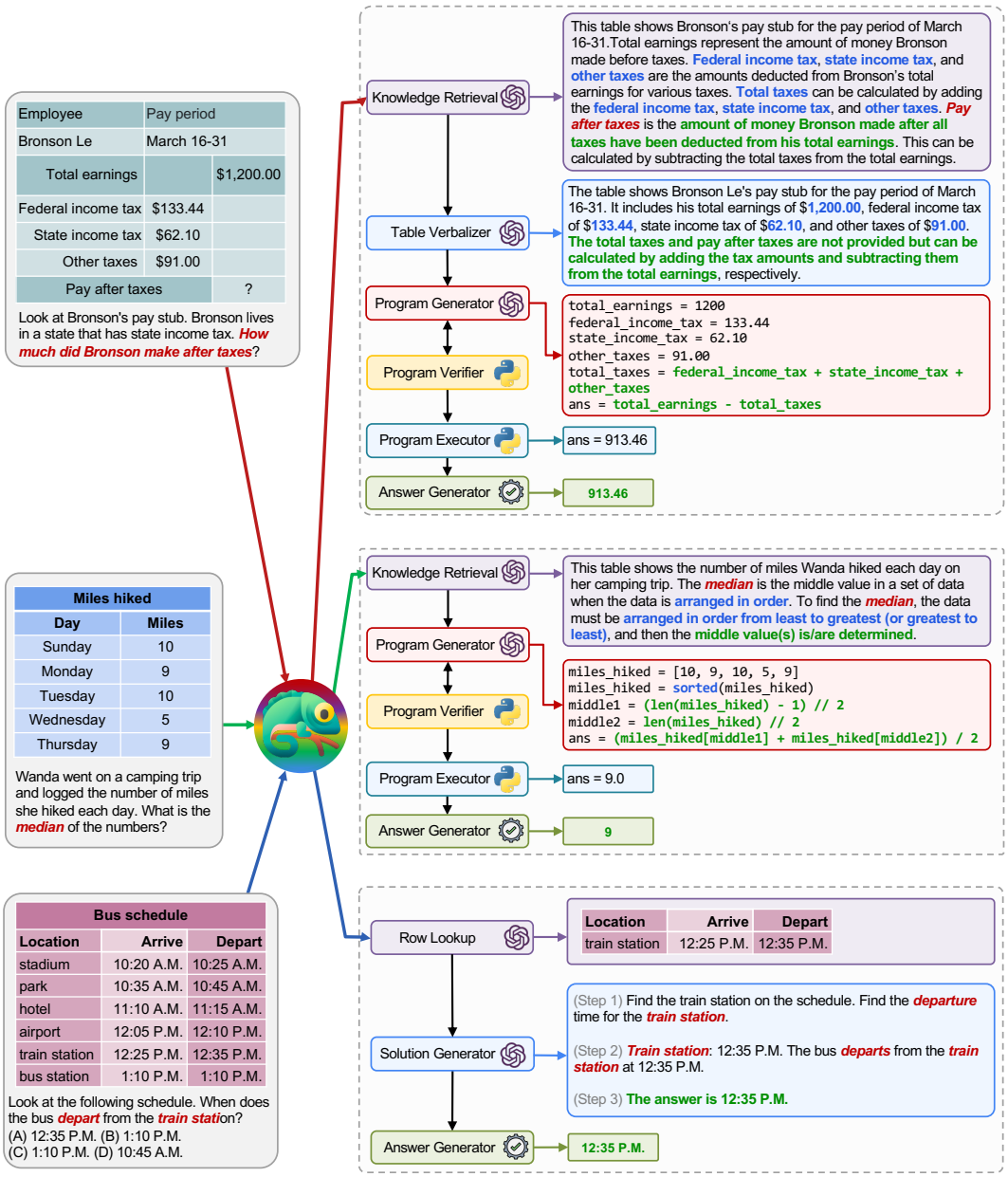


Figure 2: Three examples from our **Chameleon** with GPT-4 on TabMWP [28], a mathematical reasoning benchmark with tabular contexts. **Chameleon** demonstrates flexibility and efficiency in adapting to different queries that require various reasoning abilities.

more effectively [47, 9, 6, 33, 15, 37]. For example, Toolformer [43] constructs tool-use augmented data to train language models to select five tools. In the realm of visual tools, various approaches have been proposed to enhance the capabilities of large language models in handling visual tasks [52, 51, 45, 11, 44], augmented with Hugging Face models [44], Azure models [52], visual foundation models [51].

We compare **Chameleon** with other tool-augmented language models in Table 1. Many of these approaches are either constrained to a small set of tools or limited to task-specific tools, which reduces their capabilities across various skill dimensions and hampers their generalizability to new tasks. A recent line of work relies on large amounts of supervision [43, 21] and focuses on generating commands [34] and programs [45, 11] to infer the choice of tools. However, this approach needs to carefully tailor prompts to specific tasks and particular tools, and is neither flexible

Model	Tool Use						Skill Dimension					Inference & Extension		
	Size						Image	Web	Know.	Math	Table	Composition	Planning	Plug-n-Play
CoT [49]	1	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
Lila [33]	1	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗
PoT [6]	2	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗
Code4Struct [47]	1	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
PAL [9]	2	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗
MathPrompter [15]	2	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗
ART [37]	4	✓	✗	✗	✓	✓	✗	✓	✗	✓	✗	✓	✗	✓
Toolformer [43]	5	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	natural lang.	✗
WebGPT [34]	10	✓	✗	✗	✓	✗	✗	✓	✗	✗	✗	✓	program	✗
MM-ReAct [52]	>10	✓	✗	✗	✓	✗	✓	✓	✓	✓	✓	✓	word match	✓
Visual ChatGPT [51]	>10	✓	-	-	✗	✗	✓	✗	✗	✗	✗	✓	natural lang.	✓
ViperGPT [45]	>10	✓	-	-	✗	✗	✓	✗	✓	✓	✗	✓	program	✓
VisProg [11]	>10	✓	-	-	✗	✓	✓	✗	✗	✗	✗	✓	program	✓
HuggingGPT [44]	>10	✓	✓	✗	✗	✗	✓	✗	✗	✗	✗	✓	natural lang.	✓
Chameleon (ours)	>10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	natural lang.	✓

Table 1: A comparison of work that augments large language models with tool usage. We report the tool size and tool types, including OpenAI ( ), Hugging Face ( ), Github ( ), Web search ( ), and code ( ). We also compare the skills each method possesses, such as image understanding, browser search, knowledge retrieval, mathematical reasoning, and table understanding. Some models can compose various tools and are inherently extendable to new tools. A subset of the work proposes a planner to infer the relevant tools for execution.

nor adaptive. In contrast, **Chameleon** instructs LLMs with natural language instructions that simply describe the roles of each rule and provide a few calling examples, eliminating the need for additional training or tool-specific prompts when learning to compose different tools. More importantly, **Chameleon** offers users flexibility in terms of tool types and sources, updating the underlying LLMs, adding new tools, and adapting to new tasks by updating the instructions. Our work shares the same spirit of AutoGPT [41], an autonomous GPT-4 agent with the artificial general intelligence (AGI) ambition to incorporate numerous tools to achieve user-defined goals. While AutoGPT is still under development, our work is the first to instantiate the idea and verify its effectiveness on well-studied benchmarks.

3 General Framework: Chameleon

Distinct from previous approaches, our **Chameleon** method is capable of synthesizing the composition of various tools to accommodate a wide range of problems. It employs a natural language planner, $\mathcal{P}$, to generate a sequential program to select and compose modules stored in an inventory, $\mathcal{M}$. Then, the program is executed to generate the answer.

We formalize our planner as follows: given the input query x_0, the module inventory $\mathcal{M}$, and constraints $\mathcal{G}$, the natural language planner $\mathcal{P}$ selects a set of modules that can be executed sequentially to solve the problem. In our work, the planner $\mathcal{P}$ is an LLM prompted to generate a sequence of module names in a few-shot setup (see details in the appendix). The module inventory $\mathcal{M}$ consists of a set of pre-built modules: $\{M_i\}$, each corresponding to a tool of various types (Table 2). A T-lengthed plan sampled from $\mathcal{P}$ is denoted as $p = M^1, \ldots, M^T$, where $M^t \in \mathcal{M}$. Formally, a plan p is generated as follows:

$$p \leftarrow \mathcal{P}(x_0; \mathcal{M}; \mathcal{G}), \tag{1}$$

where $\mathcal{G}$ are the constraints for the plan generation, and x_0 the input query (problem statement). Given the generated plan, the corresponding modules for each step are then executed sequentially. When evaluating the module M^t at time step t, the output of the execution y^t is calculated by:

$$y^t \leftarrow M^t(x^{t-1}; c^{t-1}), \tag{2}$$

where x^{t-1} is the input for the current module M^t, and c^{t-1} is the cached information gathered from the history modules.

Tool Types	Tools
OpenAI	Knowledge Retrieval, Query Generator, Row Lookup, Column Lookup, Table Verbalizer, Program Generator, Solution Generator
Hugging Face	Image Captioner
Github	Text Detector
Web Search	Bing Search
Python	Program Verifier, Program Executor
Rule-based	Answer Generator

Table 2: Different types of tools in our module inventory.

Both the problem input x^t and cache c^t for the next module M^{t+1} are updated, respectively, by:

$$\begin{aligned} x^t &\leftarrow \texttt{update_input}(x^{t-1}, y^t), \\ c^t &\leftarrow \texttt{update_cache}(c^{t-1}, y^t). \end{aligned} \tag{3}$$

Finally, the answer a to the problem is generated by the last module M^T:

$$a = y^T \leftarrow M^T(x^{T-1}; c^{T-1}). \tag{4}$$

4 Applications of Chameleon

We demonstrate the applications of Chameleon on two challenging reasoning tasks: ScienceQA [27] and TabMWP [28]. The module inventory is introduced in subsection 4.1, and the specific designs for these two tasks are presented in subsection 4.2 and subsection 4.3, respectively.

4.1 Module Inventory

To accommodate various problem-solving capabilities over a diverse range of problems, it is necessary for our system to leverage a rich module inventory containing various external tools. In this section, we give a high-level description of the module inventory we used and defer the descriptions of detailed implementations to specific experiments. The complete module inventory, denoted as $\mathcal{M}$, is presented in Table 2. Each tool within the inventory is defined as follows:

Knowledge Retrieval (M_{kr}): Retrieving extra background knowledge is crucial for assisting the system in tackling complex problems. Here, the "Knowledge Retrieval" module is designed to obtain domain-specific knowledge in a generative manner, i.e., prompting the LLMs to generate relevant background information for the given query, as done by [45]. Typically, this module is beneficial for providing useful task contexts for solving problems from specialized domains, such as science and mathematics. For instance, if a question is about a tax form table, this module could generate knowledge pertaining to tax procedures, providing valuable context for the task at hand.

Bing Search (M_{bs}): Similar to the generative "Knowledge Retrieval", the goal of the "Bing Search" here is to provide broader coverage of task-relevant knowledge. In contrast to the generative "Knowledge Retrieval" module, the "Bing Search" module demonstrates its strengths in cases where a broader range of information is needed or the query requires up-to-date information from multiple sources. We implement this module using the search engine API to return relevant search results based on the input query. The returned results can then be parsed and utilized by other modules within the system to gather richer context information from diverse sources to support problem-solving more effectively.

Query Generator (M_{qg}): Generally, the original problem does not contain any tailored query for retrieving task-relevant information from the search engine. Thus, the query generator is designed to create search engine queries based on the given problem, which is then used as inputs to the "Bing Search" module. For most cases, it is a good strategy to use the "Query Generator" module before

the "Bing Search". In other words, coupled with the search engine tool, generating more targeted queries generally facilitates both the recall and precision of retrieved information.

Image Captioner (M_{ic}): This module is designed to generate captions for images, which can be employed to offer additional context for the given question. Typically, the "Image Captioner" is utilized when a question requires a semantic understanding of an image. By leveraging pre-trained image captioning models, the "Image Captioner" module produces captions for the input image.

Text Detector (M_{td}): This module is designed to identify text within a given image. Typically, the "Text Detector" is employed when a question requires the extraction of textual information from images containing diagrams, charts, tables, maps, or other visual elements. By effectively detecting text in various formats, this module aids in the analysis and understanding of image-based content.

Row Lookup (M_{rl}): In situations where questions involve tabular context, locating relevant cells to answer the question is often necessary. Large tables with numerous cells can potentially distract the system when attempting to comprehend the data. The "Row Lookup" module addresses this challenge by returning a simplified version of the table, retaining only the rows pertinent to the question. This module accepts a question and a table as input and outputs the simplified table. If all rows are relevant, the original table is returned.

Column Lookup (M_{cl}): Similar to the "Row Lookup" module, the "Column Lookup" module is designed to handle questions involving tabular context by focusing on relevant columns. This module streamlines the table by retaining only the columns that are pertinent to the question. The table keeps the same if all columns are relevant.

Table Verbalizer (M_{tv}): Since the LLM is fundamental to our system, converting structured tables into text is likely to enhance the comprehension of tabular information by various downstream modules as shown by [31] for open-domain question answering. Thus, the module is introduced to convert tables into easily comprehensible descriptions for downstream modules like "Program Generator" and "Solution Generator". This module is typically employed when the table has a small number of rows and columns and is domain-specific, such as stem-and-leaf plots or function tables. By transforming tabular data into natural language, this module facilitates better understanding and processing by subsequent modules.

Program Generator (M_{pg}): Recent work has shown that program-aided approaches are able to improve the logical and mathematical reasoning abilities of large language models [47, 9, 6, 33, 15, 37]. Taking the question and context, the "Program Generator" module generates Python programs that can solve the given question effectively. This module is particularly useful when questions and contexts involve complex computations, such as arithmetic operations on multiple numbers, or when questions require intricate logical operations, such as "if-else" statements.

Program Verifier (M_{pv}): Recent work emphasizes the contribution of the verifying generation to reduce hallucination [39, 32]. Therefore, the "Program Verifier" is designed to ensure the validity and error-free nature of the programs generated by the "Program Generator". The "Program Verifier" checks for syntax errors, logical errors, and other potential issues that may arise during program execution, improving the overall reliability and accuracy of the generated solutions.

Program Executor (M_{pe}): This module executes the program generated by "Program Generator" and produces the result, bridging the gap between program generation and final solution derivation.

Solution Generator (M_{sg}): The "Solution Generator" module aims to generate a detailed solution to the input query, taking into account all the information stored in the cache. Employing a chain-of-thought prompting approach [49], this module ensures coherent and well-structured responses based on the available information. The planner relies on this module only if this module is powerful enough to solve the problem independently, especially for simple problems.

Answer Generator (M_{ag}): This task-specific module extracts and normalizes the answer from the results generated by the "Program Executor" or "Solution Generator" using a rule-based approach. Typically, the "Answer Generator" module serves as the final module in the reasoning pipeline of our **Chameleon**, ensuring a concise and appropriate response based on the preceding steps.

Tool Types	Tools used on ScienceQA	Tools used on TabMWP
OpenAI	**Knowledge Retrieval**, Query Generator, **Solution Generator**	**Knowledge Retrieval**, Row Lookup, Column Lookup, Table Verbalizer, Program Generator, **Solution Generator**
Hugging Face	Image Captioner	
Github	Text Detector	
Web Search	Bing Search	
Python		Program Verifier, Program Executor
Rule-based	**Answer Generator**	**Answer Generator**

Table 3: Tools used on ScienceQA and TabMWP, respectively. The reusable tools in two tasks are highlighted in **green**.

4.2 Science Question Answering

Science Question Answering (ScienceQA [27]) is a multi-modal question-answering benchmark covering a wide range of scientific topics over diverse contexts. As illustrated in Figure 1, answering questions in ScienceQA requires using various knowledge, tools, and skills, such as image captioning, text detection, external knowledge retrieval, online resource search, and visual reasoning based on multiple clues. When generating programs for using tools, we constrain the search space to the relevant part of the entire inventory, as listed in Table 3.

Programs are deemed invalid if the modules "Solution Generator" and "Answer Generator" are not the final two elements. Invalid programs are then set to the default program, consisting of a sequence of "Solution Generator" and "Answer Generator", which refers to the chain-of-thought prompting baseline [49]. The constructed prompt for the natural language planner is displayed in Figure 6. The prompts for LLM-based modules, such as "Knowledge Retrieval", "Query Generator", and "Solution Generator" are shown in Table 7, Table 8, and Table 9, respectively.

4.3 Tabular Mathematical Reasoning

TabMWP [28] is a mathematical reasoning task on tables. There are diverse tabular contexts, including schedules, prices, tax forms, plots, and function relations. As the examples shown in Figure 2, this task requires AI systems to understand tables of various forms from different domains and perform precise numerical/symbolic computations. Similar to ScienceQA, we again constrain the program search space to focus on two types of tools, particularly, 1) tools can help LLMs better digest tabular information, e.g., "Row Lookup", "Column Lookup", and "Table Verbalizer"; 2) tools can perform faithful symbolic computations, such as "Program Generator", "Program Verifier", and "Program Executor". The full list is shown on the right side of Table 3.

The generated programs adhere to additional constraints, such as the inclusion of "Answer Generator", the placement of "Program Generator" prior to "Program Verifier", and the positioning of "Program Generator" before "Program Executor". If these conditions are not met, the programs are regarded as invalid and set as a sequence of "Program Generator", "Program Verifier", "Program Executor", and "Answer Generator". This program corresponds to the program-of-thought prompting baseline [6] with added program verification.

5 Experiments

We conduct experiments on two complex reasoning tasks, ScienceQA [27] and TabMWP [28], to evaluate our **Chameleon** in terms of effectiveness and adaptability.

5.1 Experimental Setup

Planner implementations. We choose the *gpt-3.5-turbo* engine for ChatGPT and the *gpt-4* engine for GPT-4 when constructing the LLM-based planner. The maximum length for generated programs is set to 128, and the temperature is set to 0 for the most deterministic generation.

Model	#Tuned Params	ALL	NAT	SOC	LAN	TXT	IMG	NO	G1-6	G7-12
Heuristic baselines										
Random Choice [27]	-	39.83	40.28	46.13	29.25	47.45	40.08	33.66	39.35	40.67
Human [27]	-	88.40	90.23	84.97	87.48	89.60	87.50	88.10	91.59	82.42
Fine-tuned models										
MCAN [55]	95M	54.54	56.08	46.23	58.09	59.43	51.17	55.40	51.65	59.72
Top-Down [1]	70M	59.02	59.50	54.33	61.82	62.90	54.88	59.79	57.27	62.16
BAN [18]	112M	59.37	60.88	46.57	66.64	62.61	52.60	65.51	56.83	63.94
DFAF [10]	74M	60.72	64.03	48.82	63.55	65.88	54.49	64.11	57.12	67.17
ViLT [19]	113M	61.14	60.48	63.89	60.27	63.20	61.38	57.00	60.72	61.90
Patch-TRM [29]	90M	61.42	65.19	46.79	65.55	66.96	55.28	64.95	58.04	67.50
VisualBERT [22, 23]	111M	61.87	59.33	69.18	61.18	62.71	62.17	58.54	62.96	59.92
UnifiedQA [17]	223M	70.12	68.16	69.18	74.91	63.78	61.38	77.84	72.98	65.00
UnifiedQA CoT [27]	223M	74.11	71.00	76.04	78.91	66.42	66.53	81.81	77.06	68.82
MM-COT$_T$ [57]	223M	70.53	71.09	70.75	69.18	71.16	65.84	71.57	71.00	69.68
MM-COT [57]	223M	84.91	87.52	77.17	85.82	87.88	82.90	86.83	84.65	85.37
MM-COT$_{Large}$ [57]	738M	91.68	95.91	82.00	90.82	95.26	88.80	92.89	92.44	90.31
LLaMA-Adapter$_T$ [56]	1.2M	78.31	79.00	73.79	80.55	78.30	70.35	83.14	79.77	75.68
LLaMA-Adapter [56]	1.8M	85.19	84.37	88.30	84.36	83.72	80.32	86.90	85.83	84.05
Few-shot GPT-3										
GPT-3 [4]	0M	74.04	75.04	66.59	78.00	74.24	65.74	79.58	76.36	69.87
GPT-3 CoT [27]	0M	75.17	75.44	70.87	78.09	74.68	67.43	79.93	78.23	69.68
Published results (Above) ▲										
Few-shot ChatGPT										
ChatGPT CoT	0M	78.31	78.82	70.98	83.18	77.37	67.92	86.13	80.72	74.03
Chameleon (ChatGPT)	0M	79.93	81.62	70.64	84.00	79.77	70.80	86.62	81.86	76.53
Few-shot GPT-4										
GPT-4 CoT	0M	83.99	85.48	72.44	90.27	82.65	71.49	92.89	86.66	79.04
Chameleon (GPT-4)	0M	**86.54**	**89.83**	**74.13**	**89.82**	**88.27**	**77.64**	**92.13**	**88.03**	**83.72**

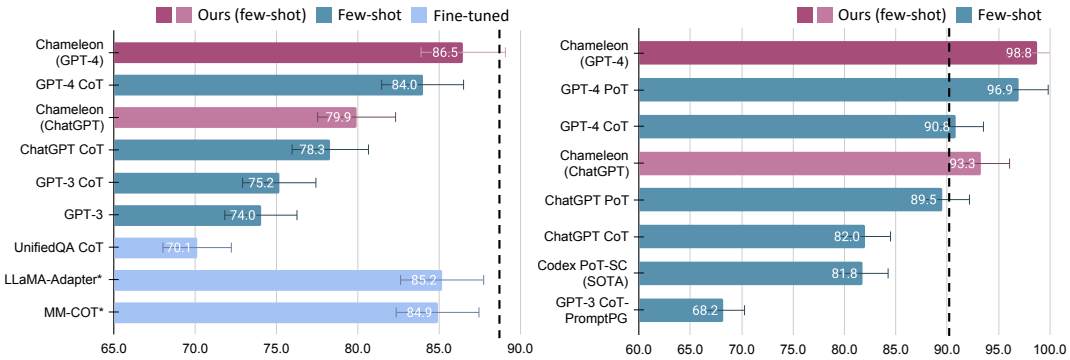
Table 4: **QA accuracy (%) on the test set of ScienceQA** [27]. We report the number of tuned parameters for this task and the overall accuracy, along with accuracy scores for different question types, including natural, social, and language sciences, text, image, and no context, as well as grades 1-6 and 7-12. The highest scores among models in each section and overall are highlighted in blue and red, respectively, and the results of our best model are marked in **bold**.

Modules implementations. The "Bing Search" module calls the Bing Search API[1] and returns the top three responses for the text query. For the "Image Captioner" module, we use the captioning model [2] to generate textual descriptions for input images. The maximum length of generated captions is set to 16, the number of beams is 4, and the maximum number of output tokens is 512. The "Text Detector" tool is based on the github model [3] to extract the text contents with coordinates in the image. The "Row Lookup" module is enabled only when there are more than three rows and 18 table cells, in order to accelerate inference. Similarly, the Column Lookup" module is enabled with two or more columns and 18 or more table cells. On ScienceQA, the Answer Generator" module extracts the answer snippet from the result provided by the Solution Generator" and selects the most similar option from the given choices. For TabMWP, the "Answer Generator" is used to normalize answers with two-place precision for questions with numerical answers and select the most similar option for multiple-choice questions. For more details, please refer to our code implementations at **https://github.com/lupantech/chameleon-llm**.

5.2 Experimental Results

ScienceQA. Table 4 presents the results of existing baselines and our **Chameleon**, with key findings highlighted in Figure 3 (a). Employing ChatGPT [35] as the base LLM, our **Chameleon** achieves an overall accuracy of 79.93%, a 1.62% improvement over Chain-of-Thought (CoT) [49] prompted ChatGPT. Notably, **Chameleon** is a generalized form of CoT, where the generated program is a sequence of "Solution Generator" and "Answer Generator". **Chameleon** benefits from additional

[1] https://www.microsoft.com/bing
[2] https://huggingface.co/nlpconnect/vit-gpt2-image-captioning
[3] https://github.com/JaidedAI/EasyOCR



(a) Results on ScienceQA (b) Results on TabMWP

Figure 3: Performance comparison of main baselines and our **Chameleon** on ScienceQA and TabMWP. The vertical dashed line represents human performance and * represents models fine-tuned in multi-modal settings. On ScienceQA, **Chameleon** using GPT-4 surpasses the previously published best few-shot model and attains results comparable to fine-tuned models. On TabMWP, **Chameleon** with GPT-4 achieves state-of-the-art results, exceeding human performance by 8.6%.

tool usage, such as "Knowledge Retrieval", "Bing Search", "Image Captioner", and "Text Detector". When built upon GPT-4 [36], our model attains an accuracy of 86.54%, outperforming GPT-4 CoT [27] by 2.55% and GPT-3 CoT by 11.37%, creating the new state of the art in the few-shot setting.

TabMWP. Results are shown in Table 5, with key models highlighted in Figure 3 (b). Similar significant improvements are observed for **Chameleon** over both fine-tuned and few-shot prompted models. It is worth noting that both CoT and Program-of-Thought (PoT) [6] can be viewed as special cases of **Chameleon**. Specifically, CoT does not use any tool whereas PoT only resorts to symbolic programming tools in our inventory, e.g., "Program Generator", "Program Executor", and "Answer Generator". **Chameleon** (ChatGPT) achieves improvements of 11.25% and 3.79% over ChatGPT CoT and ChatGPT PoT, respectively, demonstrating the advantage of our enriched tool set. By updating the underlying LLM with GPT-4, our **Chameleon** achieves a further gain of 5.50%, resulting in a 98.78% accuracy. Remarkably, our **Chameleon** (GPT-4) not only surpasses the previous best-performing model (Codex PoT-SC [6]) by 17.0%, but also outperforms the human performance by 8.56%.

5.3 Qualitative Analysis

Generated program statistics. **Chameleon** utilizes the LLM-based natural language planner to generate programs, i.e., sequences of used modules (tools). We report the statistics of the number of unique generated programs and the average length of corresponding tool sequences by our **Chameleon** in Table 6. On both ScienceQA and TabMWP, using GPT-4 as the base LLM generates fewer distinct programs, i.e., more consistent programs, than using ChatGPT, even when given the exact same prompt in the planning model. Our results are consistent with the findings in [36], which observes that GPT-4 has a superior capability of understanding long contexts, aligning with human instructions, and performing high-level reasoning compared to other LLMs such as ChatGPT.

Tool use planning. We visualize the proportions of key tools called and not called in the generated programs from **Chameleon** (ChatGPT) and **Chameleon** (GPT-4) on ScienceQA (shown in Figure 4) and on TabMWP (shown in Figure 5, respectively. Interestingly, ChatGPT and GPT-4 exhibit different planning behaviors. Generally, ChatGPT has a strong bias toward using or not using certain tools, which is highly influenced by the bias from in-context examples. For instance, ChatGPT calls "Knowledge Retrieval" in 72% of queries but only calls "Bing Search" in 3% of cases on ScienceQA; similarly, ChatGPT heavily relies on "Row Lookup" (59%) but calls "Column Lookup" less frequently (5%). However, GPT-4 acts more *objectively* and *rationally* when determining which tool to use or not. For example, GPT-4 calls "Knowledge Retrieval" more frequently (81% vs. 72%) and calls "Bing Search" more than ChatGPT (11% vs. 3%) when answering scientific questions on ScienceQA. More impressively, GPT-4 consistently calls "Query Generator" and "Bing Search" simultaneously by observing the language descriptions of tool use, while ChatGPT lacks such reasoning capability.

Model	#Tuned Params	ALL	FREE	MC	INT	DEC	EXTR	BOOL	OTH	G1-6	G7-8
Heuristic baselines											
Heuristic guess	-	15.29	6.71	39.81	8.37	0.26	30.80	51.22	26.67	17.55	12.27
Human performance	-	90.22	84.61	93.32	84.95	83.29	97.18	88.69	96.20	94.27	81.28
Fine-tuned models											
UnifiedQA$_{\text{SMALL}}$ [17]	41M	29.79	22.27	51.31	27.27	2.83	52.28	48.11	69.52	35.85	21.71
UnifiedQA$_{\text{BASE}}$ [17]	223M	43.52	34.02	70.68	40.74	7.90	84.09	55.67	73.33	53.31	30.46
UnifiedQA$_{\text{LARGE}}$ [17]	738M	57.35	48.67	82.18	55.97	20.26	94.63	68.89	79.05	65.92	45.92
TAPEX$_{\text{BASE}}$ [25]	139M	48.27	39.59	73.09	46.85	11.33	84.19	61.33	69.52	56.70	37.02
TAPEX$_{\text{LARGE}}$ [25]	406M	58.52	51.00	80.02	59.92	16.31	95.34	64.00	73.33	67.11	47.07
Zero-shot GPT-3											
GPT-3 [4]	0M	56.96	53.57	66.67	55.55	45.84	78.22	55.44	54.29	63.37	48.41
GPT-3 CoT [49]	0M	57.61	54.36	66.92	55.82	48.67	78.82	55.67	51.43	63.62	49.59
Few-shot GPT-3											
GPT-3 [4]	0M	57.13	54.69	64.11	58.36	40.40	75.95	52.41	53.02	63.10	49.16
GPT-3 CoT [49]	0M	62.92	60.76	69.09	60.04	63.58	76.49	61.19	67.30	68.62	55.31
GPT-3 CoT-PromptPG [28]	0M	68.23	66.17	74.11	64.12	74.16	76.19	72.81	65.71	71.20	64.27
Codex* [5]	0M	59.4	-	-	-	-	-	-	-	-	-
Codex PoT* [6]	0M	73.2	-	-	-	-	-	-	-	-	-
Codex PoT-SC* [6]	0M	81.8	-	-	-	-	-	-	-	-	-
Published results (Above) ▲											
Few-shot ChatGPT											
ChatGPT CoT	0M	82.03	78.43	92.32	75.38	90.30	92.30	92.89	87.62	83.06	80.66
ChatGPT PoT	0M	89.49	90.24	87.35	89.31	93.82	92.10	85.89	55.24	90.60	88.00
Chameleon (ChatGPT)	0M	93.28	93.13	93.72	92.71	94.76	91.29	98.11	78.85	93.37	93.17
Few-shot GPT-4											
GPT-4 CoT	0M	90.81	88.48	97.49	86.16	97.51	96.86	99.11	89.52	92.40	88.70
GPT-4 PoT	0M	96.93	97.40	95.58	98.48	93.22	96.25	98.00	68.57	96.97	96.87
Chameleon (GPT-4)	0M	**98.78**	**98.95**	**98.29**	**99.34**	**97.42**	**98.58**	**98.56**	**93.33**	**98.95**	**98.54**

Table 5: **QA accuracy (%) on the test set of TabMWP** [28]. We report the number of tuned parameters for this task and the overall accuracy, and accuracy of different question types, including free-text questions, multi-choice questions, integer answers, decimal answers, extractive answers, Boolean answers, other text answers, grades 1-6, and grades 7-8. * refers to the results of 1,000 test examples. The highest scores among models in each section and overall are marked in blue and red, respectively, and the results of our best model are marked in **bold**.

Task	Model	# of different programs	Average program length
ScienceQA	Chain-of-thought (CoT)	1	2
	Chameleon (ChatGPT)	14	3.03
	Chameleon (GPT-4)	11	3.40
TabMWP	Chain-of-thought (CoT)	1	2
	Program-of-thought (PoT)	1	3
	Chameleon (ChatGPT)	28	4.17
	Chameleon (GPT-4)	19	4.09

Table 6: The statistics of the number of different generated programs and the average length of generated programs by our Chameleon, respectively. Chain-of-thought (CoT) prompting and Program-of-thought (PoT) prompting are also compared as they are the special cases of Chameleon.

We visualize the transition graphs of modules for generated programs by Chameleon (GPT-4) on ScienceQA and TabMWP in Figure 8 and Figure 9 respectively. The transition probabilities in these graphs are computed from the tool transitions observed on the test sets of these two datasets. These graphs illustrate that the GPT-4 planner is able to make good decisions on how to sequence tools in a few-shot setup. For example on ScienceQA, Chameleon often decides to rely on either the knowledge retriever or Bing search, but rarely both. On TabMWP, we observe two main modes: either going through the solution-executor module, or via the program verifier and executor.

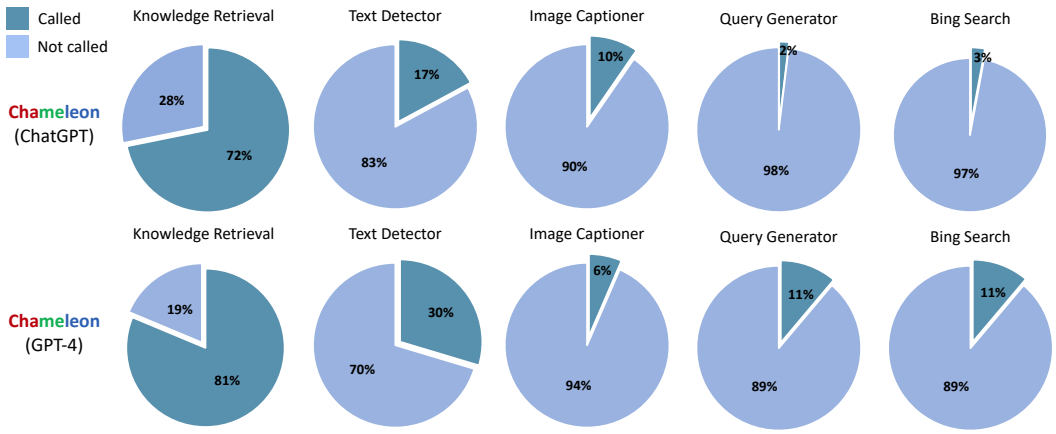


Figure 4: Tools called in the generated programs from **Chameleon** (ChatGPT) and **Chameleon** (GPT-4) on ScienceQA.

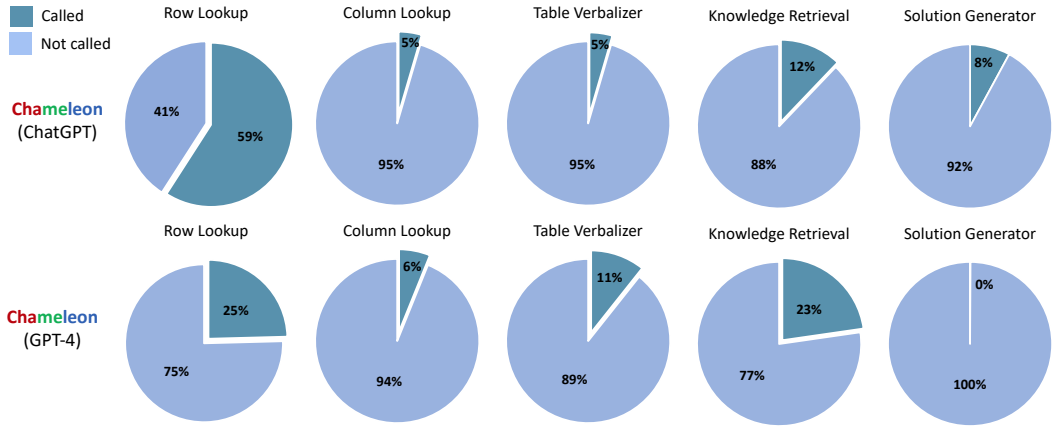


Figure 5: Tools called in the generated programs from **Chameleon** (ChatGPT) and **Chameleon** (GPT-4) on TabMWP.

5.4 Case Study

Three examples from **Chameleon** (GPT-4) on ScienceQA are visualized in Figure 1. **Chameleon** (GPT-4) is able to adapt to different input queries by generating programs that compose various tools and executing them sequentially to obtain the correct answers. For instance, to answer the first question, "*What is the direction of this push?*", the system calls the image captioner model to extract semantic information from the image and employs the knowledge retrieval model to gather background knowledge for multi-modal reasoning. In the second example, the natural language planner infers that a text detector tool is needed to understand the context of the ad. In the third example (with more details provided in Figure 10 in the appendix), the query asks, "*Which animal's skin is adapted for survival in cold places?*", which involves scientific terminology related to animal survival. Consequently, the planner decides to rely on the Bing search engine for domain-specific knowledge, benefiting from the numerous online resources available.

The adaptability and versatility of our **Chameleon** for various queries are also observed on TabMWP, as illustrated in the examples in Figure 2. The first example involves mathematical reasoning on a tax form. **Chameleon** (1) calls the knowledge retrieval model to recall basic knowledge that assists in understanding such domain-specific tables, (2) describes the table in a more readable natural language format, and (3) finally relies on program-aided tools to perform precise computations. In the second example, the system generates Python code that closely aligns with the background knowledge provided by the knowledge retrieval model. The third example requires the system to locate the cell in a large tabular context given the input query. **Chameleon** calls the row lookup model to help

accurately locate the relevant rows and generate the language solution via an LLM model, instead of relying on program-based tools.

6 Conclusion

In conclusion, we have introduced a novel *plug-and-play compositional* reasoning framework, **Chameleon**, that leverages large language models to overcome the inherent limitations of current tool-augmented LLMs. Our approach employs a diverse set of tools and demonstrates impressive adaptability and effectiveness on two challenging benchmarks, ScienceQA and TabMWP. By achieving significant improvements in accuracy over existing few-shot and state-of-the-art models, **Chameleon** showcases its potential for addressing real-world queries across various domains. As research continues to advance in large language models and tool integration, we anticipate that our framework will serve as a foundation for further innovations in pursuing more generalizable and efficient solutions to complex reasoning tasks.

7 Acknowledgment

We would like to thank Chunyuan Li, Qiuyuan Huang, and other members of the Deep Learning group at Microsoft Research for their valuable discussions.

References

[1] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[2] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1545–1554, 2016.

[3] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 39–48, 2016.

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

[5] 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. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

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

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

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

[10] Peng Gao, Zhengkai Jiang, Haoxuan You, Pan Lu, Steven CH Hoi, Xiaogang Wang, and Hongsheng Li. Dynamic fusion with intra-and inter-modality attention flow for visual question answering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6639–6648, 2019.

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

[12] Ronghang Hu, Jacob Andreas, Trevor Darrell, and Kate Saenko. Explainable neural computation via stack neural module networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 53–69, 2018.

[13] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 804–813, 2017.

[14] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *6th Annual Conference on Robot Learning*, 2022.

[15] Shima Imani, Liang Du, and Harsh Shrivastava. MathPrompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*, 2023.

[16] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, pages 2989–2998, 2017.

[17] Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. UnifiedQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics (EMNLP)*, pages 1896–1907, 2020.

[18] Jin-Hwa Kim, Jaehyun Jun, and Byoung-Tak Zhang. Bilinear attention networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1571–1581, 2018.

[19] Wonjae Kim, Bokyung Son, and Ildoo Kim. ViLT: Vision-and-language transformer without convolution or region supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 5583–5594, 2021.

[20] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

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

[22] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. VisualBERT: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*, 2019.

[23] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. What does BERT with vision look at? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5265–5275, 2020.

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

[25] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. TAPEX: Table pre-training via learning a neural sql executor. In *International Conference on Learning Representations (ICLR)*, 2022.

[26] Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning. In *The 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

[27] Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *The 36th Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[28] Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *International Conference on Learning Representations (ICLR)*, 2023.

[29] Pan Lu, Liang Qiu, Jiaqi Chen, Tony Xia, Yizhou Zhao, Wei Zhang, Zhou Yu, Xiaodan Liang, and Song-Chun Zhu. IconQA: A new benchmark for abstract diagram understanding and visual language reasoning. In *The 35th Conference on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*, 2021.

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

[31] Kaixin Ma, Hao Cheng, Xiaodong Liu, Eric Nyberg, and Jianfeng Gao. Open domain question answering with a unified knowledge interface. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1605–1620, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[32] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-Refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

[33] Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. Lila: A unified benchmark for mathematical reasoning. In *The 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

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

[35] OpenAI. Chatgpt, 2022.

[36] OpenAI. GPT-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[37] 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. *arXiv preprint arXiv:2303.09014*, 2023.

[38] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 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, 2021.

[39] Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *arXiv preprint arXiv:2302.12813*, 2023.

[40] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with GPT-4. *arXiv preprint arXiv:2304.03277*, 2023.

[41] Toran Bruce Richards. Auto-GPT: An experimental open-source attempt to make GPT-4 fully autonomous. https://github.com/Significant-Gravitas/Auto-GPT, 2023.

[42] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

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

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

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

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

[47] Xingyao Wang, Sha Li, and Heng Ji. Code4Struct: Code generation for few-shot structured prediction from natural language. *arXiv preprint arXiv:2210.12810*, 2022.

[48] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[49] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[50] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32, 1992.

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

[52] Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. MM-REACT: Prompting ChatGPT for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

[53] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. Neural-symbolic VQA: Disentangling reasoning from vision and language understanding. *Advances in neural information processing systems (NeurIPS)*, 31, 2018.

[54] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators. In *International Conference on Learning Representations (ICLR)*, 2023.

[55] Zhou Yu, Jun Yu, Yuhao Cui, Dacheng Tao, and Qi Tian. Deep modular co-attention networks for visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6281–6290, 2019.

[56] Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Qiao Yu. LLaMA-Adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.

[57] Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*, 2023.

A Appendix

▷ *Instruction for the planner model*

You need to act as a policy model, that given a question and a modular set, determines the sequence of modules that can be executed sequentially can solve the question.

The modules are defined as follows:

Query_Generator: This module generates a search engine query for the given question. Normally, we consider using "Query_Generator" when the question involves domain-specific knowledge.
Bing_Search: This module searches the web for relevant information to the question. Normally, we consider using "Bing_Search" when the question involves domain-specific knowledge.
Image_Captioner: This module generates a caption for the given image. Normally, we consider using "Image_Captioner" when the question involves the semantic understanding of the image, and the "has_image" field in the metadata is True.
Text_Detector: This module detects the text in the given image. Normally, we consider using "Text_Detector" when the question involves the unfolding of the text in the image, e.g., diagram, chart, table, map, etc., and the "has_image" field in the metadata is True.
Knowledge_Retrieval: This module retrieves background knowledge as the hint for the given question. Normally, we consider using "Knowledge_Retrieval" when the background knowledge is helpful to guide the solution.
Solution_Generator: This module generates a detailed solution to the question based on the information provided. Normally, "Solution_Generator" will incorporate the information from "Query_Generator", "Bing_Search", "Image_Captioner", "Text_Detector", and "Knowledge_Retrieval".
Answer_Generator: This module extracts the final answer in a short form from the solution or execution result. This module normally is the last module in the prediction pipeline.

Below are some examples that map the problem to the modules.

▷ *In-context example(s)*

Question: Compare the average kinetic energies of the particles in each sample. Which sample has the higher temperature?

Context: The diagrams below show two pure samples of gas in identical closed, rigid containers. Each colored ball represents one gas particle. Both samples have the same number of particles.

Options: (A) neither; the samples have the same temperature (B) sample A (C) sample B

Metadata: 'pid': 19, 'has_image': True, 'grade': 8, 'subject': 'natural science', 'topic': 'physics', 'category': 'Particle motion and energy', 'skill': 'Identify how particle motion affects temperature and pressure'

Modules: `["Text_Detector", "Knowledge_Retrieval", "Solution_Generator", "Answer_Generator"]`

Figure 6: The prompt constructed for the planner model on the ScienceQA task. The prompt consists of the instruction that describes the role of the planner model, the in-context examples that map the problem to the module sequence, and the test example.

▷ *Instruction for the planner model*

You need to act as a policy model, that given a question and a modular set, determines the sequence of modules that can be executed sequentially can solve the question.

The modules are defined as follows:

Program_Generator: This module generates a Python program that can solve the given question. It takes in the question and possible context and produces a program that can be executed by the "Program_Executor" module. Normally, we consider using "Program_Generator" when the questions and contexts involve complex computation, such as arithmetic operations over multiple numbers, or when the questions involve complex logical operations, such as "if-else" statements.
Program_Verifier: This module verifies whether the generated program from "Program_Generator" is valid and error-free. It checks for syntax errors, logical errors, and other potential issues that may arise during program execution.
Program_Executor: This module executes the generated program from "Program_Generator" and produces an output that can be further processed by other modules, such as "Question_Answering".
Row_Lookup: This module returns the simplified table that only remains the rows that are relevant to the question. It takes in the question and a table and returns the simplified table. If all rows are relevant or there are only three rows or fewer, return the original table. Normally, we only consider using "Row_Lookup" when the table involves more than three rows and the question only requires a small number of rows to answer the question.
Column_Lookup: This module returns the simplified table that only remains the columns that are relevant to the question. It takes in the question and a table and returns the simplified table. If all columns are relevant or there are only two columns, return the original table. Normally, we consider using "Column_Lookup" when the table involves more than two columns and the question only requires a small number of columns to answer the question.
Table_Verbalizer: This module converts the table to a description that can be easily understood by the downstream modules, like "Program_Generator", "Solution_Generator", "Question_Answering". Normally, we consider using "Table_Verbalizer" when the table involves a small number of rows and columns and the table is domain-specific, such as steam-and-leaf plots, function tables, etc.
Knowledge_Retrieval: This module retrieves domain-specific knowledge for the given question and table. Normally, we consider using "Knowledge_Retrieval" when the question and table involve domain-specific knowledge, such as "steam-and-leaf plots", "function tables", "tax forms", etc.
Solution_Generator: This module generates a detailed solution to the question based on the information provided. Normally, we use "Solution_Generator" when the question and table involve simple computation, such as arithmetic operations over a single number.
Answer_Generator: This module extracts the final answer in a short form from the solution or execution result. This module normally follows the "Solution_Generator" or "Problem_Executor" module.

Below are some examples that map the problem to the modules.

▷ *In-context example(s)*

Table:
designer watch | $8,141
designer coat | $6,391

Question: How much more does a designer watch cost than a designer coat? (unit: $)

Modules: `["Program_Generator", "Program_Verifier", "Program_Executor", "Answer_Generator"]`

Figure 7: The prompt constructed for the planner model on the TabMWP task. Similarly, the prompt consists of the instruction, the in-context examples, and the test example.

▷ *Instruction*

Read the following question, and generate the background knowledge as the context information that could be helpful for answering the question.

▷ *In-context example(s)*

Question: Which property do these three objects have in common?

Options: (A) hard (B) soft (C) yellow

Metadata: 'pid': 43, 'has_image': True, 'grade': 4, 'subject': 'natural science', 'topic': 'physics', 'category': 'Materials', 'skill': 'Compare properties of objects'

Detected text in the image: ['handkerchief', 'slippers', 'leisure suit']

Knowledge:
- This question is about comparing the properties of three objects: a handkerchief, slippers, and a leisure suit.
- The objects are related to the topic of physics and the skill of comparing properties of objects.
- Properties of objects can include physical characteristics such as color, texture, shape, size, weight, and material.

Table 7: The prompt constructed for the "Knowledge Retrieval" module on the ScienceQA task.

▷ *Instruction*

Read the following question and metadata, and generate the query for browser search as the context information that could be helpful for answering the question.

▷ *In-context example(s)*

Question: Which property do these two objects have in common?

Options: (A) hard (B) bendable

Metadata: 'pid': 329, 'has_image': True, 'grade': 2, 'subject': 'natural science', 'topic': 'physics', 'category': 'Materials', 'skill': 'Compare properties of objects'

Detected text in the image: [([[41, 183], [131, 183], [131, 199], [41, 199]], 'rubber gloves'), ([[245, 183], [313, 183], [313, 197], [245, 197]], 'rain boots')]

Search Query: Common material properties of jump rope and rubber gloves

Table 8: The prompt constructed for the "Query Generator" module on the ScienceQA task.

▷ *Instruction*

Given the question (and the context), select the answer from the options ["A", "B", "C", "D", "E"]. You should give concise and step-by-step solutions. Finally, conclude the answer in the format of "the answer is [ANSWER]", where [ANSWER] is one from the options ["A", "B", "C", "D", "E"]. For example, "the answer is A", "the answer is B", "the answer is C", "the answer is D", or "the answer is E". If the answer is not in the options, select the most possible option.

▷ *In-context example(s)*

Question: Which property do these two objects have in common?

Context: Select the better answer.

Options: (A) hard (B) bendable

Metadata: 'pid': 6493, 'has_image': True, 'grade': 2, 'subject': 'natural science', 'topic': 'physics', 'category': 'Materials', 'skill': 'Compare properties of objects'

Image caption: A pair of scissors next to a pair of scissors.

Detected text with coordinates in the image: [([[53, 185], [121, 185], [121, 199], [53, 199]], 'jump rope'), ([[233, 183], [323, 183], [323, 201], [233, 201]], 'rubber gloves')]

Retrieved knowledge:
- This question is about comparing the properties of two objects: rubber gloves and rain boots.
- The objects are related to the topic of physics and the skill of comparing properties of objects.
- Properties of objects can include physical characteristics such as color, texture, shape, size, weight, and material. In this case, the two objects have the property of being bendable in common.

Bing search response: The most common materials used for disposable gloves are Latex, Vinyl and Nitrile. Each material has its benefits and drawbacks. Latex Gloves are constructed from Natural Rubber Latex and are the most popular type of disposable glove.

Solution: An object has different properties. A property of an object can tell you how it looks, feels, tastes, or smells. Different objects can have the same properties. You can use these properties to put objects into groups. Look at each object. For each object, decide if it has that property. A bendable object can be bent without breaking. Both objects are bendable. A hard object keeps its shape when you squeeze it. The rubber gloves are not hard. The property that both objects have in common is bendable. Therefore, the answer is B.

Table 9: The prompt constructed for the "Solution Generator" module on the ScienceQA task.

▷ *Instruction*

Read the following table and question, and generate the domain-specific knowledge as the context information that could be helpful for answering the question.

▷ *In-context example(s)*

Table:
x | y
10 | 15
11 | 9
12 | 2

Question: The table shows a function. Is the function linear or nonlinear?

Knowledge:
- A linear function is a function whose graph is a straight line.
- A nonlinear function is a function whose graph is not a straight line.
- The equation of a linear function is $y = mx + b$, where m is the slope and b is the y-intercept.
- The equation of a nonlinear function is not $y = mx + b$.

Table 10: The prompt constructed for the "Knowledge Retrieval " module on the TabMWP task.

▷ *Instruction*

Read the following question and table. Each row is separated by a newline ('\n') and each column is separated by a vertical bar ('|'). Return the simplified table that only remains the rows that are relevant to the question. If all rows are relevant, or the number of rows is fewer than three, return the original table.

▷ *In-context example(s)*

Question: In preparation for graduation, some teachers and students volunteered for the various graduation committees. How many people are on the music committee?

Table:
Committee | Students | Teachers
Program | 5 | 17
Ticket | 20 | 5
Music | 20 | 15
Schedule | 15 | 20
Food | 18 | 2

Simplified Table:
Committee | Students | Teachers
Music | 20 | 15

Table 11: The prompt constructed for the "Row Lookup" module on the TabMWP task.

▷ *Instruction*

Read the following question and table. Each row is separated by a newline ('\n') and each column is separated by a vertical bar ('|'). Return the simplified table that only remains the columns that are relevant to the question. If all columns are relevant, return the original table.

▷ *In-context example(s)*

Question: Look at the following schedule. When does Recess end?

Table:
Subject | Begin | End
Recess | 6:15 A.M. | 7:20 A.M.
Orchestra | 7:30 A.M. | 8:40 A.M.
Art | 8:45 A.M. | 9:35 A.M.
Handwriting | 9:45 A.M. | 10:20 A.M.
Gym | 10:30 A.M. | 11:15 A.M.
Choir | 11:20 A.M. | 12:25 P.M.
Science | 12:35 P.M. | 1:35 P.M.
Reading | 1:40 P.M. | 2:50 P.M.

Simplified Table:
Subject | End
Recess | 7:20 A.M.
Orchestra | 8:40 A.M.
Art | 9:35 A.M.
Handwriting | 10:20 A.M.
Gym | 11:15 A.M.
Choir | 12:25 P.M.
Science | 1:35 P.M.
Reading | 2:50 P.M.

Table 12: The prompt constructed for the "Column Lookup" module on the TabMWP task.

▷ *Instruction*

Read the following question and table. Write a textual description of the table. The description should keep the critical information in the table for answering the question. The description should not answer the question.

▷ *In-context example(s)*

Table:
Committee | Students | Teachers
Program | 5 | 17
Ticket | 20 | 5
Music | 20 | 15
Schedule | 15 | 20
Food | 18 | 2

Table description: The table shows the number of students and teachers on each of the four graduation committees: Program, Ticket, Music, and Schedule. The Music committee has 20 students and 15 teachers.

Table 13: The prompt constructed for the "Table Verbalizer" module on the TabMWP task.

▷ *Instruction*

Read the following table and then write Python code to answer a question.

▷ *In-context example(s)*

Table:
Price | Quantity demanded | Quantity supplied
$895 | 21,000 | 3,400
$945 | 17,200 | 7,400
$995 | 13,400 | 11,400
$1,045 | 9,600 | 15,400
$1,095 | 5,800 | 19,400

Questions: Look at the table. Then answer the question. At a price of $995, is there a shortage or a surplus? Please select from the following options: ['shortage', 'surplus'].

Code:

```
# Python Code, return 'ans'. Make sure that 'ans' is a string selected
    from the options in the question
quantity_demanded_at_price_955 = 13400
quantity_supplied_at_price_955 = 11400
if quantity_demanded_at_price_955 > quantity_supplied_at_price_955:
    ans = 'shortage'
else:
    ans = 'surplus'
```

Table 14: The prompt constructed for the "Program Generator" module on the TabMWP task.

▷ *Instruction*

Read the following table and then answer a question.

▷ *In-context example(s)*

Table:
Price | Quantity demanded | Quantity supplied
$895 | 21,000 | 3,400
$945 | 17,200 | 7,400
$995 | 13,400 | 11,400
$1,045 | 9,600 | 15,400
$1,095 | 5,800 | 19,400

Question: Look at the table. Then answer the question. At a price of $995, is there a shortage or a surplus? Please select from the following options: [‘shortage’, ‘surplus’].

Solution: At the price of $995, the quantity demanded is greater than the quantity supplied. There is not enough of the good or service for sale at that price. So, there is a shortage. The answer is shortage.

Table 15: The prompt constructed for the “Solution Generator” module on the TabMWP task.

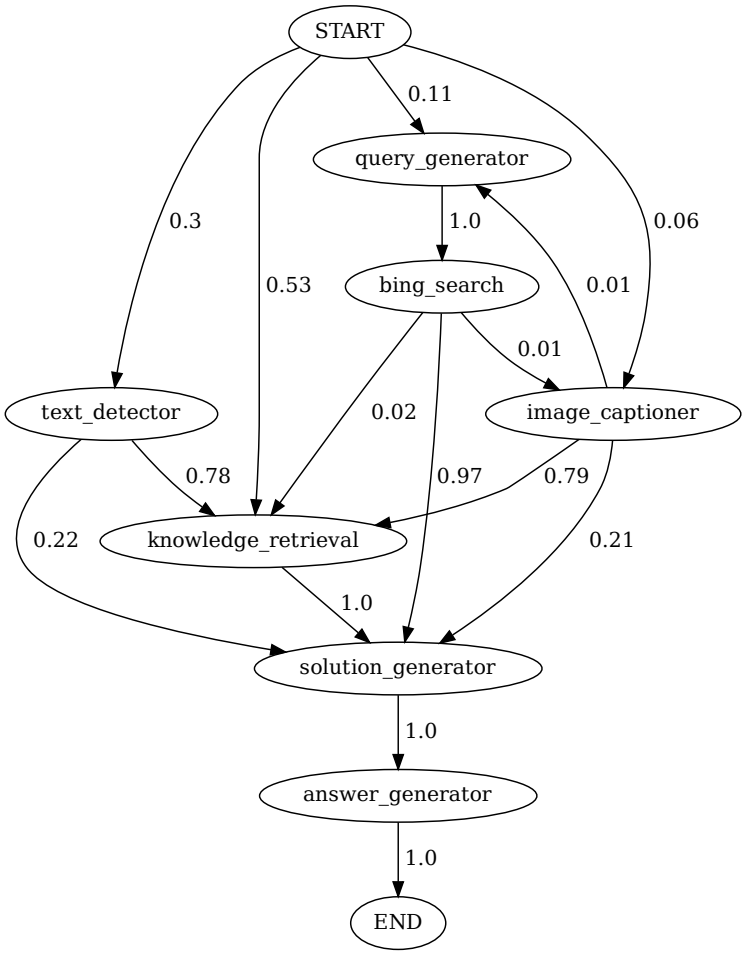


Figure 8: Transitions between modules in programs generated by **Chameleon** (GPT-4) on ScienceQA. START is the start symbol, END is a terminal symbol and the others are non-terminal symbols.

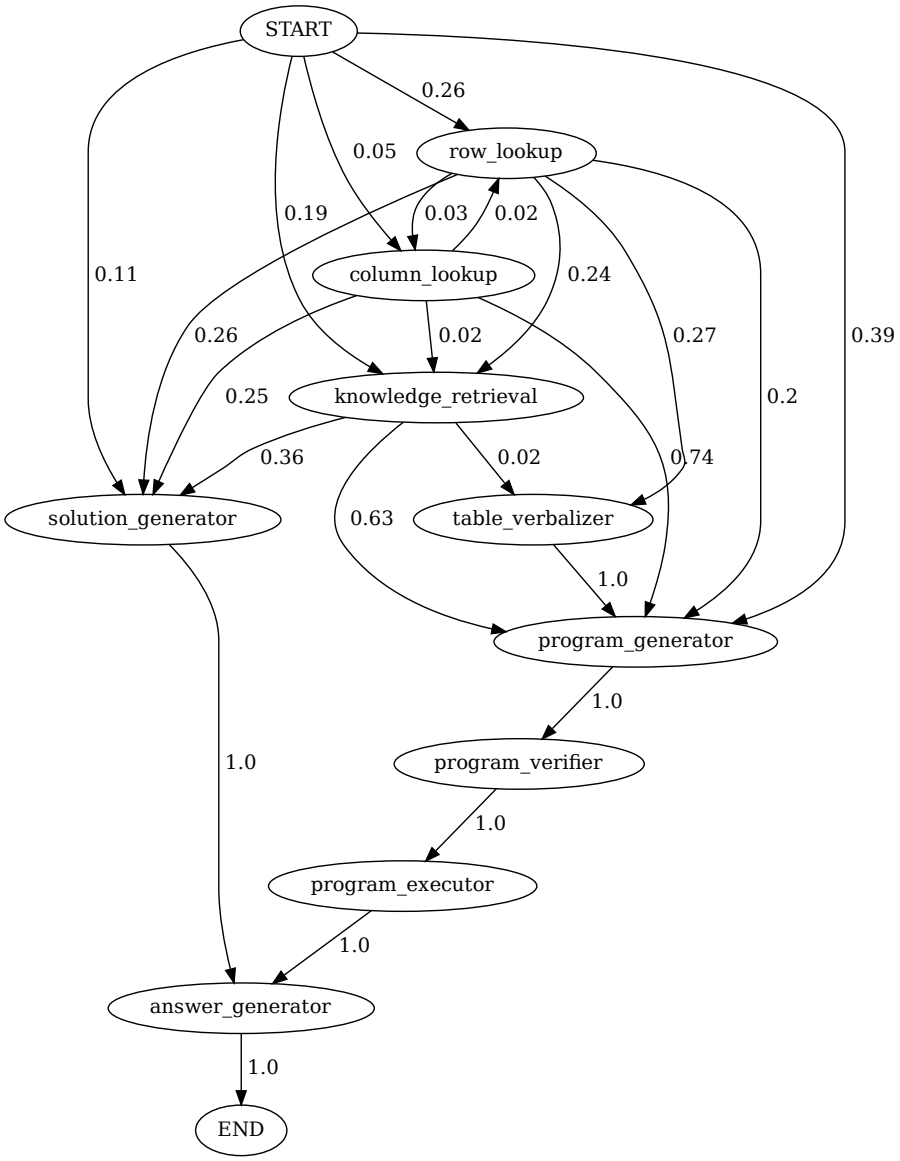


Figure 9: Transitions between modules in programs generated by **Chameleon** (GPT-4) on TabMWP. START is the start symbol, END is a terminal symbol and the others are non-terminal symbols.

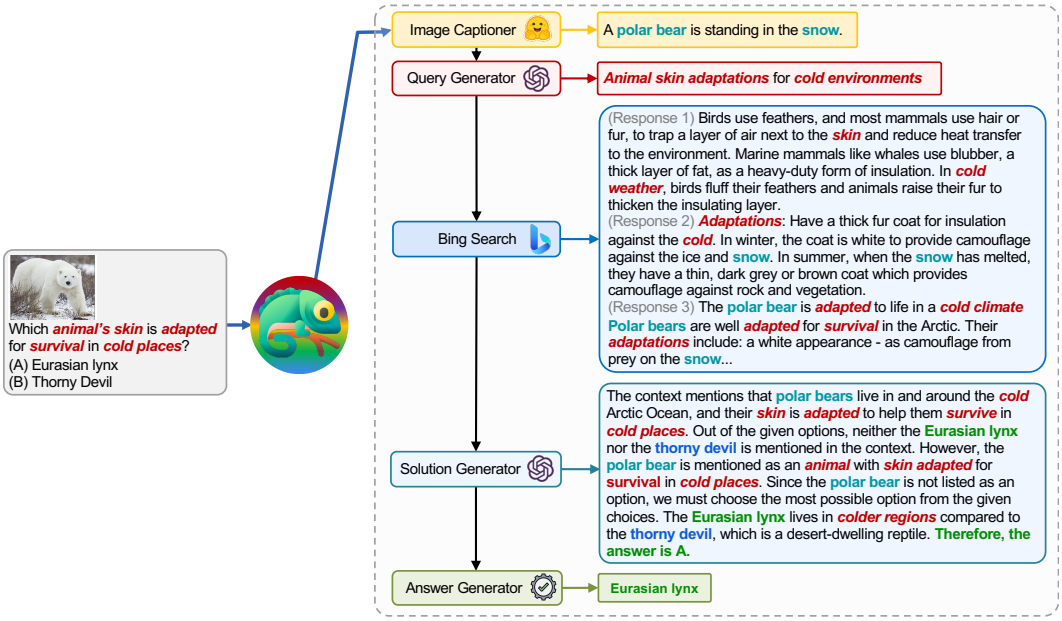


Figure 10: One example from our **Chameleon** (GPT-4) on ScienceQA.