

题目

摘要

摘要 大型语言模型(1lm)由于其强大的推理能力,在解决各种自然语言处理任务方面取得了显著进展。然而,大模型有着固有的局限性,因为它们不能访问最新的信息(存储在Web上或特定于任务的知识库中),不能使用外部工具,也不能执行精确的数学和逻辑推理。在本论文引入了Chameleon,这是一个可插入式的组合推理框架,利用LLMs合成程序并为各种任务组合各种工具。与现有的工具增强型LLMs不同,Chameleon使用更丰富的工具集,包括LLMs、现成的视觉模型、网络搜索引擎、Python函数和基于启发式的模块。此外,Chameleon利用LLMs的上下文学习能力,构建在LLM上作为自然语言规划器,无需任何训练或精心策划的规则。在每个模块的执行期间,该模块处理查询和缓存的上下文,返回由模块自身确定的结果,并更新查询和上下文以供后续执行。最后,系统将模块组合成一个顺序程序,允许后续模块利用先前缓存的上下文和更新的查询。

关键词: LLMs; 大模型; 大模型增强; 组合工具; 自然语言处理

1 引言

GPT-3、PaLM、LLaMA、ChatGPT以及最近开发的GPT-4,在各种自然语言处理任务中取得显著进展。它们能够以零样本学习的方式解决各种任务,或者在有一些示例的情况下完成任务,并在规划和决策方面显示出类似于人类的巨大潜力。尽管具有这些能力,LLMs也面临一些固有的限制,比如无法获取最新信息、执行精确的数学推理或使用专门的模型。因此,为了解决这些缺点,通过赋予当前LLMs自动组合外部工具以解决实际任务的能力变得至关重要。

2 相关工作

2.1 组合推理

神经模块和组合方法已被探索用于自动执行所需的子任务分解,增强各种推理任务的可解释性和适应性。早期的研究 [?] [?]认为复杂的推理任务基本上是组成的,并提出了神经模块网络(NMN)将其分解为子任务。然而,这些方法依赖于脆弱的现成解析器,并且受到模块配置的限制。后来的一些工作 [?] [?] [?]进一步通过端到端方式预测实例特定的网络布局,而不依赖于解析器,使用强化学习 [?]和弱监督学习。在视觉推理中,已经提出了包含程序生成器和执行引擎的模型,以结合深度表示学习和符号程序执行 [?] [?]。在数学推理领域,一个可解释的求解器已经被开发出来,它将定理知识作为条件规则,并逐步进行符号推理 [?]

2.2 工具增强语言模型

近年来,大型语言模型(large language models, llm) [?] [?] [?]的发展取得了巨大进展,并刺激了提示学习和指导学习的研究。尽管大型语言模型的表现令人印象深刻,但它们存在固有的局限性,例如无法获取最新信息 [?],无法利用外部工具 [?],无法进行精确的数学推理 [?].最近的基准,如ScienceQA和TabMWP,已经出现,以评估大型语言模型解决复杂推理挑战的能力,特别是那些强调使用外部工具的能力。同时,人们对利用外部工具和模块化方法来增强大型语言模型的兴趣也越来越大。这些增强的大型语言模型可以在网络搜索引擎的帮助下访问实时信息,并利用外部资源中的特定领域知识。一些工作利用Python解释器生成复杂的程序,利用强大的计算资源,更有效地执行逻辑推理任务。例如,Toolformer [?]构建了使用工具的增强数据来训练语言模型,以选择五种工具。在可视化工具领域,已经提出了各种方法来增强大型语言模型处理视觉任务的能力,并通过Hugging Face模型, Azure模型,视觉基础模型进行了增强。

3 本文方法

3.1 本文方法概述

系统架构如图 1所示。Chameleon由一个模块库和一个基于LLM的规划器组成,模块库定义了不同类型的工具,而规划器的目标是将原始问题分解成可以通过特定任务工具有效解决的子任务。模块库具有多种工具类型,如表 2所示,使Chameleon能够展示各种推理能力,包括图像理解、知识检索、网络搜索、复杂的数学推理和表格理解。规划器被形式化如下:给定输入查询 x_0 ,模块库 M 和约束 G ,自然语言规划器 P 选择一组可以按顺序执行的模块,以通过生成类似自然语言的程序回答查询。在规划器中,相应的模块将按顺序执行,最终结果包含缓存和当前输出。

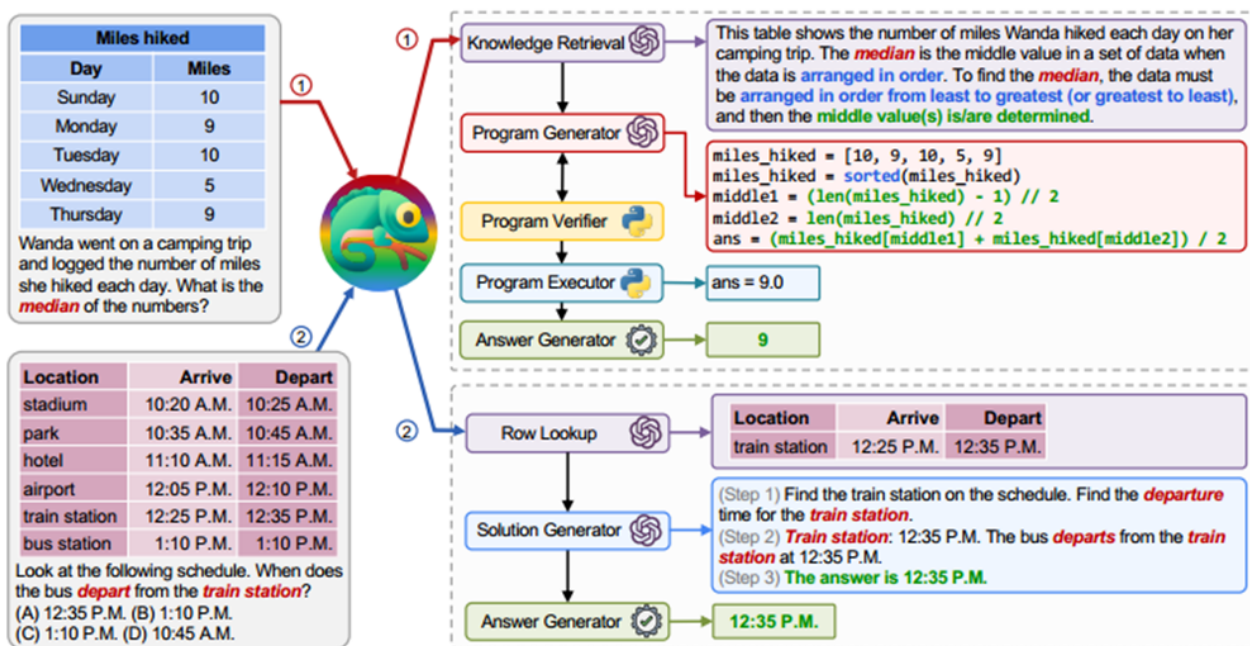


图 1. 系统架构图







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

图 2. 模块库工具类型

4 复现细节

本次主要复现了论文模块库部分，它包括以下部分：知识检索（Knowledge Retrieval）、网络搜索（Bing Search）、图像生成说明（Image Captioner）、识别图像文本（Text Detector）、表格简化（Row Lookup and Column Lookup）、表格翻译（Table Verbalizer）、输入查询（Solution Generator）、输出规范（Answer Generator）。

4.1 Knowledge Retrieval

如图 3，该模块用于检索对解决复杂问题至关重要的额外背景知识。它适用于一些专业领域，如科学和数学，为任务提供上下文，帮助模型做出更好的回答。

```

def knowledge_retrieval(self):
    # 检索例子
    question_text = self.get_question_text()
    metadata = self.get_metadata()
    response = self.cache["response"] if "response" in self.cache else ""

    # 构建prompt
    demo_prompt = prompt_kr.prompt.strip() # demo prompt
    if response != "":
        test_prompt = f"Question: {question_text}\n\nMetadata: {metadata}\n\n{response}\n\nKnowledge:\n"
    else:
        test_prompt = f"Question: {question_text}\n\nMetadata: {metadata}\n\nKnowledge:\n" # test prompt
    full_prompt = demo_prompt + "\n\n" + test_prompt # full prompt

    messages=[
        {"role": "user", "content": full_prompt},
    ]

    # 调chatGPT
    knowledge = get_chat_response(messages, self.api_key, self.kr_engine, self.kr_temperature, self.kr_max_tokens)

    # 更新缓存
    if knowledge != "" and knowledge != None:
        response += f"\n\nKnowledge:\n{knowledge}"
        response = response.strip()

    # 更新缓存
    self.cache["response"] = response
    self.cache["knowledge_retrieval:input"] = test_prompt
    self.cache["knowledge_retrieval:output"] = knowledge
    return test_prompt, knowledge

```

图 3. Knowledge Retrieval

4.2 Bing Search

该模块使用搜索引擎API，该模块基于输入查询返回相关的搜索结果，然后由后续模块解析并使用，以从多样化的来源中收集更丰富的上下文信息，增强问题解决的效果。如图 4、5和 6，和论文一样使用的是Bing的api。原本想更换为百度的api，但是搜索结果不如Bing，还有可能返回广告，因此弃用。

```

def call_bing_search(endpoint, bing_api_key, query, count):
    headers = {'Ocp-Apim-Subscription-Key': bing_api_key}
    params = {"q": query, "textDecorations": True,
              "textFormat": "HTML", "count": count, "mkt": "en-GB"}
    try:
        server = _validate_server(endpoint) # server address
        server_response = requests.get(server, headers=headers, params=params)
        resp_status = server_response.status_code
        if resp_status == 200:
            result = server_response.json()
            return result
    except:
        pass

    return None

```

图 4. Bing Search 1

```

def bing_search(self):
    # 获取输入
    endpoint = self.endpoint
    count = self.search_count
    query = self.cache["query"] if "query" in self.cache else None
    response = self.cache["response"] if "response" in self.cache else ""

    # 调用bing API key进行搜索, 类似爬虫, 并构造回答
    if query != None and query != "":
        result = call_bing_search(endpoint, bing_api_key, query, count)
    else:
        result = None
    responses = parse_bing_result(result)

    if len(responses) > 0 and responses[0] != "":
        response += f"\n\nBing search response: {responses}"
        response = response.strip()

    # 更新缓存
    self.cache["response"] = response
    self.cache["bing_search:input"] = query
    self.cache["bing_search:output"] = responses
    return query, responses

```

图 5. Bing Search 2

```

def parse_bing_result(result):
    responses = []
    try:
        value = result["webPages"]["value"]
    except:
        return responses

    for i in range(len(value)):
        snippet = value[i]['snippet'] if 'snippet' in value[i] else ""
        snippet = snippet.replace(_old: "<b>", _new: "").replace(_old: "</b>", _new: "").strip()
        if snippet != "":
            responses.append(snippet)

    return responses

```

图 6. Bing Search 3

4.3 Image Captioner

该模块为图像生成说明，为查询提供关键的补充上下文。源代码使用了预训练模型，本次复现时改成调用openAI的接口，将视觉数据转换为语言，有助于有效理解和推理图像内容。如图 7和 8。

```

def image_captioner(self):
    # 获取图像输入
    image_file = self.cache["example"]["image_file"]
    response = self.cache["response"] if "response" in self.cache else ""

    # 解析图像
    if "caption" in self.cache["example"]:
        caption = self.cache["example"]["caption"]
    else:
        if not os.path.exists(image_file):
            caption = ""
        else:
            caption = self.analyse_image(image_file)

    # 更新缓存
    if caption != "":
        response += f"\n\nImage caption: {caption}"
        response = response.strip()

    # 更新缓存
    self.cache["response"] = response
    self.cache["image_captioner:input"] = image_file
    self.cache["image_captioner:output"] = caption
    return image_file, caption

```

图 7. Image Captioner 1

```

def analyse_image(self, image_file):

    client = openai.ClientSession()

    response = client.chat.completions.create(
        model="gpt-4-vision-preview",
        messages=[
            {
                "role": "user",
                "content": [
                    {"type": "text", "text": "What's in this image?"},
                    {
                        "type": "image_url",
                        "image_url": {image_file},
                    },
                ],
            },
        ],
        max_tokens=300,
    )

    return response.choices[0]

```

图 8. Image Captioner 2

4.4 Text Detector

模块设计用于识别给定图像中的文本。通常，当问题要求从包含图表、图表、表格、地图或其他视觉元素的图像中提取文本信息时，使用“Text Detector”模块。通过有效检测各种格式的文本，此模块有助于分析和理解基于图像的内容。源代码使用了OCR识别图片文本信息，本次复现改为调用openAI接口，与前一个模块的区别主要在于prompt的不同，如图 9和 10。

```

def text_detector(self):
    # 获取图像输入
    image_file = self.cache["example"]["image_file"]
    response = self.cache["response"] if "response" in self.cache else ""

    # 解析图像
    texts = []
    if "ocr" in self.cache["example"]:
        try:
            ocr = eval(self.cache["example"]["ocr"])
            if len(ocr) > 0:
                texts = [(t[0], t[1]) for t in ocr] # (coordiantes, text)
        except:
            pass
    else:
        if not os.path.exists(image_file):
            texts = []
        else:
            texts = self.analyse_text_in_image(image_file)

    # 更新缓存
    if len(texts) > 0:
        response += f"\n\nDetected text in the image: {texts}"
        response = response.strip()

    # 更新缓存
    self.cache["response"] = response
    self.cache["text_detector:input"] = image_file
    self.cache["text_detector:output"] = texts
    return image_file, texts

```

图 9. Text Detector 1


```

def analyse_text_in_image(self, image_file):

    client = openai.ClientSession()

    response = client.chat.completions.create(
        model="gpt-4-vision-preview",
        messages=[
            {
                "role": "user",
                "content": [
                    {"type": "text", "text": "What text is in this image?"},
                    {
                        "type": "image_url",
                        "image_url": {image_file},
                    },
                ],
            }
        ],
        max_tokens=300,
    )

    return response.choices[0]

```

图 10. Text Detector 2

4.5 Row Lookup and Column Lookup

当查询涉及表格上下文时，大表格可能会分散系统的注意力，因此“Row Lookup”简化表格，仅保留与查询相关的行。如果所有行都相关，则返回原始表格。同样的，“Column Lookup”模块通过关注相关列来处理涉及表格上下文的问题。它简化表格，仅保留相关的列，如果所有列都相关，则返回原始表格。如图 11。

```

def row_lookup(self):
    # 获取查询输入
    question = self.cache["example"]["question"]
    table = self.cache["example"]["table"]
    context = self.cache["example"]["context"] if "context" in self.cache["example"] else ""
    row_num = self.cache["example"]["row_num"]
    column_num = self.cache["example"]["column_num"]
    cell_num = row_num * column_num

    if row_num <= self.rl_row_threshold or cell_num <= self.rl_cell_threshold:
        test_prompt = None
        simple_table = table

    # 构造简化表格
    else:
        demo_prompt = prompt_rl.prompt.strip()
        if context != "":
            test_prompt = f"Question: {question}\n\nTable:\n{table}\n\n(context):\n\nSimplified Table:\n"
        else:
            test_prompt = f"Question: {question}\n\nTable:\n{table}\n\nSimplified Table:\n"

        full_prompt = demo_prompt + "\n\n" + test_prompt
        messages=[
            {"role": "user", "content": full_prompt},
        ]
        # 调chatGPT
        if self.rl_cand == 1:
            simple_table = get_chat_response(messages, self.api_key, self.rl_engine, self.rl_temperature, self.rl_max_tokens)
        else:
            simple_tables = get_chat_response(messages, self.api_key, self.rl_engine, self.rl_temperature, self.rl_max_tokens, self.rl_cand)
            simple_table = max(simple_tables, key=simple_tables.count)

    # 更新缓存
    self.cache["example"]["table"] = simple_table
    self.cache["row_lookup:input"] = test_prompt
    self.cache["row_lookup:output"] = simple_table
    return test_prompt, simple_table

```

图 11. Row Lookup and Column Lookup

4.6 Table Verbalizer

如图 12，将表格翻译成易于理解的描述。

```

def table_verbalizer(self):
    # 获取表格输入
    question = self.cache["example"]["question"]
    table = self.cache["example"]["table"]
    context = self.cache["example"]["context"] if "context" in self.cache["example"] else ""

    # 构造翻译表格的prompt
    demo_prompt = prompt_tv.prompt.strip()
    if context != "":
        test_prompt = f"Question: {question}\n\nTable: \n\n{table}\n\n{context}\n\nTable description: \n"
    else:
        test_prompt = f"Question: {question}\n\nTable: \n\n{table}\n\nTable description: \n"
    full_prompt = demo_prompt + "\n\n" + test_prompt
    messages=[
        {"role": "user", "content": full_prompt},
    ]

    # 执行
    verbalization = get_chat_response(messages, self.api_key, self.tv_engine, self.tv_temperature, self.tv_max_tokens)
    context += f"\n\nTable description: {verbalization}".strip()

    # 更新缓存
    self.cache["example"]["context"] = context
    self.cache["table_verbalizer:input"] = test_prompt
    self.cache["table_verbalizer:output"] = verbalization
    return test_prompt, verbalization

```

图 12. Table Verbalizer

4.7 Solution Generator

该模块使用所有缓存信息生成对输入查询的详细解决方案。采用了链式思维的提示方法，确保连贯且结构良好的响应。如果能够独立解决查询，特别是对于更简单的查询，规划器可以直接使用此模块，而不是使用其他功能模块。如图 13。

```

def solution_generator(self):
    # get the module input
    test_prompt, full_prompt = self.build_prompt_for_sg()
    messages=[
        {"role": "user", "content": full_prompt},
    ]

    # excute the module
    success = False
    patience = self.sg_patience
    count = 0
    while count < patience and not success:
        if self.sg_temperature < 0.1 and count > 0:
            _temperature = min(self.sg_temperature + 0.1, 1.0)
        else:
            _temperature = self.sg_temperature

        solution = get_chat_response(messages, self.api_key, self.sg_engine, _temperature, self.sg_max_tokens)

        pattern = re.compile(r"The answer is ([\s\S]+).$") # "The answer is XXXXX.",
        res = pattern.findall(solution)
        if len(res) > 0:
            success = True
        count += 1

    # update the cache
    self.cache["solution_generator:input"] = test_prompt
    self.cache["solution_generator:output"] = solution
    return test_prompt, solution

```

图 13. Solution Generator 1

4.8 Answer Generator

如图 14。该模块从结果中提取和规范化答案。

```

def answer_generator(self):
    # 获取输入
    output = self.cache["solution"]
    options = self.cache["example"]["choices"]
    inds = ["A", "B", "C", "D", "E"]

    # 进行规范化
    success = False
    if output:
        pattern = re.compile(r"[Tt]he answer is ([A-Z])") # "The answer is A.",
        res = pattern.findall(output)
        if len(res) > 0:
            ans = res[0] # "A"
            if ans in inds[:len(options)]:
                success = True
                prediction = options[inds.index(ans)]

    if not success:
        prediction = normalize_prediction_scienceqa(output, options)

    # 更新缓存
    self.cache["prediction"] = prediction
    self.cache["answer_generator:input"] = output
    self.cache["answer_generator:output"] = prediction
    return output, prediction

```

图 14. Answer Generator 1

4.9 与已有开源代码对比

本次所复现的论文开源了它的代码，我参考了其代码并进行复现和修改，主要复现了模块库部分，实现任务比较偏工程化。本次复现不仅完全实现了论文的功能，还对图生文部分，即为图像生成说明和识别给定图像文本这两个任务做了优化。在源代码中，作者分别使用了预训练模型和OCR完成这两个任务，而我本次复现采用调用网络接口的形式，即调用Gpt自己的识别图片接口，将图片文件交给大模型分析，避免了预训练模型的步骤，使得我们复现的代码运行占用更少的资源，提高效率，还能得出差不多的结果。

4.10 实验环境搭建

本次复现只需python环境3.8，安装相关包即可。

4.11 界面分析与使用说明

首先需要在环境中配置openAI使用Gpt接口的key，以及使用Bing搜索接口的key，才可运行代码。如图 15所示，制定好数据集和结果路径，就可以开始运行。

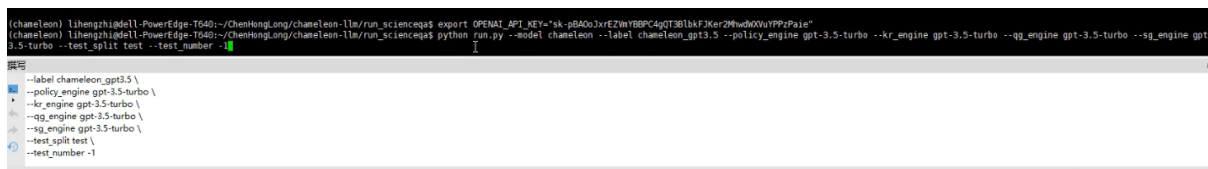


图 15. 操作界面示意

4.12 创新点

本次复现的创新点主要在于图像生成说明和识别给定图像文本这两个任务做了优化。在源代码中，作者原本使用了预训练模型和OCR完成这两个任务，而我本次复现调用OpenAi Gpt自己的接口，将图片文件交给大模型分析，避免了预训练模型的步骤，使得我们复现的代码运行不仅占用更少的资源，提高效率，还能得出差不多的结果。

5 实验结果分析

本次实验主要是对论文源码所给的ScienceQA数据集进行分析，通过判断系统的大语言模型回答正确与否进行判定，最后计算正确或错误的回答次数，以及相应的百分比。ScienceQA是一个涉及多种上下文格式和各种科学主题的多模态问题回答基准测试，由于Gpt3.5 token收费昂贵，本次实验只用了数据集10%进行测试。测试结果如图 16和 17所示。在源代码中，图 16第一张图展现了源代码回答准确率为76.53%，而我复现的回答准确率为75.94%；图 17，显示了问题集的数量为4241个问题，源代码回答对了其中的3670个问题，而复现代码回答对了3580个问题。

```
Result file:
../results/scienceqa/chameleon_chatgpt_test_cache.jsonl
number of questions: 4241

acc_average:    79.93
acc_nat:        81.62
acc_sol:        70.64
acc_lan:        84.00
acc_txt:        79.77
acc_img:        70.80
acc_no:         86.62
acc_grade_1_6:  81.86
acc_grade_7_12: 76.53

& 79.93 & 81.62 & 70.64 & 84.00 & 79.77 & 70.80 & 86.62 & 81.86 & 76.53 \\
```

```
Result file:
../results/scienceqa/chameleon_chatgpt_test_cache.jsonl
number of questions: 4241

acc_average:    79.06
acc_nat:        80.95
acc_sol:        69.07
acc_lan:        83.27
acc_txt:        79.08
acc_img:        69.71
acc_no:         85.99
acc_grade_1_6:  80.83
acc_grade_7_12: 75.94

& 79.06 & 80.95 & 69.07 & 83.27 & 79.08 & 69.71 & 85.99 & 80.83 & 75.94 \\
```

图 16. 实验结果示意

```

None
None
====Input Arguments====
{
  "data_root": "../data/scienceqa",
  "output_root": "../results",
  "model": "chameleon",
  "label": "chameleon_gpt4",
  "task_name": "scienceqa",
  "test_split": "test",
  "test_number": -1,
  "seed": 0,
  "modules": null,
  "policy_engine": "gpt-3.5-turbo",
  "policy_temperature": 0.0,
  "policy_max_tokens": 128,
  "kr_engine": "gpt-3.5-turbo",
  "kr_temperature": 0.0,
  "kr_max_tokens": 512,
  "qg_engine": "gpt-3.5-turbo",
  "qg_temperature": 0.0,
  "qg_max_tokens": 64,
  "qg_patience": 5,
  "bing_file": "../data/scienceqa/bing_responses.json",
  "endpoint": "https://api.bing.microsoft.com/v7.0/search",
  "search_count": 1,
  "use_caption": false,
  "caption_file": "../data/scienceqa/captions.json",
  "ocr_file": "../data/scienceqa/ocrs.json",
  "sg_engine": "gpt-3.5-turbo",
  "sg_temperature": 0.0,
  "sg_max_tokens": 512,
  "sg_patience": 5,
  "debug": false
}
# Number of test examples: 4241
../results/scienceqa/chameleon_gpt4_test.json
Result file exists: ../results/scienceqa/chameleon_gpt4_test.json
Count: 4241, Correct: 3670, Wrong: 571
0it [00:00, ?it/s]

sk-pBA0oJxrEZVnYBBPC4gQT3B1bkFJKer2MhwdwXVuYPPzPaie
None
====Input Arguments====
{
  "data_root": "../data/scienceqa",
  "output_root": "../results",
  "model": "chameleon",
  "label": "chameleon_gpt4",
  "task_name": "scienceqa",
  "test_split": "test",
  "test_number": -1,
  "seed": 0,
  "modules": null,
  "policy_engine": "gpt-3.5-turbo",
  "policy_temperature": 0.0,
  "policy_max_tokens": 128,
  "kr_engine": "gpt-3.5-turbo",
  "kr_temperature": 0.0,
  "kr_max_tokens": 512,
  "qg_engine": "gpt-3.5-turbo",
  "qg_temperature": 0.0,
  "qg_max_tokens": 64,
  "qg_patience": 5,
  "bing_file": "../data/scienceqa/bing_responses.json",
  "endpoint": "https://api.bing.microsoft.com/v7.0/search",
  "search_count": 1,
  "use_caption": false,
  "caption_file": "../data/scienceqa/captions.json",
  "ocr_file": "../data/scienceqa/ocrs.json",
  "sg_engine": "gpt-3.5-turbo",
  "sg_temperature": 0.0,
  "sg_max_tokens": 512,
  "sg_patience": 5,
  "debug": false
}
# Number of test examples: 4241
../results/scienceqa/chameleon_gpt4_test.json
Result file exists: ../results/scienceqa/chameleon_gpt4_test.json
Count: 4241, Correct: 3580, Wrong: 661
0it [00:00, ?it/s]

```

图 17. 实验结果示意

结果验证了使用OpenAI的识别图片接口来加强大模型识别图片能力，包括为图像生成说明和识别给定图像文本这两个任务，具有可行性。

6 总结与展望

总之，我们介绍了一种新的即插即用组合推理框架Chameleon，它通过以即插即用的方式使用外部工具对当前大型语言模型进行扩展，从而解决了它们的局限性。我们的方法采用了多种工具，并在两个具有挑战性的基准(ScienceQA和TabMWP)上展示了令人印象深刻的适应性和有效性。通过在现有的最先进的模型上取得显著的精度改进，Chameleon展示了它在解决各种领域的实际查询方面的潜力。