第 5 章 使用 LangChain 工具进行文档查询

在当今快节奏的商业和研究环境中，跟上不断增长的信息量可能是一项艰巨的任务。对于计算机科学和人工智能等领域的工程师和研究人员来讲，以及时了解最新发展至关重要，然而，阅读和理解大量论文可能既费时又费力。这就是自动化发挥作用的地方。在本章中将介绍一种自动总结研究论文和回答问题的方法，使研究人员更容易消化和了解情况。通过利用语言模型和一系列问题将开发的摘要以简洁和简化的格式总结论文的核心断言、含义和机制。这不仅可以在研究课题时节省时间和精力，还可以确保我们具有有效地驾驭科学进步的能力。我们还将介绍 OpenAI 模型中的函数，以及它们在信息提取中的应用。读者将看到它们如何作为简历(CV)的解析器在应用程序中工作。这个函数的语法是特定于 OpenAI 的 API 的，并且有许多应用程序，但是，LangChain 提供了一个平台，允许为任何 LLM 创建工具，从而增强其功能。这些工具使 LLM 能够与 API 交互、访问实时信息并执行各种任务，例如检索搜索、数据库查询、编写电子邮件，甚至拨打电话。本章将使用检索增强生成(Retrieval Augmented Generation，RAG)实现问答应用。这是一种通过将相关数据注入上下文来更新 LLM 的技术，例如 GPT。最后本章将讨论智能体的不同决策策略，其中将实施两种策略，第 1 种策略为计划和执行(或计划和解决)，第 2 种策略为一次性 Agent，并且它们将被集成到可视化界面中，作为浏览器中的可视化应用程序(使用 Streamlit)进行问答。

5.1 幻觉现象

GPT-3、Llama 和 Claude 2 等 LLM 的快速发展引起了人们对其局限性和潜在风险的关注，其中的一个主要问题是幻觉现象，即模型生成的输出是无意义的、不连贯的或不忠实于所提供的输入。幻觉现象在实际应用中会带来性能和安全风险，例如医疗或机器翻译。幻觉现象还有另一个方面，即 LLM 生成的文本包含敏感的个人信息，例如电子邮件地址、电话号码和实际地址。这带来了重大的隐私问题，因为它表明语言模型可以从其训练语料库中记忆和恢复此类私有数据，尽管这些数据不存在于源输入中。

LLM 上下文中的幻觉现象是指生成的文本不忠实于预期内容或荒谬的现象。这个术语与心理幻觉相提并论，因为幻觉涉及感知不存在的东西。在 NLP 中，幻觉文本可能在提供的上下文中显得流畅和自然，但缺乏特异性或可验证性。相反幻觉的对立面是忠实性，即生成的内容与来源保持一致和真实。

当生成的输出与源内容相矛盾时，就会出现内在幻觉现象，而外在幻觉现象涉及生成源材料无法验证或支持的信息。外在幻觉现象有时可以包括事实正确的外部信息，但从事实安全的角度来看，它们的不可验证性引起了人们的担忧。目前，解决幻觉问题的努力正在进行中，但需要全面了解不同的任务，以制定有效的缓解方法。LLM 中的幻觉现象可能由以下多种因素引起：

（1）编码器的表示学习不完善。

（2）错误的解码，包括处理源输入的错误部分和解码策略的选择。

（3）暴露偏差，即训练和推理时间之间的差异。

（4）参数化知识偏差，即预训练模型将自己的知识优先于输入，从而导致产生过多的信息。

解决 LLM 出现幻觉现象的方法可以分为两类：数据相关方法、建模和推理方法。

1. 数据相关方法

构建忠实的数据集：从头开始构建具有干净忠实目标的数据集，或重写真实句子，同时确保语义一致性。

自动清理数据：识别和过滤现有语料库中不相关或矛盾的信息，以减少语义干扰。

信息增强：使用外部信息（例如检索到的知识或合成数据）增强输入，以提高语义理解并解决源-目标差异。

2. 建模和推理方法

架构：修改编码器架构以增强语义解释，修改注意力机制以优先处理源信息，修改解码器结构以减少幻觉并强制执行隐式或显式约束。

培训：结合计划、强化学习 、多任务学习和可控生成技术，通过改善对齐、优化奖励功能及平衡忠诚度和多样性来减轻产生幻觉现象。

后处理：通过生成后优化策略来纠正输出中出现的幻觉现象，或者使用后处理校正方法专门优化结果以确保忠实度。

LLM 出现幻觉现象的后果是传播错误信息或出于政治目的滥用的危险。错误信息包括虚假信息、欺骗性新闻和谣言，对社会构成重大威胁，尤其是在内容创作和通过社交媒体传播的便利性方面。对社会的威胁包括对科学的不信任、公共卫生叙事、社会两极分化和民主进程。新闻学和档案研究对这个问题进行了广泛研究，事实核查举措也随之增加。致力于事实核查的组织为独立事实核查员和记者提供培训和资源，从而扩大专家事实核查工作的规模。解决错误信息对于维护信息的完整性和打击其对社会的不利影响至关重要。在文献中，这种问题被称为文本蕴涵，其中包括模型预测文本对之间的方向真值关系。在本章中将重点介绍通过信息增强和后处理进行自动事实核查。可以从 LLM 或使用外部工具检索

事实。在前一种情况下，预先训练的语言模型可以取代知识库和检索模块，利用其丰富的知识来回答开放领域的问题，并使用提示来检索特定事实。大致的思路如图 5-1 所示[13]。

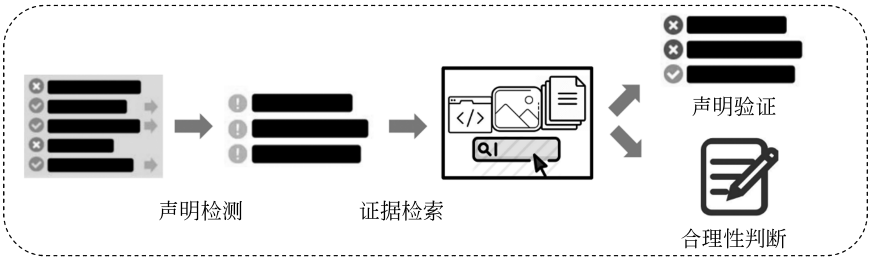

图 5-1　自动事实核查管道的 3 种状态

从图 5-1 可以区分 3 个阶段。

(1) 声明检测：识别需要验证的声明。

(2) 证据检索：检索证据以查找支持或反驳主张的来源。

(3) 声明验证：根据证据评估声明的真实性。

从 2018 年发布的 24 层 BERT-Large 模型开始，LLM 已经在维基百科等大型知识库上进行了预训练，因此它们能够回答来自维基百科的知识问题。例如，为了回答“微软的总部在哪里”这个问题，该问题将被重写为“微软的总部在[MASK]”，并输入语言模型中作为答案。这种方法有趣的是，如果未接收源文本的 LLM 在生成目标时产生的损失小于接收源文本的 LLM，则表明生成的 Token 是幻觉的。幻觉 Token 与目标 Token 总数的比率可以作为生成输出中幻觉程度的衡量标准。在 LangChain 中有一个链可用于通过提示链接进行事实检查，在其中模型主动质疑语句中的假设。在这个名为 LLMCheckerChain 的自我检查链中，模型被按顺序进行提示，代码如下：

```
Here's a statement: {statement}\nMake a bullet point list of the assumptions you
made when producing the above statement.\n
```

在这里读者应注意，这是一个字符串模板，其中花括号中的元素将被变量替换。接下来，这些假设被反馈给模型，以便通过如下提示逐一检查它们，代码如下：

```
Here is a bullet point list of assertions:
    {assertions}
    For each assertion, determine whether it is true or false. If it is false,
explain why.\n\n
```

最后，模型的任务是做出最终判断，输出的结果如下：

```
In light of the above facts, how would you answer the question '{question}'
```

LLMCheckerChain 自行完成这一切，代码如下：

```
//第 5 章/ LLMCheckerChain.py
from langchain.chains import LLMCheckerChain
from langchain.llms import OpenAI
llm = OpenAI(temperature=0.7)
text = "What type of mammal lays the biggest eggs?"
checker_chain = LLMCheckerChain.from_llm(llm, verbose=True)
checker_chain.run(text)
```

LLM 可以对此问题返回不同的结果，其中一些是错误的，而另一些则被正确地识别为错误。本文在下面的示例中得到了蓝鲸、北美海狸或已灭绝的巨型恐鸟等结果，结果如下：

```
//第 5 章/ LLMCheckerChainOut.py
Monotremes, a type of mammal found in Australia and parts of New Guinea, lay the
largest eggs in the mammalian world. The eggs of the American echidna (spiny
anteater) can grow as large as 10 cm in length, and dunnarts (mouse-sized
marsupials found in Australia) can have eggs that exceed 5 cm in length.
• Monotremes can be found in Australia and New Guinea
• The largest eggs in the mammalian world are laid by monotremes
• The American echidna lays eggs that can grow to 10 cm in length
• Dunnarts lay eggs that can exceed 5 cm in length
• Monotremes can be found in Australia and New Guinea - True
• The largest eggs in the mammalian world are laid by monotremes - True
• The American echidna lays eggs that can grow to 10 cm in length - False, the
American echidna lays eggs that are usually between 1 to 4 cm in length.
• Dunnarts lay eggs that can exceed 5 cm in length - False, dunnarts lay eggs that
are typically between 2 to 3 cm in length.
The largest eggs in the mammalian world are laid by monotremes, which can be found
in Australia and New Guinea. Monotreme eggs can grow to 10 cm in length.
> Finished chain.
```

虽然这并不能保证答案是正确的，但它可以阻止一些不正确的结果。事实核查方法涉及将声明分解为更小的可检查查询，这些查询可以表述为问答任务。专门为搜索领域数据集而设计的工具可以帮助事实核查人员有效地寻找证据。谷歌和必应等现成的搜索引擎还可以检索与主题和证据相关的内容，以准确捕捉陈述的真实性。在接下来的部分中将应用此方法根据 Web 搜索和其他工具返回结果。具体来讲将实现一个链来汇总文档，用户可以从这些文档中提出任何问题并得到答案。

5.2 文档总结

本节将讨论自动化总结长文本和研究论文的过程。在当今快节奏的商业和研究环境中，跟上不断增长的信息量可能是一项艰巨的任务。对于计算机科学和人工智能等领域的工程师和研究人员来讲，以及时了解最新发展至关重要，然而，阅读和理解大量论文可能既

费时又费力。这就是自动化发挥作用的地方。工程师受到他们构建和创新的愿望的驱使，通过创建管道和流程来自动化执行重复性任务，从而避免自己执行重复性任务。这种方法经常被误认为是懒惰，它使工程师能够专注于更复杂的挑战，并更有效地利用他们的技能。接下来将构建一个自动化工具，该工具可以以更易于理解的格式快速总结长文本的内容。该工具旨在帮助研究人员跟上每天发表的论文，特别是在人工智能等快速发展的领域。通过自动化总结过程，研究人员可以节省时间和精力，同时确保他们随时了解各自领域的最新发展。该工具将基于 LangChain，并利用 LLM 以更简洁和简化的方式总结论文的核心断言、含义和机制。它还可以回答有关论文的具体问题，使其成为文献综述和加速科学研究的宝贵资源。

该工具还可以被进一步开发，以允许自动处理多个文档并针对特定研究领域进行定制。总体而言，该方法旨在通过提供一种更有效、更易于访问的方式来了解最新研究，从而使研究人员受益。LangChain 支持使用 LLM 处理文档的 mapreduce 方法，从而可以有效地处理和分析文档。当读取大文本并将它们拆分为适合 LLM 的标记上下文长度的文档（块）时，可以单独对每个文档应用一个链，然后将输出组合到单个文档中。mapreduce 过程包括以下两个步骤。

（1）map 步骤：LLM 链单独应用于每个文档并将输出视为新文档。

（2）reduce 步骤：所有新文档都被传递到单独的合并文档链以获得单个输出。

这种方法允许并行处理文档，并允许使用 LLM 来推理、生成或分析单个文档及组合它们的输出。该过程的机制涉及压缩或折叠映射的文档，以确保它们适合组合文档链，这可能还涉及使用 LLM。如果需要，则可以用递归方式执行压缩步骤。以下是加载 PDF 文档并对其进行总结的简单示例，代码如下：

```
//第 5 章/ pdfLoad.py
from langchain.chains.summarize import load_summarize_chain
from langchain import OpenAI
from langchain.document_loaders import PyPDFLoader
pdf_loader = PyPDFLoader(pdf_file_path)
docs = pdf_loader.load_and_split()
llm = OpenAI()
chain = load_summarize_chain(llm, chain_type="map_reduce")
chain.run(docs)
```

变量 pdf_file_path 是带有 PDF 文件路径的字符串。map 和 reduce 步骤的默认提示如下：

```
Write a concise summary of the following:
{text}
CONCISE SUMMARY:
```

用户可以为每个步骤指定任何提示。在 LangChainHub 上可以看到 qa-withsources 提

示符，它采用如下的 reduce/combine 提示符：

```
Given the following extracted parts of a long document and a question, create a
final answer with references (\"SOURCES\"). \nIf you don't know the answer, just
say that you don't know. Don't try to make up an answer.\nALWAYS return a \"SOURCES\"
part in your answer.\n\nQUESTION: {question}\n==========\nContent: {text}
```

在这个提示中提出了一个具体的问题，但同样用户可以给 LLM 一个更抽象的指令来提取假设和含义。文本将是 map 步骤的摘要。这样的指示将有助于防止出现幻觉现象，其他说明示例可能是将文档翻译成不同的语言。一旦开发者开始进行大量调用，尤其是在 map 步骤中，成本就会增加。这里进行了大量调用并使用了很多 Token。这里有必要介绍关于 Token 的使用量的计算情况。

用户在使用 LLM 时，尤其是在进行长循环（例如使用映射操作）操作时，跟踪 Token 使用情况并了解花费了多少金钱非常重要。用户需要了解不同 LLM 的功能、定价选项和用例。OpenAI 提供了不同的模型，即 GPT-4、ChatGPT 和 InstructGPT，以满足各种自然语言处理需求。GPT-4 是一种强大的语言模型，适用于解决自然语言处理的复杂问题。它根据所用 Token 的大小和数量提供灵活的定价选项。ChatGPT 模型，如 GPT-3.5-Turbo，专注于聊天机器人和虚拟助手等对话应用程序。它们擅长准确和流畅地生成响应。ChatGPT 模型的定价基于使用的 Token 数量。InstructGPT 模型专为单圈指令跟踪而设计，并针对快速准确的响应生成进行了优化。InstructGPT 系列中的不同型号，例如 Ada 和 Davinci，提供了不同级别的速度和功能。Ada 是最快的模型，适用于速度至关重要的应用，而 Davinci 是最强大的模型，能够处理复杂的指令。InstructGPT 模型的定价取决于模型的功能，范围从 Ada 等低成本选项到 Davinci 等更昂贵的选项。DALL·E、Whisper 和 API 服务是 OpenAI 用于图像生成、语音转录、翻译和访问的 LLM。DALL·E 是一种 AI 驱动的图像生成模型，可以被无缝地集成到应用程序中，用于生成和编辑新颖的图像和艺术作品。OpenAI 提供了 3 层分辨率，允许用户选择他们需要的细节级别。分辨率越高，复杂性和细节就越多，而分辨率越低，表示效果越抽象，所以每幅图像的价格因分辨率而异。Whisper 是一种 AI 工具，可以将语音转录为文本并将多种语言翻译成英语。它有助于捕捉对话，促进交流，并提高跨语言的理解。Whisper 的使用费用基于每分钟费率进行计算。OpenAI 的 API 提供了对 GPT-3 等强大 LLM 的访问，这使开发人员能够创建高级应用程序。OpenAI 在注册 API 时会为用户提供初始 Token 使用限制，这表明了在特定时间范围内可用于与 LLM 交互的 Token 数。随着用户使用量的增加，OpenAI 可能会提高 Token 的使用限制，授予对模型的更多访问权限等。如果用户需要为其应用程序添加更多的 Token，则可以请求增加配额。用户可以通过连接到 OpenAI 回调来跟踪 OpenAI 平台中的 Token 使用情况，代码如下：

```
//第 5 章/ tokenUsage.py
with get_openai_callback() as cb:
    response = llm_chain.predict(text="Complete this text!")
```

```
    print(f"Total Tokens: {cb.total_tokens}")
    print(f"Prompt Tokens: {cb.prompt_tokens}")
    print(f"Completion Tokens: {cb.completion_tokens}")
    print(f"Total Cost (USD): ${cb.total_cost}")
```

在此示例中，带有 llm_chain 的代码行可以是 OpenAI 平台上的任何 LLM。用户应该能够看到输出里面包含成本和 Token 的使用量。还有另外两种方法可以获取 Token 的使用情况。作为 OpenAI 回调的替代，llm 类的 generate()方法会返回 LLMResult 类型的响应，而不是字符串。这包括 Token 的使用情况和完成原因，示例代码如下：

```
input_list = [
    {"product": "socks"},
    {"product": "computer"},
    {"product": "shoes"}
]
llm_chain.generate(input_list)
```

结果如下：

```
LLMResult(generations=[[Generation(text='\n\nSocktastic!', generation_info=
{'finish_reason': 'stop', 'logprobs': None})], [Generation(text='\n\nTechCore
Solutions.', generation_info={'finish_reason': 'stop', 'logprobs': None})],
[Generation(text='\n\nFootwear Factory.', generation_info={'finish_reason':
'stop', 'logprobs': None})]], llm_output={'token_usage': {'prompt_tokens': 36,
'total_tokens': 55, 'completion_tokens': 19}, 'model_name': 'text-davinci-003'})
```

最后，OpenAI API 中的聊天完成响应格式包含一个带有 Token 信息的使用对象：

```
  {
  "model": "gpt-3.5-turbo-0613",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 17,
    "prompt_tokens": 57,
    "total_tokens": 74
  }
}
```

接下来将介绍如何使用带有 LangChain 的 OpenAI 函数从文档中提取某些信息。

5.3　信息提取

2023 年 6 月，OpenAI 宣布对 OpenAI 的 API 进行更新，包括函数调用的新功能。开发人员现在可以向 gpt-4-0613 和 gpt-3.5-turbo-0613 模型描述函数，并让模型智能地生成一

个 JSON 对象，其中包含调用这些函数的参数。此功能旨在增强 GPT 模型与外部工具和 API 之间的联系，提供一种从模型中检索结构化数据的可靠方法。函数调用使开发人员能够创建可以使用外部工具或 OpenAI 插件回答问题的聊天机器人。它还允许将自然语言查询转换为 API 调用或数据库查询，并从文本中提取结构化数据。开发人员可以使用 JSON 模式向模型描述函数，并指定要调用的所需函数。在 LangChain 框架中，开发者可以在 OpenAI 中调用这些函数来进行信息提取或调用插件。对于信息提取可以使用 OpenAILLM 从文本中提取特定实体及其属性，以及从提取链的文档中提取特定实体及其属性。例如，这可以帮助识别文本中提到的人。通过 OpenAI 函数参数并指定架构，开发者可以确保模型输出具有正确的类型和属性。这种方法允许通过定义具有所需属性及其类型的架构来精确提取实体。它还允许指定哪些属性是必需的，哪些是可选的。模式的默认格式是字典，但开发者也可以在 Pydantic 中定义属性及其类型，从而在提取过程中提供控制性和灵活性。以下是一份简历中所需的信息架构的示例，代码如下：

```
//第 5 章/ cvSchema.py
from typing import Optional
from pydantic import BaseModel
class Experience(BaseModel):
    start_date: Optional[str]
    end_date: Optional[str]
    description: Optional[str]
class Study(Experience):
    degree: Optional[str]
    university: Optional[str]
    country: Optional[str]
    grade: Optional[str]
class WorkExperience(Experience):
    company: str
    job_title: str
class Resume(BaseModel):
    first_name: str
    last_name: str
    linkedin_url: Optional[str]
    email_address: Optional[str]
    nationality: Optional[str]
    skill: Optional[str]
    study: Optional[Study]
    work_experience: Optional[WorkExperience]
    hobby: Optional[str]
```

紧接着可以用它来从简历中提取信息。下面将尝试将一份简历内的信息读取出来，读者可以在这个网址找到这份简历的原版：https://github.com/xitanggg/open-resume。

利用 LangChain 中的 create_extraction_chain_pydantic()函数，开发者可以提供自己的

模板作为输入，输出将是遵循这个模板的实例化对象，代码如下：

```
//第 4 章/ pdfTemplate.py
from langchain.chains import create_extraction_chain_pydantic
from langchain.chat_models import ChatOpenAI
from langchain.document_loaders import PyPDFLoader
pdf_loader = PyPDFLoader(pdf_file_path)
docs = pdf_loader.load_and_split()
#用户需要注意并非所有模型都适用
llm = ChatOpenAI(model_name="gpt-3.5-turbo-0613")
chain = create_extraction_chain_pydantic(pydantic_schema=Resume, llm=llm)
return chain.run(docs)
```

输出的结果如下：

```
[Resume(first_name='John', last_name='Doe', linkedin_url='linkedin.com/in/
john-doe', email_address='hello@openresume.com', nationality=None, skill=
'React', study=None, work_experience=WorkExperience(start_date='May 2023',
end_date='Present', description='Lead a cross-functional team of 5 engineers in
developing a search bar, which enables thousands of daily active users to search
content across the entire platform. Create stunning home page product demo
animations that drives up sign up rate by 20%. Write clean code that is modular and
easy to maintain while ensuring 100% test coverage.', company='ABC Company',
job_title='Software Engineer'), hobby=None)]
```

这个结果并非完美，其中只有一种工作经验被解析出来，但考虑到迄今为止付出的很少的努力，这是一个良好的开端。更多功能可以被添加进去，例如让 LLM 根据简历猜测候选人的个性或领导能力。OpenAI 以某种语法将这些函数调用注入系统消息中，因为他们的模型经过针对性的优化。另外在 LangChain 框架中本身就具有将注入函数调用作为提示的功能。这意味着用户可以在 LLM 应用程序中使用 OpenAI 以外的模型进行函数调用。下面的章节将对此进行研究探讨，并尝试构建 Streamlit 框架下的交互式 Web 应用程序。

5.4　使用工具

由于 LLM 是在一般语料库数据上训练的，所以它对于需要特定领域知识的任务可能不那么有效。也是因为同样的原因，LLM 本身无法与环境交互并访问外部数据源，但是，LangChain 提供了平台，用于创建访问实时信息的工具，可以让 LLM 执行天气预报、预订、建议食谱等任务。Agent 和链框架内的工具让开发者能开发由具有数据感知和代理功能的 LLM 应用程序，并开辟了广泛的方法来解决 LLM 问题，扩展了它们的用例并使其更加通用和强大。工具的一个重要方面是它们能够在特定领域内工作或处理特定输入。例如，LLM 缺乏固有的数学能力，但是，像计算器这样的数学工具可以接受数学表达式或方程式

作为输入并计算结果。LLM 与这种数学工具相结合，进行计算并提供准确的答案。通常，这种检索方法和 LLM 的组合称为 RAG，它通过从外部来源检索相关数据并将其注入上下文来解决 LLM 的局限性问题。这些检索到的数据用作附加信息，以增强对 LLM 的提示。通过 RAG 将 LLM 与用例特定信息相结合，可以提高响应的质量和准确性。通过检索相关数据，RAG 有助于减少 LLM 的幻觉反应。例如，医疗保健应用程序中使用的 LLM 可以在推理过程中从外部来源（如医学文献或数据库）检索相关的医疗信息，然后可以将检索到的数据合并到上下文中，以增强生成的响应，并确保它们准确无误并与特定领域的知识保持一致。在此方案中实现 RAG 的好处是双重的。首先，它允许将最新信息合并到响应中（大家都知道目前 LLM 都有数据截止日期）。这确保了用户能够获得准确和相关的信息，即使是最近发生的事件或不断变化的主题，其次，RAG 通过利用外部信息来源增强了 ChatGPT 提供更详细和上下文答案的能力。通过从与特定主题相关的新闻文章或网站等来源检索特定上下文，LLM 的响应将更加准确。

RAG 的工作原理是从数据源中检索信息，以补充提供给语言模型的提示，为模型提供生成准确响应所需的上下文。RAG 涉及以下几个步骤。

（1）提示：用户向聊天机器人提供提示，描述他们对输出的期望。

（2）研究：执行上下文搜索并从各种数据源中检索相关信息。这可能涉及查询数据库、根据关键字搜索索引文档或调用 API 从外部源检索数据。

（3）更新资源：检索到的上下文将被注入原始提示中，并使用与用户查询相关的其他事实信息对其进行扩充。此增强的提示提高了准确性，因为它提供了对事实数据的访问。

（4）旁白：基于此增强输入，LLM 生成包含事实的正确的信息响应，并将其发送回聊天机器人以交付给用户。

因此，通过结合外部数据源并将相关上下文注入提示中，RAG 增强了 LLM 生成信息的准确性、及时性及与搜索主题的一致性。

接下来介绍如何在 LangChain 框架下应用 RAG，首先创建一个拥有几个工具的 Agent，代码如下：

```
//第 5 章/ Ragagentinit.py
from langchain.agents import (
    AgentExecutor, AgentType, initialize_agent, load_tools
)
from langchain.chat_models import ChatOpenAI
def load_agent() -> AgentExecutor:
    llm = ChatOpenAI(temperature=0, streaming=True)
    tools = load_tools(
        tool_names=["ddg-search", "wolfram-alpha", "arxiv", "wikipedia"],
        llm=llm
    )
    return initialize_agent(
```