

15 | 深入使用LLMChain，给AI连上Google和计算器

2023-04-12 徐文浩 来自北京

《AI大模型之美》



你好，我是徐文浩。

上一讲里，我们一起学习了怎么通过 LangChain 这个 Python 包，链式地调用 OpenAI 的 API。通过链式调用的方式，我们可以把一个需要询问 AI 多轮才能解决的问题封装起来，把一个通过自然语言多轮调用才能解决的问题，变成了一个函数调用。

不过，LangChain 能够帮到我们的远不止这一点。前一阵，ChatGPT 发布了 [🔗 Plugins](#) 这个插件机制。通过 Plugins，ChatGPT 可以浏览整个互联网，还可以接上 Wolfram 这样的科学计算工具，能够实现很多原先光靠大语言模型解决不好的问题。不过，这个功能目前还是处于 wait list 的状态，我也还没有拿到权限。

不过没有关系，我们通过 LangChain 也能实现这些类似的功能。今天这一讲，我们就继续深入挖掘一下 Langchain，看看它怎么解决这些问题。


解决 AI 数理能力的难题

很多人发现，虽然 ChatGPT 回答各种问题的时候都像模像样的，但是一到计算三位数乘法的时候就露馅儿了。感觉它只是快速估计了一个数字，而不是真的准确计算了。我们来看下面这段代码，我们让 OpenAI 帮我们计算一下 352×493 等于多少，你会发现，它算得大差不差，但还是算错了。这就很尴尬，如果我们真的想要让它来担任一个小学数学的助教，总是给出错误的答案也不是个事儿。

 复制代码

```
1 import openai, os
2
3 openai.api_key = os.environ.get("OPENAI_API_KEY")
4
5 from langchain.prompts import PromptTemplate
6 from langchain.llms import OpenAI
7 from langchain.chains import LLMChain
8
9 llm = OpenAI(model_name="text-davinci-003", max_tokens=2048, temperature=0.5)
10 multiply_prompt = PromptTemplate(template="请计算一下{question}是多少?", input_variables=["question"])
11 math_chain = LLMChain(llm=llm, prompt=multiply_prompt, output_key="answer")
12 answer = math_chain.run({"question": "352乘以493"})
13 print("OpenAI API 说答案是:", answer)
14
15 python_answer = 352 * 493
16 print("Python 说答案是:", python_answer)
```

输出结果：


 复制代码

```
1 OpenAI API 说答案是:
2 352 x 493 = 174,336
3 Python 说答案是: 173536
```

注：可以看到，OpenAI 给出的结果，答案是错误的。


不过，有人很聪明，说虽然 ChatGPT 直接算这些数学题不行，但是它不是会写代码吗？我们直接让它帮我们写一段利用 Python 计算这个数学式子的代码不就好了吗？的确，如果你让它

写一段 Python 代码，给出的代码是没有问题的。

 复制代码


```
1 multiply_by_python_prompt = PromptTemplate(template="请写一段Python代码，计算{questi
2 math_chain = LLMChain(llm=llm, prompt=multiply_by_python_prompt, output_key="answ
3 answer = math_chain.run({"question": "352乘以493"})
4 print(answer)
```

输出结果：

 复制代码


```
1 print(352 * 493)
```

不过，我们不想再把这段代码，复制粘贴到 Python 的解释器或者 Notebook 里面，再去手工执行一遍。所以，我们可以在后面再调用一个 Python 解释器，让整个过程自动完成，对应的代码我也放在了下面。

 复制代码

```
1 multiply_by_python_prompt = PromptTemplate(template="请写一段Python代码，计算{questi
2 math_chain = LLMChain(llm=llm, prompt=multiply_by_python_prompt, output_key="answ
3 answer_code = math_chain.run({"question": "352乘以493"})
4
5 from langchain.utilities import PythonREPL
6 python_repl = PythonREPL()
7 result = python_repl.run(answer_code)
8 print(result)
```

输出结果：


 复制代码

```
1 173536
```

注：生成的 Python 脚本是正确的，再通过调用 Python 解释器就能得到正确的计算结果。


可以看到，LangChain 里面内置了一个 utilities 的包，里面包含了 PythonREPL 这个类，可以实现对 Python 解释器的调用。如果你去翻看一下对应代码的源码的话，它其实就是简单地调用了一下系统自带的 exec 方法，来执行 Python 代码。utilities 里面还有很多其他的类，能够实现很多功能，比如可以直接运行 Bash 脚本，调用 Google Search 的 API 等等。你可以去 LangChain 的 [📄 文档](#)，看看它内置的这些工具类有哪些。

如果你仔细想一下，你会发现这其实也是一种链式调用。只不过，调用链里面的第二步，不是去访问 OpenAI 的 API 而已。所以，对于这些工具能力，LangChain 也把它们封装成了 LLMChain 的形式。比如刚才的数学计算问题，是一个先生成 Python 脚本，再调用 Python 解释器的过程，LangChain 就把这个过程封装成了一个叫做 LLMMathChain 的 LLMChain。不需要自己去生成代码，再调用 PythonREPL，只要直接调用 LLMMathChain，它就会在背后把这一切都给做好，对应的代码我也放在下面。

 复制代码

```
1 from langchain import LLMMathChain
2
3 llm_math = LLMMathChain(llm=llm, verbose=True)
4 result = llm_math.run("请计算一下352乘以493是多少?")
5 print(result)
```

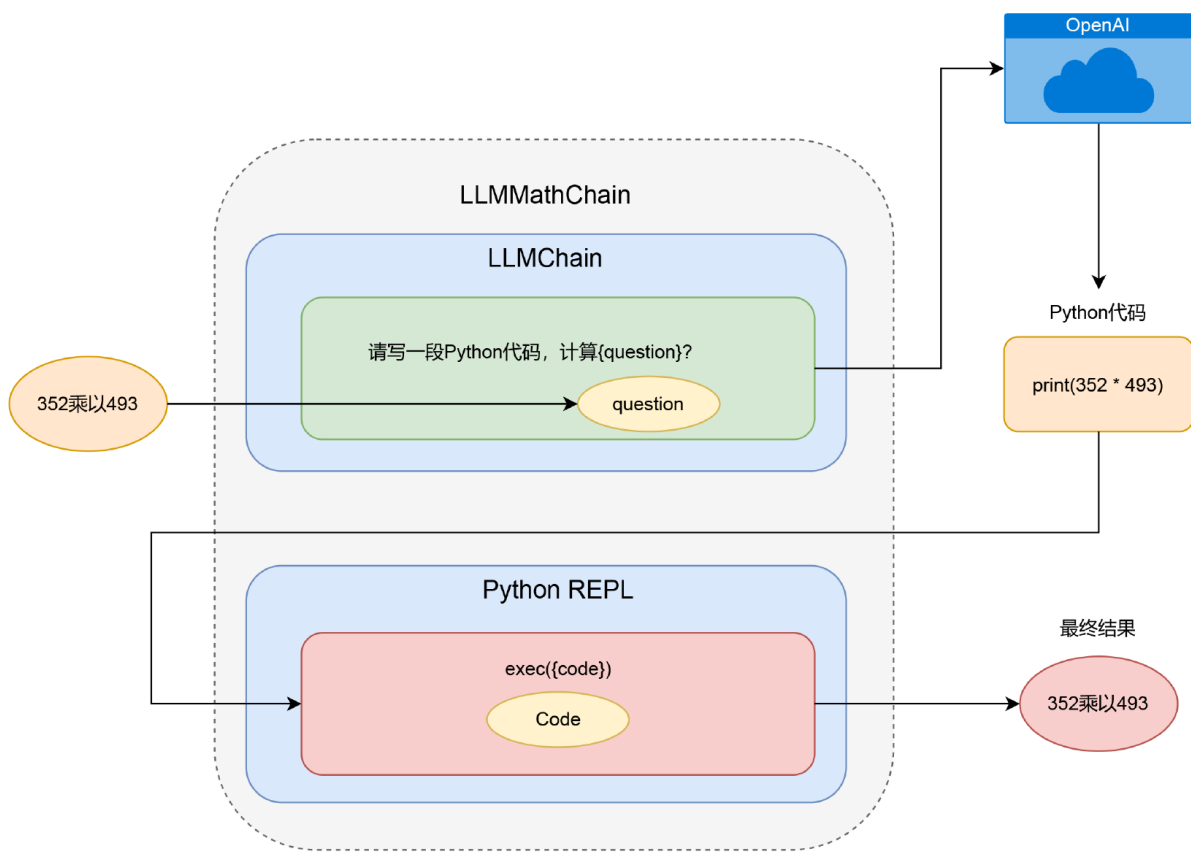
输出结果：

 复制代码

```
1 > Entering new LLMMathChain chain...
2 请计算一下352乘以493是多少?
3
4 print(352 * 493)
5
6 Answer: 173536
7 > Finished chain.
8 Answer: 173536
```

LangChain 也把前面讲过的 utilities 包里面的很多功能，都封装成了 Utility Chains。比如，SQLDatabaseChain 可以直接根据你的数据库生成 SQL，然后获取数据，

LLMRequestsChain 可以通过 API 调用外部系统，获得想要的答案。你可以直接在 LangChain 关于 Utility Chains 的 [文档](#) 里面，找到有哪些工具可以用。



LLMathChain通过OpenAI生成Python代码，再通过REPL执行Python代码，完成数学计算

通过 RequestsChain 获取实时外部信息

这里我们来重点讲一讲如何通过 API 来调用外部系统，获得想要的答案。之前在介绍 llama-index 的时候，我们已经介绍过一种为 AI 引入外部知识的方法了，那就是计算这些外部知识的 Embedding，然后作为索引先保存下来。但是，这只适用于处理那些预先准备好会被问到的知识，比如一本书、一篇论文。这些东西，内容多但是固定，也不存在时效性问题，我们可以提前索引好，而且用户问的问题往往也有很强的相似性。

但是，对于时效性强的问题，这个方法不太适用，因为我们可能没有必要不停地更新索引。比如，你想要知道实时的天气情况，我们不太可能把全球所有城市最新的天气信息每隔几分钟都

索引一遍。

这个时候，我们可以使用 LLMRequestsChain，通过一个 HTTP 请求来得到问题的答案。最简单粗暴的一个办法，就是直接通过一个 HTTP 请求来问一下 Google。

 复制代码

```
1 from langchain.chains import LLMRequestsChain
2
3 template = """在 >>> 和 <<< 直接是来自Google的原始搜索结果。
4 请把对于问题 '{query}' 的答案从里面提取出来，如果里面没有相关信息的话就说 "找不到"
5 请使用如下格式：
6 Extracted:<answer or "找不到">
7 >>> {requests_result} <<<
8 Extracted:"""
9
10 PROMPT = PromptTemplate(
11     input_variables=["query", "requests_result"],
12     template=template,
13 )
14 requests_chain = LLMRequestsChain(llm_chain = LLMChain(llm=OpenAI(temperature=0),
15 question = "今天上海的天气怎么样？"
16 inputs = {
17     "query": question,
18     "url": "https://www.google.com/search?q=" + question.replace(" ", "+")
19 }
20 result=requests_chain(inputs)
21 print(result)
22 print(result['output'])
```

输出结果：

 复制代码

```
1 {'query': '今天上海的天气怎么样?', 'url': 'https://www.google.com/search?q=今天上海的
2 小雨; 10°C~15°C; 东北风 风力4-5级
```

我们来看看这段代码，基于 LLMRequestsChain，我们用到了之前使用过的好几个技巧。


1. 首先，因为我们是简单粗暴地搜索 Google。但是我们想要的是一个有价值的天气信息，而不是整个网页。所以，我们还需要通过 ChatGPT 把网页搜索结果里面的答案给找出来。所以我们定义了一个 PromptTemplate，通过一段提示语，让 OpenAI 为我们在搜索结果里面，找出问题的答案，而不是去拿原始的 HTML 页面。
2. 然后，我们使用了 LLMRequestsChain，并且把刚才 PromptTemplate 构造的一个普通的 LLMChain，作为构造函数的一个参数，传给 LLMRequestsChain，帮助我们在搜索之后处理搜索结果。
3. 对应的搜索词，通过 query 这个参数传入，对应的原始搜索结果，则会默认放到 requests_results 里。而通过我们自己定义的 PromptTemplate 抽取出来的最终答案，则会放到 output 这个输出参数里面。

我们运行一下，就可以看到我们通过简单搜索 Google 加上通过 OpenAI 提取搜索结果里面的答案，就得到了最新的天气信息。

通过 TransformationChain 转换数据格式

有了实时的外部数据，我们就又有很多做应用的创意了。比如说，我们可以根据气温来推荐大家穿什么衣服。我们可以要求如果最低温度低于 0 度，就要推荐用户去穿羽绒服。或者，根据是否下雨来决定要不要提醒用户出门带伞。

不过，在现在的返回结果里，天气信息（天气、温度、风力）只是一段文本，而不是可以直接获取的 JSON 格式。当然，我们可以在 LLMChain 里面再链式地调用一次 OpenAI 的接口，把这段文本转换成 JSON 格式。但是，这样做的话，一来还要消耗更多的 Token、花更多的钱，二来这也会进一步增加程序需要运行的时间，毕竟一次往返的网络请求也是很慢的。这里的文本格式其实很简单，我们完全可以通过简单的字符串处理完成解析。

 复制代码


```
1 import re
2 def parse_weather_info(weather_info: str) -> dict:
3     # 将天气信息拆分成不同部分
4     parts = weather_info.split('; ')
5
6     # 解析天气
7     weather = parts[0].strip()
8
```

```

9      # 解析温度范围, 并提取最小和最大温度
10     temperature_range = parts[1].strip().replace('°C', '').split('~')
11     temperature_min = int(temperature_range[0])
12     temperature_max = int(temperature_range[1])
13
14     # 解析风向和风力
15     wind_parts = parts[2].split(' ')
16     wind_direction = wind_parts[0].strip()
17     wind_force = wind_parts[1].strip()
18
19     # 返回解析后的天气信息字典
20     weather_dict = {
21         'weather': weather,
22         'temperature_min': temperature_min,
23         'temperature_max': temperature_max,
24         'wind_direction': wind_direction,
25         'wind_force': wind_force
26     }
27
28     return weather_dict
29
30 # 示例
31 weather_info = "小雨; 10°C~15°C; 东北风 风力4-5级"
32 weather_dict = parse_weather_info(weather_info)
33 print(weather_dict)

```


输出结果:

 复制代码

```
1 {'weather': '小雨', 'temperature': {'min': 10, 'max': 15}, 'wind': {'direction': '东北风', 'force': '4-5级'}}
```

注: 上面这段代码, 其实是我让 GPT-4 写的。

我们在这里实现了一个 **parse_weather_info** 函数, 可以把前面 LLMRequestsChain 的输出结果, 解析成一个 dict。不过, 我们能不能更进一步, 把这个解析的逻辑, 也传到 LLMChain 的链式调用的最后呢? 答案当然是可以的。对于这样的要求, Langchain 里面也有一个专门的解决方案, 叫做 TransformChain, 也就是做格式转换。


 复制代码


```

1 from langchain.chains import TransformChain, SequentialChain
2
3
4 def transform_func(inputs: dict) -> dict:
5     text = inputs["output"]
6     return {"weather_info" : parse_weather_info(text)}
7
8 transformation_chain = TransformChain(input_variables=["output"],
9                                       output_variables=["weather_info"], transform_func=transform_func)
10
11 final_chain = SequentialChain(chains=[requests_chain, transformation_chain],
12                               input_variables=["query", "url"], output_variables=["weather_info"],
13                               llm_chain=llm_chain)
14 final_result = final_chain.run(inputs)
15 print(final_result)

```

输出结果：

 复制代码

```

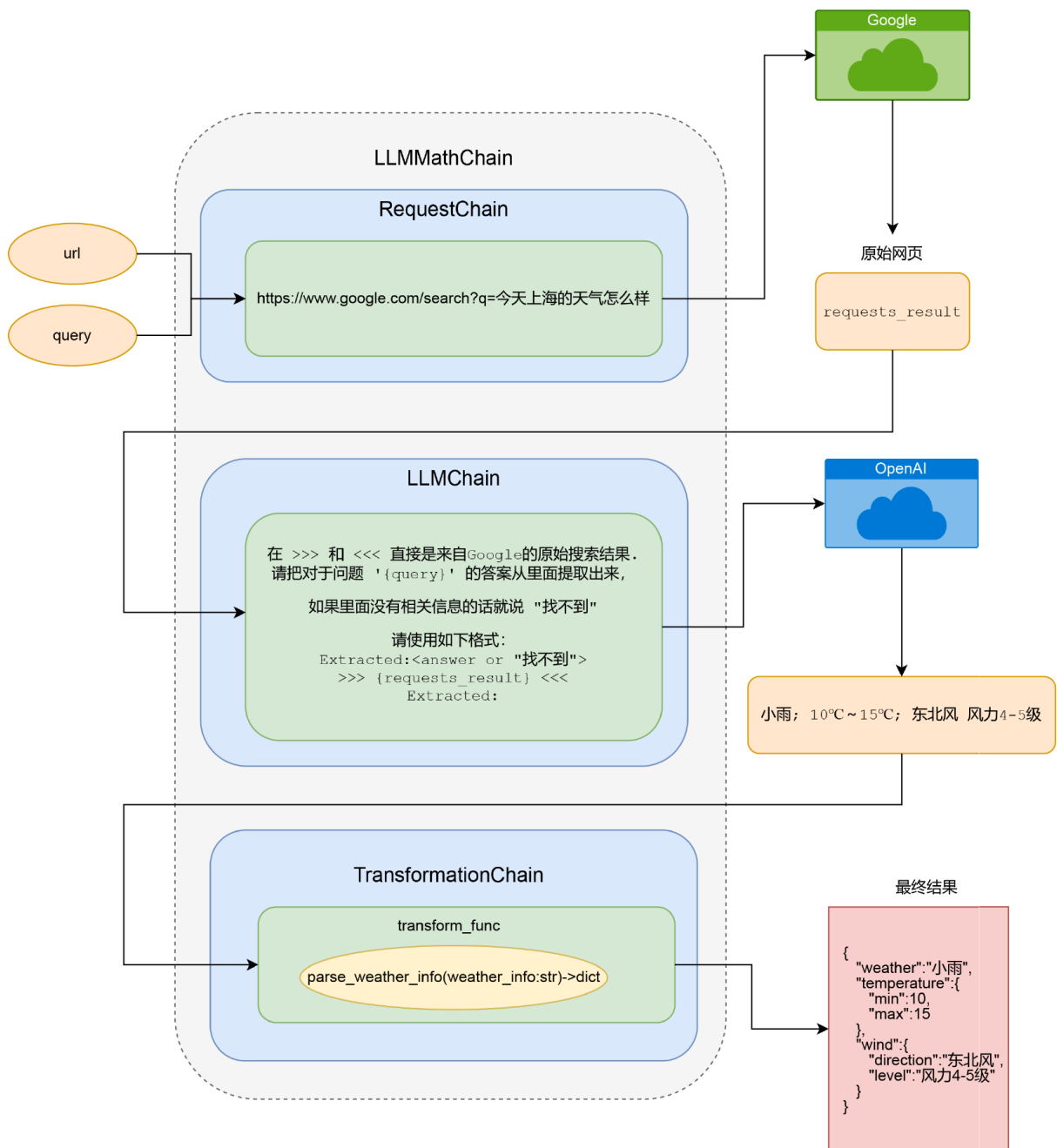
1 {'weather': '小雨', 'temperature': {'min': 10, 'max': 15}, 'wind': {'direction': '北', 'speed': 10}}

```

注：在 requests_chain 后面跟上一个 transformation_chain，我们就能把结果解析成 dict，供后面其他业务使用结构化的数据。

1. 我们在这里，先定义了一个 transform_func，对前面的 parse_weather_info 函数做了一下简单的封装。它的输入，是整个 LLMChain 里，执行到 TransformChain 之前的整个输出结果的 dict。我们前面看到整个 LLMRequestsChain 里面的天气信息的文本内容，是通过 output 这个 key 拿到的，所以这里我们也是先通过它来拿到天气信息的文本内容，再调用 parse_weather_info 解析，并且把结果输出到 weather_info 这个字段里。
2. 然后，我们就定义了一个 TransformChain，里面的输入参数就是 output，输出参数就是 weather_info。
3. 最后，我们通过上一讲用过的 SequentialChain，将前面的 LLMRequestsChain 和这里的 TransformChain 串联到一起，变成一个新的叫做 final_chain 的 LLMChain。

在这三步完成之后，未来我们想要获得天气信息，并且拿到一个 dict 形式的输出，只要调用 final_chain 的 run 方法，输入我们关于天气的搜索文本就好了。



通过三个步骤，拿到最后结构化的天气信息

最后，我们来梳理一下 `final_chain` 都做了哪些事。

1. 通过一个 HTTP 请求，根据搜索词拿到 Google 的搜索结果页。

2. 把我们定义的 Prompt 提交给 OpenAI，然后把我们搜索的问题和结果页都发给了 OpenAI，让它从里面提取出搜索结果页里面的天气信息。
3. 最后我们通过 transform_func 解析拿到的天气信息的文本，被转换成一个 dict。这样，后面的程序就好处理了。

通过 VectorDBQA 来实现先搜索再回复的能力

此外，还有一个常用的 LLMChain，就是我们之前介绍的 llama-index 的使用场景，也就是针对自己的资料库进行问答。我们预先把资料库索引好，然后每次用户来问问题的时候，都是先到这个资料库里搜索，再把问题和答案一并交给 AI，让它去组织语言回答。

 复制代码


```
1 from langchain.embeddings.openai import OpenAIEmbeddings
2 from langchain.vectorstores import FAISS
3 from langchain.text_splitter import SpacyTextSplitter
4 from langchain import OpenAI, VectorDBQA
5 from langchain.document_loaders import TextLoader
6
7 llm = OpenAI(temperature=0)
8 loader = TextLoader('./data/ecommerce_faq.txt')
9 documents = loader.load()
10 text_splitter = SpacyTextSplitter(chunk_size=256, pipeline="zh_core_web_sm")
11 texts = text_splitter.split_documents(documents)
12
13 embeddings = OpenAIEmbeddings()
14 docsearch = FAISS.from_documents(texts, embeddings)
15
16 faq_chain = VectorDBQA.from_chain_type(llm=llm, vectorstore=docsearch, verbose=True)
```

注：上面的代码创建了一个基于 FAISS 进行向量存储的 docsearch 的索引，以及基于这个索引的 VectorDBQA 这个 LLMChain。

先来看第一段代码，我们通过一个 TextLoader 把文件加载进来，还通过 SpacyTextSplitter 给文本分段，确保每个分出来的 Document 都是一个完整的句子。因为我们这里的文档是电商 FAQ 的内容，都比较短小精悍，所以我们设置的 chunk_size 只有 256。然后，我们定义了使用 OpenAIEmbeddings 来给文档创建 Embedding，通过 FAISS 把它存储成一个


VectorStore。最后，我们通过 VectorDBQA 的 from_chain_type 定义了一个 LLM。对应的 FAQ 内容，我是请 ChatGPT 为我编造之后放在了 ecommerce_faq.txt 这个文件里。

问题：

 复制代码


```
1 question = "请问你们的货，能送到三亚吗？大概需要几天？"  
2 result = faq_chain.run(question)  
3 print(result)
```

输出结果：

 复制代码


```
1 > Entering new VectorDBQA chain...  
2 > Finished chain.  
3 我们支持全国大部分省份的配送，包括三亚。一般情况下，大部分城市的订单在2-3个工作日内送达，偏远地
```

问题：

 复制代码

```
1 question = "请问你们的退货政策是怎么样的？"  
2 result = faq_chain.run(question)  
3 print(result)
```

输出结果：

 复制代码

```
1 > Entering new VectorDBQA chain...  
2 > Finished chain.  
3 自收到商品之日起7天内，如产品未使用、包装完好，您可以申请退货。某些特殊商品可能不支持退货，请在购
```

我向它提了两个不同类型的问题，faq_chain 都能够正确地回答出来。你可以去看看 data 目录下面的 ecommerce_faq.txt 文件，看看它的回答是不是和文档写的内容一致。在 VectorDBQA 这个 LLMChain 背后，其实也是通过一系列的链式调用，来完成搜索 VectorStore，再向 AI 发起 Completion 请求这样两个步骤的。

可以看到 LLMChain 是一个很强大的武器，它可以把解决一个问题需要的多个步骤串联到一起。这个步骤可以是调用我们的语言模型，也可以是调用一个外部 API，或者在内部我们定义一个 Python 函数。这大大增强了我们利用大语言模型的能力，特别是能够弥补它的很多不足之处，比如缺少有时效的信息，通过 HTTP 调用比较慢等等。

小结

好了，这一讲到这里也就结束了。

我们可以看到，Langchain 的链式调用并不局限于使用大语言模型的接口。这一讲里，我们就看到四种常见的将大语言模型的接口和其他能力结合在一起的链式调用。

1. LLMMathChain 能够通过 Python 解释器变成一个计算器，让 AI 能够准确地进行数学运算。
2. 通过 RequestsChain，我们可以直接调用外部 API，然后再让 AI 从返回的结果里提取我们关心的内容。
3. TransformChain 能够让我们根据自己的要求对数据进行处理和转化，我们可以把 AI 返回的自然语言的结果进一步转换成结构化的数据，方便其他程序去处理。
4. VectorDBQA 能够完成和 llama-index 相似的事情，只要预先做好内部数据资料的 Embedding 和索引，通过对 LLMChain 进行一次调用，我们就可以直接获取回答的结果。

这些能力大大增强了 AI 的实用性，解决了几个之前大语言模型处理得不好的问题，包括数学计算能力、实时数据能力、和现有程序结合的能力，以及搜索属于自己的资料库的能力。你完全可以定义自己需要的 LLMChain，通过程序来完成各种任务，然后合理地组合不同类型的 LLMChain 对象，来实现连 ChatGPT 都做不到的事情。而 ChatGPT Plugins 的实现机制，其实也是类似的。

思考题

最后，我给你留一道思考题。我们前面说过，Langchain 里有 SQLDatabaseChain 可以直接让我们写需求访问数据库。在官方文档里也给出了对应的 [例子](#)，你可以去试一试体验一下，想一想它是通过什么样的提示语信息，来让 AI 写出可以直接执行的 SQL 的？

欢迎你把你体验之后的感受以及思考后的结果分享在评论区，也欢迎你把这一讲分享给感兴趣的朋友，我们下一讲再见！

推荐试用

我们目前对于 Langchain 的讲解，都是通过 Python 编程的方式来实现真实业务场景的需求的。有人直接为 Langchain 做了一个可以拖拽的图形界面叫做 [LangFlow](#)。你可以试着下载体验一下，看看图形界面是不是可以进一步提升你的效率。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (18)



奥伟

2023-04-12 来自北京

老师你好，chain和pipe的原理还是比较像，学完这讲打开了我对ChatGPT可应用空间。实际应用中最大的问题还是api 的响应速度和接口限流。

请问老师有没有更好的解决办法？目前我知道的是多个账号负载均衡和调用容错重试。

作者回复：申请开通海外的Azure OpenAI，额度和性能都更有保障

共 2 条评论 >



neohope

2023-04-12 来自上海

ChatGPT被作为一种能力去使用，也就是X+AI模式，确实有不小的想象空间。

但反过来，如有一个大模型，能快速整合各种外部能力，从X+AI，变成AI+X、Y、Z，很可能会成为下一代互联网的入口，并从各种维度给人类带来全新体验，期待这个时代能尽快到来。



金口

2023-04-14 来自湖南

怎么让模板更通用呢？比如那个算术问题，如何判断是个计算问题，感觉模板只能解决调用问题。

作者回复：继续往下先看到17讲之后看看你有没有解决问题的思路。

共 2 条评论 >



Yezhiwei

2023-04-13 来自北京

之前自己试过 LangFlow 非常简单，把步骤记录下来了，难道在网络，如果网络稳定，很快看到效果

https://mp.weixin.qq.com/s/jjNkkoUu-q8Yb1J6_u6WGg

作者回复：👍



自然卷的Neil

2023-04-12 来自浙江

```
"url": "https://www.google.com/search?q=" + question.replace(" ", "+")
```

这一段代码为什么需要加上+ question.replace(" ", "+")

我看了langchain的手册也是这么写的，

如果输入是'query': 'What are the Three (3) biggest countries, and their respective sizes?',

输出会变成'url': 'https://www.google.com/search?q=What+are+the+Three+(3)+biggest+countries,+and+their+respective+sizes?'

不太理解这样做的意义是啥

作者回复：HTTP协议的URL里面不支持空格

Google的URL含义里面，空格默认会替换成+

你试着用Google搜索一下看一下浏览器的地址栏就知道了

共 3 条评论 >





海阔天空

2023-04-12 来自浙江

浩哥，文稿中代码、类库都是python脚步的，有java相关的生态推进吗？

作者回复: Java目前还没有看到，不过现在GPT-4很多人用来做不同语言之间的代码翻译，你可以试着写一个Java版本的Langchain。



金口

2023-04-12 来自湖南

这个确实不错，我一直在思考如何让chatgpt优雅的完成最后一步，langchain这个工具能很大程度解决这个问题。这个工具支持清华的那个开源大模型吗？

作者回复: 第12讲我们介绍过在llama-index里，怎么封装一个自己的CustomLLM。Langchain也可以用类似的方式来解决呀。



zgxx

2023-05-24 来自浙江

```
final_result = final_chain.run(inputs)
```

inputs是哪里来的



厚积薄发

2023-05-11 来自德国

```
from langchain.chains import LLMRequestsChain
```

```
template = """在 >>> 和 <<< 直接是来自Google的原始搜索结果.
```

```
请把对于问题 '{query}' 的答案从里面提取出来，如果里面没有相关信息的话就说 "找不到"
请使用如下格式：
```

```
Extracted:<answer or "找不到">
```

```
>>> {requests_result} <<<
```

```
Extracted:"""
```

```
PROMPT = PromptTemplate(
```



```
input_variables=["query", "requests_result"],
template=template,
)
requests_chain = LLMRequestsChain(llm_chain = LLMChain(llm=OpenAI(temperature=0),
prompt=PROMPT),verbose=True)
question = "今天上海的天气怎么样？"
inputs = {
    "query": question,
    "url": "https://www.google.com/search?q=" + question.replace(" ", "+")
}
result=requests_chain(inputs)
print(result)
print(result['output'])
```

老师，我测试这段代码，返回值是找不到，是什么原因哈

作者回复：我之前会遇到在Colab上运行不成功，原因是 Google 搜索的反爬策略应该还是没有给到正常的网页返回内容

你可以看一下，是否网络能够访问Google搜索，以及是否用了一些数据中心的机器被Google的反爬虫屏蔽了



Geek_9691fb

2023-05-06 来自浙江

ecommerce_faq.txt 在哪里可以找到

作者回复: <https://github.com/xuwenhao/geektime-ai-course> 的 data目录下



沐瑞Lynn

2023-04-27 来自重庆

用LLMRequestsChain想谷歌提问，总是回答找不到；然后用requests直接写了个程序调google 页面，发现可能被拦截。然后问题了GPT，得到回答

“这段代码使用 requests 模块向 Google 搜索发出 GET 请求来获取指定问题的搜索结果页面的 HTML 代码。然后使用 print() 函数将 HTML 代码打印出来。

但是使用该方法获取 Google 搜索结果可能会被 Google 识别为机器行为，并被阻拦。建议使用 Google 提供的 API 来获取搜索结果。”记录一下，看看大家是不是有同样的问题吧。

作者回复: 如果在Colab发起，似乎很容易被拦截，如果是本地发起，好像没有问题



旭

2023-04-26 来自北京

如果是提供服务的话，链式计算中间步骤的容错性有没有好办法保证？

作者回复: 可以加入一些容错的步骤，类似于13/14讲生成unit test的时候，去检查生成的条目数，或者编译代码看是否能够编译通过。



慕枫

2023-04-17 来自贵州

RequestsChain怎么判断是问题是否需要调用外部api？

作者回复: 你使用RequestsChain的时候已经指明了外部的URL，一定会调用外部API呀。



凝望

2023-04-16 来自广东

RAyH4c

微博

给开发和使用LLM应用的各位提个醒，各路聊天机器人和ChatXXx应用都使用的LangChain库，对输入的提示词包了一层exec和eval函数，导致所有提示词都能远程执行任意python代码...

老师好，看微博有人说这个问题，请问影响大吗？

作者回复: 有影响，在生产环境使用会有安全隐患，不过相信社区很快会修正这个问题

共 3 条评论 >



榕

2023-04-13 来自广东

老师好，通过langchain等武器库确实给了我们使用大语言模型很大的想象空间，但前提得有大语言模型支持。对于地区限制访问的问题，那对于国内的个人或企业要真正落地这些能力，目前是通过哪些方式来做呢？谢谢

作者回复: 可以看看在一些开源模型上调优，我们前面也介绍过比如ChatGLM或者用现在百度阿里开放出来的。

共 2 条评论 >



Geek_4ec46c

2023-04-12 来自福建

老师这段代码里面:

```
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.text_splitter import SpacyTextSplitter
from langchain import OpenAI, VectorDBQA
from langchain.document_loaders import TextLoader
```

```
llm = OpenAI(temperature=0)
loader = TextLoader('./data/ecommerce_faq.txt')
documents = loader.load()
text_splitter = SpacyTextSplitter(chunk_size=256, pipeline="zh_core_web_sm")
texts = text_splitter.split_documents(documents)
```

```
embeddings = OpenAIEmbeddings()
docsearch = FAISS.from_documents(texts, embeddings)
```

```
faq_chain = VectorDBQA.from_chain_type(llm=llm, vectorstore=docsearch, verbose=True)
```

可以把索引给保存到数据库吗？以后继续使用？

作者回复: 可以啊, Langchain支持很多不同类型的VectorStore, 可以去看文档
<https://python.langchain.com/en/latest/modules/indexes/vectorstores.html>

这里使用FAISS只是演示的时候环境要求比较低。



Geek_4ec46c

2023-04-12 来自福建

老师,我有个问题可以解答一下吗?我们目前开发的一款app,就有客户问到类似的问题,比如今天的时间,Chatgpt就没办法完整的回答,我看到上面的编译Python的方式似乎可行,但是似乎用户的问题千奇百怪,比如有的又会问到数学...而有的llm直接就能解决,在这种情况下,我要怎么实现会更好?

作者回复: RequestsChain可以访问Google, 17讲里介绍的Agent还可以让AI自动选择用什么工具, 继续往下看吧

共 2 条评论 >



Evan

2023-04-12 来自上海

RequestsChain 非常有效的, 能调用外部接口的

