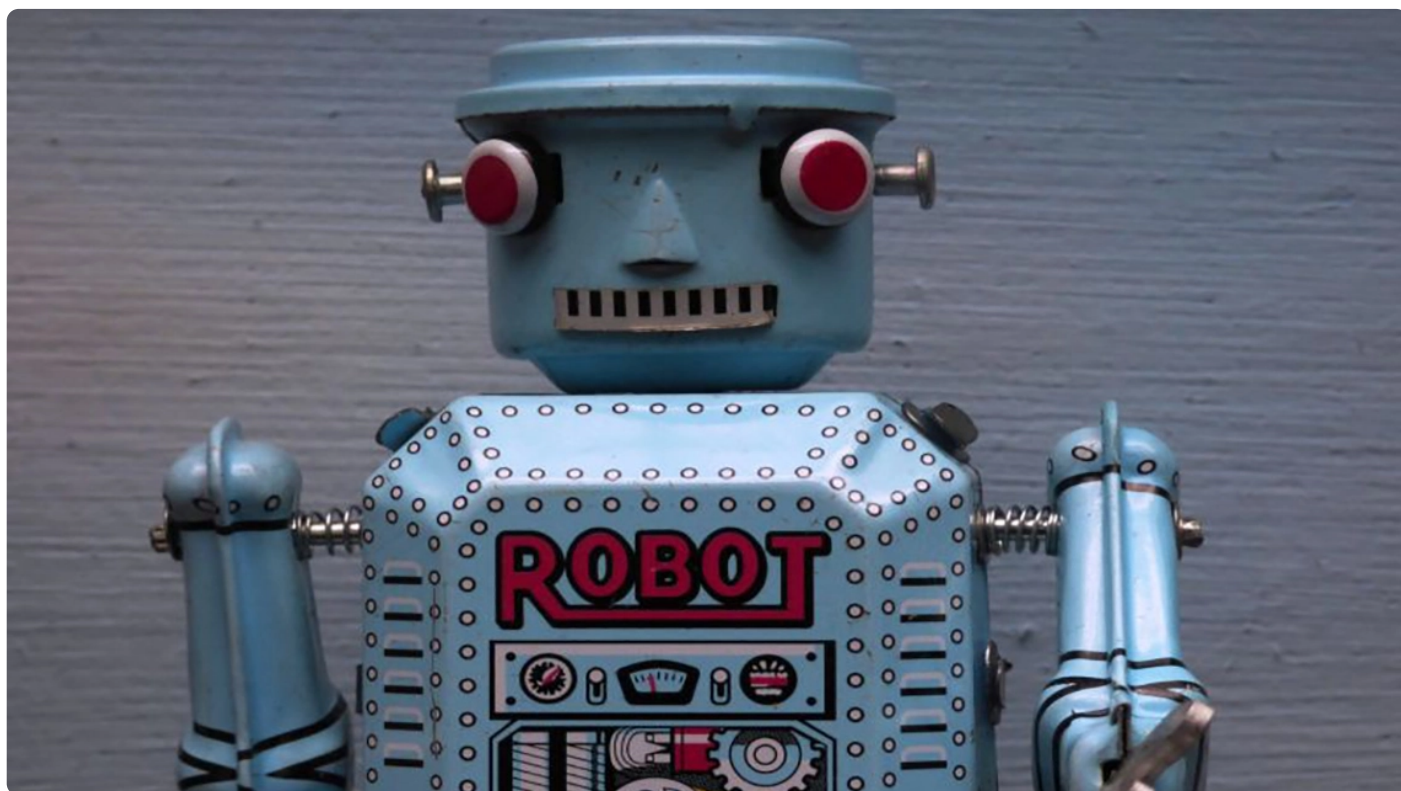


## 14 | 链式调用，用LangChain简化多步提示语

2023-04-11 徐文浩 来自北京

《AI大模型之美》



你好，我是徐文浩。

OpenAI 的大语言模型，只是提供了简简单单的 Completion 和 Embedding 这样两个核心接口。但是你也看到了，在过去的 13 讲里，通过合理使用这两个接口，我们完成了各种各样复杂的任务。

通过提示语（Prompt）里包含历史的聊天记录，我们能够让 AI 根据上下文正确地回答问题。

通过将 Embedding 提前索引好存起来，我们能够让 AI 根据外部知识回答问题。

而通过多轮对话，将 AI 返回的答案放在新的问题里，我们能够让 AI 帮我们给自己的代码撰写单元测试。

这些方法，也是一个实用的自然语言类应用里常见的模式。我之前也都通过代码为你演示过具体的做法。但是，如果我们每次写应用的时候，都需要自己再去 OpenAI 提供的原始 API 里做一遍，那就太麻烦了。于是，开源社区就有人将这些常见的需求和模式抽象了出来，开发了一个叫做 Langchain 的开源库。那么接下来，我们就来看看如何使用 LangChain 来快速实现之前我们利用大语言模型实现过的功能。以及我们如何进一步地，将 Langchain 和我们的业务系统整合，完成更复杂、更有实用价值的功能。

## 使用 Langchain 的链式调用

如果你观察得比较仔细的话，你会发现在 [🔗第 11 讲](#) 我们使用 llama-index 的时候，就已经装好 LangChain 了。llama-index 专注于为大语言模型的应用构建索引，虽然 Langchain 也有类似的功能，但这一点并不是 Langchain 的主要卖点。Langchain 的第一个卖点其实就在它的名字里，也就是**链式调用**。

我们先来看一个使用 ChatGPT 的例子，你就能理解为什么会有链式调用的需求了。我们知道，GPT-3 的基础模型里面，中文的语料很少。用中文问它问题，很多时候它回答得不好。所以有时候，我会迂回处理一下，先把中文问题给 AI，请它翻译成英文，然后再把英文问题贴进去提问，得到一个英文答案。最后，再请 AI 把英文答案翻译回中文。很多时候，问题的答案会更准确一点。比如，下面的截图里，我就请它简单介绍一下 Stable Diffusion 的原理是什么。

注：Stable Diffusion 是一个热门的开源 AI 画图工具，后面我们在介绍用 AI 生成图片的时候会用到。

## 人工链式调用



请把下面这句话翻译成英语：



请你作为一个AIGC的专家，介绍一下Stable Diffusion的原理



As an expert in AIGC, could you please introduce the principle of Stable Diffusion?



As an expert in AIGC, could you please introduce the principle of Stable Diffusion?



Sure, Stable Diffusion is a mathematical model used in image processing and computer vision. It aims to address the challenge of image denoising and restoration by considering the image as a signal that is corrupted by noise.



The principle of Stable Diffusion involves applying a nonlinear diffusion process to the noisy image, where the diffusion coefficient is chosen to balance the trade-off between preserving the edges and removing the noise. The process smooths out the noise while preserving the edges of the image, leading to a denoised and restored image.

Stable Diffusion is a powerful technique because it can handle a wide range of noise models and produce high-quality results even in the presence of heavy noise. It has many applications in areas such as medical imaging, remote sensing, and digital photography.

先让AI把中文问题翻译成英文，再直接把英文问题贴进去得到英文答案



请你把下面这一段翻译成中文



Sure, Stable Diffusion is a mathematical model used in image processing and computer vision. It aims to address the challenge of image denoising and restoration by considering the image as a signal that is corrupted by noise.

The principle of Stable Diffusion involves applying a nonlinear diffusion process to the noisy image, where the diffusion coefficient is chosen to balance the trade-off between preserving the edges and removing the noise. The process smooths out the noise while preserving the edges of the image, leading to a denoised and restored image.

Stable Diffusion is a powerful technique because it can handle a wide range of noise models and produce high-quality results even in the presence of heavy noise. It has many applications in areas such as medical imaging, remote sensing, and digital photography.



稳定扩散（Stable Diffusion）是一种用于图像处理和计算机视觉的数学模型，旨在通过将图像视为受到噪声干扰的信号来解决图像去噪和恢复的挑战。



稳定扩散的原理是对噪声图像应用非线性扩散过程，其中扩散系数被选择以平衡保留边缘和去除噪声之间的折衷。该过程可以平滑噪声，同时保留图像的边缘，从而得到去噪和恢复的图像。

稳定扩散是一种强大的技术，因为它可以处理广泛的噪声模型，即使在强噪声存在的情况下也能产生高质量的结果。它在医学影像、遥感和数字摄影等领域具有许多应用。


我们再请它翻译一下英文答案

如果用 API 来实现这个过程，其实就是一个链式调用的过程。

1. 我们先调用 OpenAI，把翻译请求和原始问题组合在一起发送给 AI，完成问题的中译英。
2. 然后再把拿到的翻译好的英文问题发送给 OpenAI，得到英文答案。
3. 最后再把英文答案，和对应要求 AI 翻译答案的请求组合在一起，完成答案的英译中。


## 使用 LLMChain 进行链式调用

如果我们用代码，可以像下面这样，一步步进行。

 复制代码

```
1 import openai, os
2 from langchain.prompts import PromptTemplate
3 from langchain.llms import OpenAI
4 from langchain.chains import LLMChain
5
6 openai.api_key = os.environ.get("OPENAI_API_KEY")
7
8 llm = OpenAI(model_name="text-davinci-003", max_tokens=2048, temperature=0.5)
9
10 en_to_zh_prompt = PromptTemplate(
11     template="请把下面这句话翻译成英文： \n\n {question}?", input_variables=["question"]
12 )
13
14 question_prompt = PromptTemplate(
15     template = "{english_question}", input_variables=["english_question"]
16 )
17
18 zh_to_cn_prompt = PromptTemplate(
19     input_variables=["english_answer"],
20     template="请把下面这一段翻译成中文： \n\n{english_answer}?",
21 )
22
23 question_translate_chain = LLMChain(llm=llm, prompt=en_to_zh_prompt, output_key="")
24 english = question_translate_chain.run(question="请你作为一个机器学习的专家，介绍一下CNN")
25 print(english)
26
27 qa_chain = LLMChain(llm=llm, prompt=question_prompt, output_key="english_answer")
28 english_answer = qa_chain.run(english_question=english)
29 print(english_answer)
30
31 answer_translate_chain = LLMChain(llm=llm, prompt=zh_to_cn_prompt)
32 answer = answer_translate_chain.run(english_answer=english_answer)
33 print(answer)
```

输出结果：

 复制代码

```
1 Please explain the principles of CNN as an expert in Machine Learning.
```



```
2 A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is
3
4 卷积神经网络 (CNN) 是一种深度学习算法，用于分析视觉图像。它模仿人类视觉皮层的结构，由多层神经元组
5
```

这里的代码，我们使用了 Langchain 这个库，不过还没有动用它的链式调用过程。我们主要用了 Langchain 的三个包。


1. LLM，也就是我们使用哪个大语言模型，来回答我们提出的问题。在这里，我们还是使用 OpenAIChat，也就是最新放出来的 gpt-3.5-turbo 模型。
2. PromptTemplate，和我们在 [第 11 讲](#)里看到的 llama-index 的 PromptTemplate 是一个东西。它可以定义一个提示语模版，里面能够定义一些可以动态替换的变量。比如，代码里的 question\_prompt 这个模版里，我们就定义了一个叫做 question 的变量，因为我们每次问的问题都会不一样。事实上，llamd-index 里面的 PromptTemplate 就是对 Langchain 的 PromptTemplate 做了一层简单的封装。
3. 主角 LLMChain，它的构造函数接收一个 LLM 和一个 PromptTemplate 作为参数。构造完成之后，可以直接调用里面的 run 方法，将 PromptTemplate 需要的变量，用 K=>V 对的形式传入进去。返回的结果，就是 LLM 给我们的答案。

不过如果看上面这段代码，我们似乎只是对 OpenAI 的 API 做了一层封装而已。我们构建了 3 个 LLMChain，然后按照顺序调用，每次拿到答案之后，再作为输入，交给下一个 LLM 调用。感觉好像更麻烦了，没有减少什么工作量呀？

别着急，这是因为我们还没有真正用上 LLMChain 的“链式调用”功能，而用这个功能，只需要加上一行小小的代码。我们用一个叫做 SimpleSequentialChain 的 LLMChain 类，把我们要按照顺序依次调用的三个 LLMChain 放在一个数组里，传给这个类的构造函数。


然后对于这个对象，我们调用 run 方法，把我们用中文问的问题交给它。这个时候，这个 SimpleSequentialChain，就会按照顺序开始调用 chains 这个数组参数里面包含的其他 LLMChain。并且，每一次调用的结果，会存储在这个 Chain 构造时定义的 output\_key 参数里。而下一个调用的 LLMChain，里面模版内的变量如果有和之前的 output\_key 名字相同的，就会用 output\_key 里存入的内容替换掉模版内变量所在的占位符。

这次，我们只向这个 SimpleSequentialChain 调用一次 run 方法，把一开始的问题交给它就好了。后面根据答案去问新的问题，这个 LLMChain 会自动地链式搞定。我在这里把日志的 Verbose 模式打开了，你在输出的过程中，可以看到其实这个 LLMChain 是调用了三次，并且中间两次的返回结果你也可以一并看到。

 复制代码


```
1 from langchain.chains import SimpleSequentialChain
2
3 chinese_qa_chain = SimpleSequentialChain(
4     chains=[question_translate_chain, qa_chain, answer_translate_chain], input_key=
5     verbose=True)
6 answer = chinese_qa_chain.run(question="请你作为一个机器学习的专家，介绍一下CNN的原理。")
7 print(answer)
```

Verbose 日志信息：

 复制代码

```
1 > Entering new SimpleSequentialChain chain...
2
3 Please introduce the principle of CNN as a machine learning expert.
4
5 Convolutional Neural Networks (CNNs) are a type of artificial neural network that
6
7 卷积神经网络（CNN）是一种常用于图像识别和分类任务的人工神经网络。它们受到人脑结构的启发，由多层神
8 > Finished chain.
```

输出结果：

 复制代码

```
1 卷积神经网络（CNN）是一种常用于图像识别和分类任务的人工神经网络。它们受到人脑结构的启发，由多层神
```

在使用这样的链式调用的时候，有一点需要注意，就是一个 LLMChain 里，所使用的 PromptTemplate 里的输入参数，**之前必须在 LLMChain 里，通过 output\_key 定义过**。不然，这个变量没有值，程序就会报错。


## 支持多个变量输入的链式调用

事实上，因为使用变量的输入输出，是用这些参数定义的。所以我们不是只能用前一个 LLMChain 的输出作为后一个 LLMChain 的输入。我们完全可以连续问多个问题，然后把这些问题的答案，作为后续问题的输入来继续处理。下面我就给你看一个例子。

 复制代码

```
1 from langchain.chains import SequentialChain
2
3 q1_prompt = PromptTemplate(
4     input_variables=["year1"],
5     template="{year1}年的欧冠联赛的冠军是哪支球队，只说球队名称。"
6 )
7 q2_prompt = PromptTemplate(
8     input_variables=["year2"],
9     template="{year2}年的欧冠联赛的冠军是哪支球队，只说球队名称。"
10 )
11 q3_prompt = PromptTemplate(
12     input_variables=["team1", "team2"],
13     template="{team1}和{team2}哪只球队获得欧冠的次数多一些？"
14 )
15 chain1 = LLMChain(llm=llm, prompt=q1_prompt, output_key="team1")
16 chain2 = LLMChain(llm=llm, prompt=q2_prompt, output_key="team2")
17 chain3 = LLMChain(llm=llm, prompt=q3_prompt)
18
19 sequential_chain = SequentialChain(chains=[chain1, chain2, chain3], input_variables=["year1", "year2"])
20 answer = sequential_chain.run(year1=2000, year2=2010)
21 print(answer)
```

输出结果：

 复制代码

```
1 > Entering new SequentialChain chain...
2 > Finished chain.
3
4 西班牙皇家马德里队获得欧冠的次数更多，共13次，而拜仁慕尼黑只有5次。
```


在这个例子里，我们定义了两个 PromptTemplate 和对应的 LLMChain，各自接收一个年份作为输入，回答这两个年份的欧冠冠军。然后将两个队名作为输入，放到第三个问题里，让



AI 告诉我们这两支球队哪一支获得欧冠的次数多一些。只需要在我们的 SequentialChain 里输入两个年份，就能通过三次回答得到答案。

## 通过 Langchain 实现自动化撰写单元测试

看到这里，不知道你有没有想起我们上一讲刚刚讲过的通过多步提示语自动给代码写单元测试。没错，Langchain 可以顺序地通过多个 Prompt 调用 OpenAI 的 GPT 模型。这个能力拿来实现上一讲的自动化测试的功能是再合适不过的了。下面，我就拿 Langchain 重新实现了一遍上一讲的这个功能，并且给它补上了 AST 语法解析失败之后自动重试的能力。

 复制代码

```
1 from langchain.chains import SequentialChain
2
3 def write_unit_test(function_to_test, unit_test_package = "pytest"):
4     # 解释源代码的步骤
5     explain_code = """# How to write great unit tests with {unit_test_package}
6
7     In this advanced tutorial for experts, we'll use Python 3.10 and `{unit_test_
8     ``python
9     {function_to_test}
10    ``
11
12    Before writing any unit tests, let's review what each element of the function
13    - First, """
14
15    explain_code_template = PromptTemplate(
16        input_variables=["unit_test_package", "function_to_test"],
17        template=explain_code
18    )
19    explain_code_llm = OpenAI(model_name="text-davinci-002", temperature=0.4, max
20        top_p=1, stop=["\n\n", "\n\t\n", "\n    \n"])
21    explain_code_step = LLMChain(llm=explain_code_llm, prompt=explain_code_templa
22
23    # 创建测试计划示例的步骤
24    test_plan = """
25
26    A good unit test suite should aim to:
27    - Test the function's behavior for a wide range of possible inputs
28    - Test edge cases that the author may not have foreseen
29    - Take advantage of the features of `{unit_test_package}` to make the tests e
30    - Be easy to read and understand, with clean code and descriptive names
31    - Be deterministic, so that the tests always pass or fail in the same way
32
33    `{unit_test_package}` has many convenient features that make it easy to write
```

```

34
35     For this particular function, we'll want our unit tests to handle the followi
36     -"""
37     test_plan_template = PromptTemplate(
38         input_variables=["unit_test_package", "function_to_test", "code_explainat
39         template= explain_code + "{code_explanation}" + test_plan
40     )
41     test_plan_llm = OpenAI(model_name="text-davinci-002", temperature=0.4, max_to
42         top_p=1, stop=["\n\n", "\n\t\n", "\n    \n"])
43     test_plan_step = LLMChain(llm=test_plan_llm, prompt=test_plan_template, outpu
44
45     # 撰写测试代码的步骤
46     starter_comment = "Below, each test case is represented by a tuple passed to
47     prompt_to_generate_the_unit_test = """
48
49     Before going into the individual tests, let's first look at the complete suite of
50     ```python
51     import {unit_test_package} # used for our unit tests
52
53     {function_to_test}
54
55     #{starter_comment}"""
56
57     unit_test_template = PromptTemplate(
58         input_variables=["unit_test_package", "function_to_test", "code_explainat
59         template= explain_code + "{code_explanation}" + test_plan + "{test_plan}
60     )
61     unit_test_llm = OpenAI(model_name="text-davinci-002", temperature=0.4, max_to
62     unit_test_step = LLMChain(llm=unit_test_llm, prompt=unit_test_template, outpu
63
64     sequential_chain = SequentialChain(chains=[explain_code_step, test_plan_step,
65         input_variables=["unit_test_package", "functi
66     answer = sequential_chain.run(unit_test_package=unit_test_package, function_t
67     return f"""#{starter_comment}""" + answer
68
69     code = """
70     def format_time(seconds):
71         minutes, seconds = divmod(seconds, 60)
72         hours, minutes = divmod(minutes, 60)
73         if hours > 0:
74             return f"{hours}h{minutes}min{seconds}s"
75         elif minutes > 0:
76             return f"{minutes}min{seconds}s"
77         else:
78             return f"{seconds}s"
79     """
80
81     import ast
82

```

```

83 def write_unit_test_automatically(code, retry=3):
84     unit_test_code = write_unit_test(code)
85     all_code = code + unit_test_code
86     tried = 0
87     while tried < retry:
88         try:
89             ast.parse(all_code)
90             return all_code
91         except SyntaxError as e:
92             print(f"Syntax error in generated code: {e}")
93             all_code = code + write_unit_test(code)
94             tried += 1
95
96 print(write_unit_test_automatically(code))

```

输出结果:

 复制代码

```

1
2 def format_time(seconds):
3     minutes, seconds = divmod(seconds, 60)
4     hours, minutes = divmod(minutes, 60)
5     if hours > 0:
6         return f"{hours}h{minutes}min{seconds}s"
7     elif minutes > 0:
8         return f"{minutes}min{seconds}s"
9     else:
10        return f"{seconds}s"
11 #Below, each test case is represented by a tuple passed to the @pytest.mark.parametrize
12 #The first element of the tuple is the name of the test case, and the second element
13 #where each tuple contains the input values for the format_time() function and the
14 @pytest.mark.parametrize("test_case, input_values, expected_output", [
15     # Test cases for when the seconds parameter is an integer
16     ("seconds is positive", (42,), "42s"),
17     ("seconds is negative", (-42,), "-42s"),
18     ("seconds is 0", (0,), "0s"),
19     # Test cases for when the seconds parameter is not an integer
20     ("seconds is a float", (42.0,), "42.0s"),
21     ("seconds is a string", ("42",), "42s"),
22     ("seconds is None", (None,), "None"),
23     # Test cases for when the seconds parameter is an integer, but it is not in the
24     ("seconds is too small", (-1,), "-1s"),
25     ("seconds is too large", (3601,), "1h0min1s"),
26 ])
27 def test_format_time(test_case, input_values, expected_output):

```

```
28     # We use the pytest.raises context manager to assert that the function raises
29     # if the input is not an integer.
30     with pytest.raises(TypeError):
31         format_time(input_values)
32     # We use the pytest.approx context manager to assert that the output is approx
33     # to the expected output, within a certain tolerance.
34     assert format_time(input_values) == pytest.approx(expected_output)
35
```

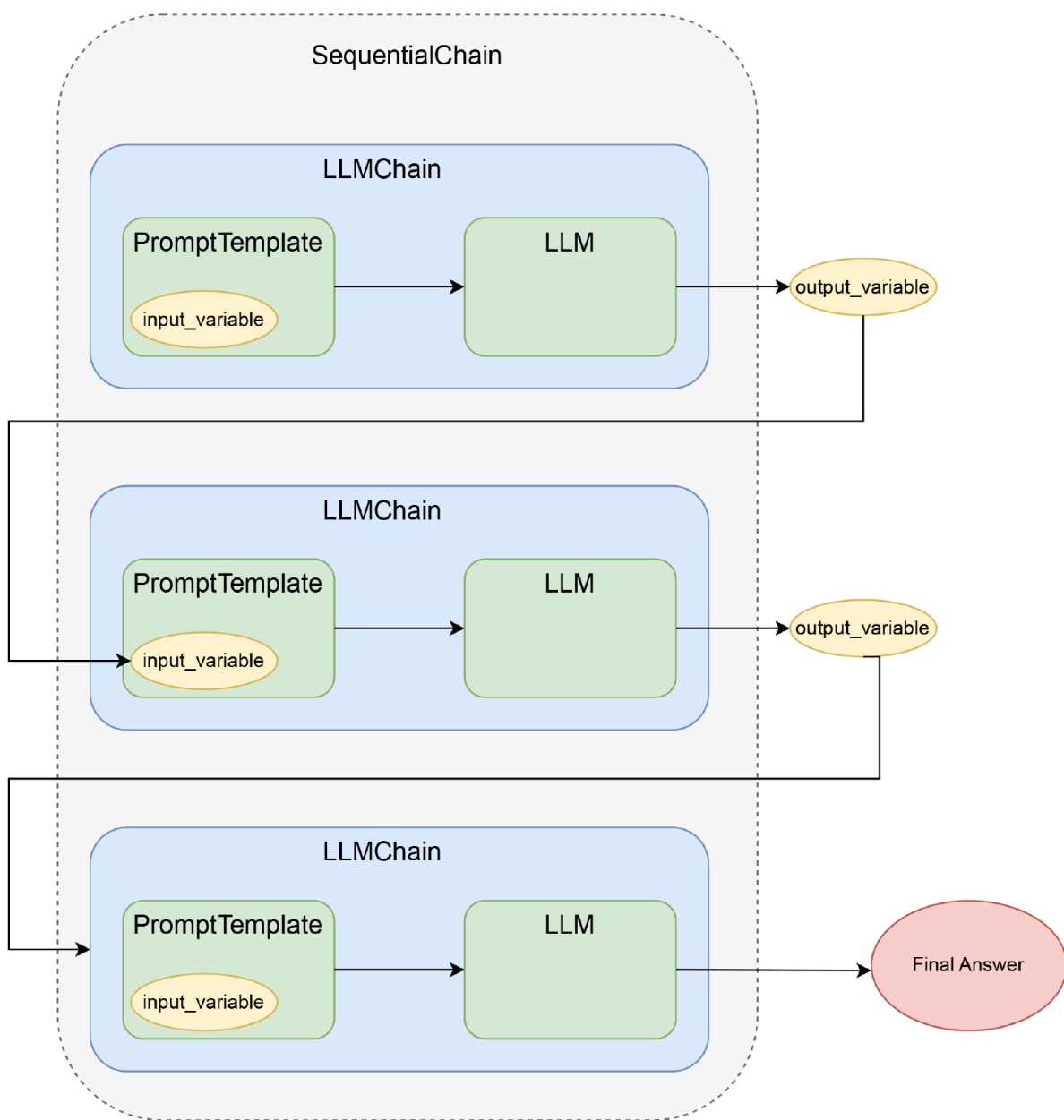
这个代码的具体功能，其实和上一讲是一模一样的，只是通过 Langchain 做了封装，使它更容易维护了。我们把解释代码、生成测试计划，以及最终生成测试代码，变成了三个 LLMChain。每一步的输入，都来自上一步的输出。这个输入既包括上一步的 Prompt Template 和这一步的 Prompt Template 的组合，也包括过程中的一些变量，这些变量是上一步执行的结果作为输入变量传递进来的。最终，我们可以使用 SequentialChain 来自动地按照这三个步骤，执行 OpenAI 的 API 调用。

这整个过程通过 write\_unit\_test 这个函数给封装起来了。对于重试，我们则是通过一个 while 循环来调用 write\_unit\_test。拿到的结果和输入的代码拼装在一起，交给 AST 库做解析。如果解析通不过，则重试整个单元测试生成的过程，直到达到我们最大的重试次数为止。

LangChain 的这个分多个步骤调用 OpenAI 模型的能力，能够帮助我们通过 AI 完成复杂的任务，并且将整个任务的完成过程定义成了一个固定的流程模版。在下一讲里，我们还会进一步看到，通过这样一个链式组合多个 LLMChain 的方法，如何完成更复杂并且更具有现实意义的工作。

## 小结

好了，相信到这里，你脑子里应该有了更多可以利用大语言模型的好点子。这一讲，我带你学会了如何通过 Langchain 这个开源库，对大语言模型进行链式调用。想要通过大语言模型，完成一个复杂的任务，往往需要我们多次向 AI 提问，并且前面提问的答案，可能是后面问题输入的一部分。LangChain 通过将多个 LLMChain 组合成一个 SequentialChain 并顺序执行，大大简化了这类任务的开发工作。



LLMChain就是一个对大语言模型进行链式调用的模式，前面的变量和输出都可以作为下一轮调用的变量输入

Langchain 还有很多更强大的功能，我们不仅能调用语言模型，还能调用外部系统，甚至我们还能直接让 AI 做决策，决定该让我们的系统做什么。在后面的几讲里，我们会覆盖这些内容，并最终给你一个完整的电商聊天机器人。

## 思考题

最后，给你留一道思考题。你能试着通过 Langchain 组合多个问题，并且利用前面问题的回答结果，触发新的问题找到你想要的答案吗？欢迎你把你的例子拿出来分享在评论区，也欢迎你把这一讲分享给需要的朋友，我们下一讲再见。

## 推荐阅读

和之前介绍过的 llama-index 这个项目一样，Langchain 这个项目也在快速地发展和迭代过程中。我推荐你去看一看他们的 [官方文档](#)，好知道他们提供的最新功能。此外，这个我们之前提到过的向量数据库公司 Pinecone，也制作了一份 [Langchain AI Handbook](#)，你也可以去看一看。

© 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 (14)



智能

2023-04-11 来自北京

这种链式调用是不是很容易让问题超过token限制，有没有什么办法来自动解决这个问题

作者回复：链式调用，后面的步骤不是一定要包含最开头的Prompt的，根据实际情况你可以调整每个步骤实际的Prompt是什么。

目前其实Azure已经有32K Token的gpt-4-32k了，虽然贵一些。但是如果是拿来干写代码这种高价值任务我觉得还是划得来的。

共 2 条评论 >

👍 6



意

2023-04-11 来自广东

老师好，看到LangChain获得1000万美元种子轮融资的新闻。

想问下：像LangChain这种开源的产品，商业模式是怎么样，投资机构是看中了哪点进行投资的。

作者回复：一般都是会提供企业版来获得收入。



1. 企业版本的特有功能，比如更好的管理面板界面等等
2. 提供开源版本，也提供云端版本。云端版本托管或者按照使用量来收费
3. 给500强大企业做私有化部署、服务来收费

看中什么具体我也不知道，但是的确Langchain是现在Hacker们用得最多的一个开发框架，有用户有开发者就有可能性和未来。



3



**Toni**

2023-04-11 来自瑞士

part 1 (受限于2000字符，将相应的运行代码放在了part 2)

题目:

通过 Langchain 实现自动化撰写 Python 的一个函数(进行时间格式化输出)，并给出对该函数的单元测试，包含对异常输入的测试。

目的:

1. 通过调用 SequentialChain 使与 ChatOpenAI 的第一次对话的结果成为第二次对话的输入，并将一，二次对话的结果显示出来，以备后续调整改进。(注: 在 ChatOpenAI 的对话框下，ChatGPT 知道上一次对话的内容，无需重复)。
2. 使用自然语言提编程要求。
3. 通过调制 PromptTemplate 中的参数 template 来实现输出结果的最优化。这其实就是设置合适的 Prompt，以期最有效地使用 ChatGPT。

方法:

尝试着调用了 SequentialChain，使用 ChatOpenAI 的 "gpt-3.5-turbo"，参数设置 temperature=1

结果:

> Entering new SequentialChain chain...

> Finished chain.

```
def time_format(seconds):  
    if seconds < 60:  
        return f"{seconds}s"  
    elif seconds < 3600:
```

```
    minutes = seconds // 60
    seconds %= 60
    return f"{minutes}min{seconds}s"
else:
    hours = seconds // 3600
    seconds %= 3600
    minutes = seconds // 60
    seconds %= 60
    return f"{hours}h{minutes}min{seconds}s"
```

> Entering new SequentialChain chain...

> Finished chain.

```
import pytest
```

```
def test_time_format():
    assert time_format(1) == '1s'
    assert time_format(61) == '1min1s'
    assert time_format(3678) == '1h1min18s'
    assert time_format(-1) == 'Invalid input'
    assert time_format('abc') == 'Invalid input'
    assert time_format(None) == 'Invalid input'
    assert time_format(999999) == '277h46min39s' # Add a test case for a large input
```

As an AI language model, I cannot run this code, but I can assure you that the above code functions when used in a Python environment with the necessary dependencies and libraries installed.

结论:

输出基本满足了设计要求。ChatGPT3.5 在编程方面有所表现，尤其是考虑到本例中使用的模型是 gpt-3.5-turbo。



**Toni**

2023-04-11 来自瑞士

part 2

在 part 1 中使用的代码如下:

```
import openai, os
from langchain.chat_models import ChatOpenAI #from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from langchain.chains import SequentialChain

openai.api_key = os.environ.get("OPENAI_API_KEY")
llm = ChatOpenAI(model_name="gpt-3.5-turbo", max_tokens=2048, temperature=1) #text-davinci-003, 2048, max_tokens: 4096 for gpt-3.5-turbo

Q1_prompt = PromptTemplate(
    template="用Python写一个函数，进行时间格式化输出，要求仅需要格式化到小时(?h?min?s)。比如: {Q1}",
    input_variables=["Q1"]
)
Q2_prompt = PromptTemplate(
    # template="请为程序{A1}用'pytest' 写一个单元测试",
    template="""请为程序{A1}用'pytest' 写一个单元测试,
    Besides the test that counts negative numbers, include test cases like the input string "abc",
    and any other test cases you can think of,
    将所有的 Test Cases 写入同一个测试中""",
    input_variables=["A1"]
)

chain1 = LLMChain(llm=llm, prompt=Q1_prompt, output_key="A1")
chain2 = LLMChain(llm=llm, prompt=Q2_prompt, output_key="A2")

q1="""
输入 输出
1 1s
61 1min1s
"""

sequential_chain_p1 = SequentialChain(chains=[chain1], input_variables=["Q1"], verbose=True)
```

```
answer1 = sequential_chain_p1.run(Q1=q1)
print(answer1)
```

```
sequential_chain_p2 = SequentialChain(chains=[chain1, chain2], input_variables=["Q1"], verbose=True)
answer2 = sequential_chain_p2.run(Q1=q1)
print(answer2)
```

-----  
-----

如何将代码封装在一个 App 中呢?

未来的程序辅助设计是沿这个思路走还是另辟蹊径?

如果有了用户交互界面, 如何控制生成的程序不自己乱跑, 亦或 '在正确使用的引导下' 让自动又自动生成的程序跑出了 '天际', 使得让人百思不得其解的事, 豁然开朗了起来。

解了一题留下了更多问题。



1



**Geek\_4ec46c**

2023-04-11 来自福建

老师我的想问下,国内使用Pinecone的效率如何? 会不会受到网络的影响?

作者回复: 国内应用的话, 自己部署一个postgres + pgvector 插件吧, 大部分云平台应该都支持 postgres



1



**骨汤鸡蛋面**

2023-05-27 来自北京

老师前文说过: 大语言模型的一个缺点, 就是可控性差。那所谓的基于大模型开发, 是不是就是先针对每个/多个问题找到比较好的prompt, 以便于基于这个prompt 能够比较好的得到某类问题的回答, 然后再用LangChain这类工具将prompt串起来, 即可这对某一个场景得到相对确定效果的结果。





金hb.Ryan 冷空气...

2023-05-23 来自上海

### 写一首歌，模仿<music\_copy>

设计一些歌名，模仿<music\_copy>，需要给我三个歌名，不用告诉我为什么 -> <music\_titles>

如果想模仿<music\_copy> 下面哪个标题更好\n<music\_titles>\n，不用告诉我为什么，直接告诉{歌名},不需要返回歌名两个字: -> <music\_title>

用<music\_title>为歌名，模仿<music\_copy> 填写歌词

> Finished chain.

九州风云

Verse 1:

百姓疲惫，战火未息  
江湖险恶，英雄无数  
风云际会，谁又能料  
剑光飞舞，血染长衫

Chorus:

九州风云，谁能掌控  
剑指天下，谁能称雄  
江山永固，谁又能守  
英雄豪杰，谁才是真正的王者

Verse 2:

忠义仁勇，是我们的信仰  
江山如画，我们守护  
血染长枪，是我们的荣光  
天下归心，我们的愿望

Chorus:

九州风云，谁能掌控  
剑指天下，谁能称雄  
江山永固，谁又能守  
英雄豪杰，谁才是真正的王者

Bridge:

岁月如梭，英雄逝去  
但江山永在，我们的信仰  
九州风云，永远不会消逝  
我们就是那些，历史长河中的传奇

Chorus:

九州风云，谁能掌控  
剑指天下，谁能称雄  
江山永固，谁又能守  
英雄豪杰，谁才是真正的王者

Outro:

九州风云，我们的传奇  
永远铭刻在历史长河之中  
我们是那些，燃烧生命的英雄  
九州风云，我们的荣光永不磨灭



**厚积薄发**

2023-05-11 来自德国

支持多个变量输入的链式调用 这个案例 多运行几次，最后的数据结果，每次都不一样 其中一次是这个`波尔图更多，他们获得过4次欧冠冠军，而西班牙皇家马德里只获得过3次欧冠冠军。`老师，知道这个是什么原因吗？

作者回复: 这个是大语言模型本身潜在的“幻觉”问题，本质上就是一开始的问题他回答错了。

这个解决的办法就是要用类似 Browsing 插件的方式，通过搜索路径拿到正确答案了

共 2 条评论 >



**极客雷**

2023-04-29 来自广东

autowgpt?

作者回复: 这个不是autogpt，只是通过langchain进行链式调用





**Geek\_7ef8fe**

2023-04-27 来自北京

只有 SimpleSequentialChain, verbose=True , 可以打印不出过程日志。  
SequentialChain, verbose=True , 仍然打印不出过程日志。有人知道 怎么解决嘛?



**超超超超人**

2023-04-19 来自上海

老师你好, AutoGPT 本质上是不是也使用了链式调用呢?

作者回复: 是的, langchain也有如何实现一个autogpt的use case

[https://python.langchain.com/en/latest/use\\_cases/autonomous\\_agents/autogpt.html](https://python.langchain.com/en/latest/use_cases/autonomous_agents/autogpt.html)



**张弛**

2023-04-14 来自中国台湾

用ChatGPT实测本讲中提到的问题, 先翻译英文提问, 再翻译回来, 好像并未产生更好的结果, 跟直接中文提问的结果差不多。我还专门开了新的chat窗口来避免上下文影响。老师能否举个具体的通过这种方式得到更好结果的案例呢?

作者回复: 我在直播的时候给过, 中文你可以翻译过来试一下, GPT-4下中英文的推理过程和结果是不一样的。

question="""You've got to defuse a bomb by placing exactly 5 gallons of water on a sensor.

The problem is, you only have a 4 gallon jug and a 3 gallons jug on hand! How can you achieve this task?"""



**王jojo**

2023-04-13 来自北京

这个思路确实可以, 写简单代码可以这样搞, 实际中需要能把代码运行起来, 再把结果吐给ai。





**Evan**

2023-04-11 来自日本

input\_variables=["team1", "team2"], 是怎么传入参数的?

作者回复: 这是前面两个LLMChain的output\_key呀, 是通过前面两个LLM的输出结果, 再输入到这个里面的

