LangChain 介绍与模型组件

安利一波掘金的LangChain 实战课，目前已经更新完毕，主要介绍了如何使用LangChain + LLM 的全新开发范式，由浅至深的介绍了LangChain的原理和应用，并通过实战样例帮助读者增强掌握程度。

LangChain

- 什么是LangChain：LangChain可以通过API调用如 ChatGPT、Llama等大型语言模型，还可以实现数据感知，将语言模型将其他数据源连接起来，并且与环境进行交互，使得模型能够对其环境有更深入的理解。

Langchain可以实现文本到图像的生成、文档问答、聊天机器人等。LangChain提供了一系列工具、套件和接口，可以简化创建由LLMs和聊天模型提供支持的应用程序的过程。LangChain包含6个组件：

- **模型（Models）**，包含各大语言模型的LangChain接口和调用细节，以及输出解析机制。
- **提示模板（Prompts）**，使提示工程流线化，进一步激发大语言模型的潜力。
- **数据检索（Indexes）**，构建并操作文档的方法，接受用户的查询并返回最相关的文档，轻松搭建本地知识库。
- **记忆（Memory）**，通过短时记忆和长时记忆，在对话过程中存储和检索数据，让ChatBot记住你是谁。
- **链（Chains）**，是LangChain中的核心机制，以特定方式封装各种功能，并通过一系列的组合，自动而灵活地完成常见用例。
- **代理（Agents）**，通过“代理”让大模型自主调用外部工具和内部工具，使强大的“智能化”自主Agent成为可能！

安装：

```
pip install langchain[llms]
pip install --upgrade langchain
```

OpenAI API

OpenAI API主要提供两类模型，Chat Model（比如ChatGPT和GPT-4）和Text Model（GPT3）。这两种模型都能接受输入prompt并返回文本。

- 调用Text模型

```
import os
```

```python
os.environ["OPENAI_API_KEY"] = '你的Open API Key'
# 或者 export OPENAI_API_KEY='你的Open API Key'
# 导入OpenAI库，并创建一个Client。
from openai import OpenAI
client = OpenAI()
# 指定 gpt-3.5-turbo-instruct（也就是 Text 模型）并调用 completions 方法，返回结果。
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  temperature=0.5,
  max_tokens=100,
  prompt="请给我的花店起个名")
```

序号	参数	含义
1	model	模型的类型，这里的 text-davinci-003 是最强大的 GPT-3 版 Text 模型。 早期的版本还包括： text-curie-003：这也是一个 GPT-3 模型，但比 davinci 小一些，它的性能较差，但速度更快、成本更低。 text-babbage-003：这是一个更小的 GPT-3 模型，用于需要快速响应和较低成本的任务。 text-ada-003：这是最小的GPT-3模型，通常用于大规模筛选或初步处理任务。
2	prompt	提示，也就是输入给模型的问题或者指示，告诉模型我们要它做什么。
3	temperature	这个参数可以影响模型输出的随机性。值越高（接近1），输出就越随机；值越低（接近0），输出就越确定。 例如，如果你设置 temperature=0.8，模型会有较高的可能性生成不同的输出； 如果你设置 temperature=0.2，模型生成的输出就会更加一致，更有可能重复相同的输出。
4	max_tokens	这个参数可以限制模型输出的最大长度，注意这个长度是以 Tokens 为单位的， 而一个 Token 可以是一个字、一个词或一个字符，这取决于语言和模型。
5	suffix	这个参数允许用户为模型生成的输出文本后附加一个后缀。 例如，如果你想让每次模型生成的文本都以某种特定格式或标记结尾，你可以使用这个参数。 默认情况下，它设置为 null，意味着没有后缀将被添加。如果指定了一个字符串作为此参数的值，那么该字符串 将被追加到每个输出文本的末尾。
6	top_p	这是与 temperature 参数类似的另一个参数，它使用所谓的核心抽样。模型将只考虑概率质量最高的 Tokens。 例如，top_p 设置为 0.1 意味着只有前 10% 的最有可能的 Tokens 会被考虑。
7	n	这个参数决定了为每个提示生成多少个完整的输出。需要注意的是，使用这个参数可能会更快地耗尽你的 Token 配额。
8	stream	这个参数决定是否实时流式传输生成的 Tokens。如果设置为 true，Tokens 将在生成时被发送。
9	logprobs	这个参数要求 API 包括最有可能的 Tokens 的对数概率。 例如，如果 logprobs 设置为 5，API 将返回最有可能的 5 个 Tokens 的列表。
10	echo	如果设置为 true，除了生成的完成内容外，还会回显提示。
11	stop	这个参数允许你指定一个或多个序列，当模型遇到这些序列时，它会停止生成 Tokens，返回的文本将不包含 stop 序列。
12	presence_penalty	这是一个在 -2.0 到 2.0 之间的数字。 正值会惩罚已经出现在文本中的新 Tokens，这可以帮助模型更多地谈论新话题。
13	frequency_penalty	这也是一个在 -2.0 到 2.0 之间的数字。 正值会惩罚到目前为止在文本中频繁出现的 Tokens，这可以减少模型重复相同内容的可能性。
14	best_of	这个参数会在服务器端生成 best_of 个完整的输出，并返回其中最好的一个（即每个 Token 的对数概率最高的那个）。 与 n 参数结合使用时，best_of 决定了候选完成的数量，而 n 决定了返回的数量。
15	logit_bias	这个参数可以修改指定 Tokens 在完成中出现的可能性。 接受一个 JSON 对象，该对象将 Tokens 映射到 -100 到 100 之间的偏置值。
16	user	这是一个可选参数，表示你的最终用户的唯一标识符，可以帮助 OpenAI 监测和检测滥用。

```
# 从响应中获取第一个（如果在调用大模型时，没有指定n参数，那么就只有唯一的一个响应）选择，然后获取该选择的文本，并移除其前后的空白字符。
print(response.choices[0].text.strip())
# 花漾时光、花语梦境、繁花小筑
```

序号	字段	含义
1	id	响应的唯一标识符。
2	object	表示该响应的对象类型，对于生成操作，这个字段通常为 text_completion。
3	created	表示响应生成的时间。
4	model	表示生成响应的模型的名称。
5	choices	一个列表，其中包含了模型生成的所有输出。 除非指定 n 参数，否则通常只包含一个条目，即索引 [0]。
6	usage	提供了关于文本生成过程的统计信息，包括 prompt_tokens（提示的 Token 数量）、completion_tokens（生成的 Token 数量）、total_tokens（总的 Token 数量）。

- 调用Chat模型

```
response = client.chat.completions.create(
  model="gpt-4",
  messages=[
        {"role": "system", "content": "You are a creative AI."},
        {"role": "user", "content": "请给我的花店起个名"},
    ],
  temperature=0.8,
  max_tokens=60
)
```

这里边中的message是一个列表，包含多条消息，每个消息包含一个role和content。role包含以下角色：

1. system：系统消息主要用于设定对话的背景或上下文，这可以帮助模型理解它在对话中的角色和任务。
2. user：用户消息是从用户或人类角色发出的，包含用户的问题。
3. assistant：助手消息是模型的回复。例如，在你使用API发送多轮对话中新的对话请求时，可以通过助手消息提供先前对话的上下文。然而，请注意在对话的最后一条消息应始终为用户消息，因为模型总是要回应最后这条用户消息。

Chat模型更适合处理对话或者多轮次交互的情况，并且可以设置对话场景，给模型提供额外的指导信息。

通过LangChain调用OpenAI

调用 Text 模型

```
import os
os.environ["OPENAI_API_KEY"] = '你的Open API Key'
from langchain.llms import OpenAI
llm = OpenAI(
    model="gpt-3.5-turbo-instruct",
    temperature=0.8,
    max_tokens=60,)
response = llm.predict("请给我的花店起个名")
print(response)
```

调用 Chat 模型

```
import os
os.environ["OPENAI_API_KEY"] = '你的Open API Key'
from langchain.chat_models import ChatOpenAI
chat = ChatOpenAI(model="gpt-4",
                  temperature=0.8,
                  max_tokens=60)
from langchain.schema import (
    HumanMessage,
    SystemMessage
)
messages = [
    SystemMessage(content="你是一个很棒的智能助手"), # 系统消息
    HumanMessage(content="请给我的花店起个名") # 用户消息
]
response = chat(messages)
print(response)
```

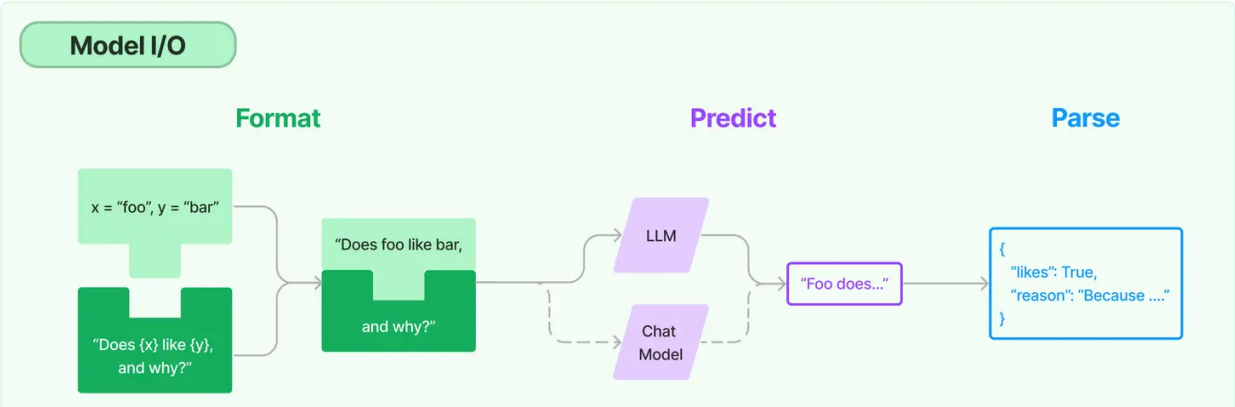
基础组件

Model

模型位于LangChain框架的最底层，Langchain的本质就是通过API调用大模型解决问题。

I/O

模型的使用过程拆解成三块，分别是输入提示（Format）、调用模型（Predict）和输出解析（Parse）。



输入提示

为了更好的将提示信息输入到模型中，可以创建提示模板。提示工程要求给模型清晰明确的提示，让模型逐步去思考。例如：

```
# 导入LangChain中的提示模板
from langchain.prompts import PromptTemplate
# 创建原始模板
template = """您是一位专业的鲜花店文案撰写员。\n
对于售价为 {price} 元的 {flower_name} ，您能提供一个吸引人的简短描述吗?
"""
# 根据原始模板创建LangChain提示模板
prompt = PromptTemplate.from_template(template)
# 打印LangChain提示模板的内容
print(prompt)
```

语言模型

第六行将模板实例化，通过服用提示模板可以批量生成答案。只需要定义一次模板，就能生成不同的提示。

```
# 导入LangChain中的OpenAI模型接口
from langchain_openai import OpenAI
# 创建模型实例
model = OpenAI(model_name='gpt-3.5-turbo-instruct')
# 输入提示
input = prompt.format(flower_name=["玫瑰"], price='50')
# 得到模型的输出
output = model.invoke(input)
# 打印输出内容
print(output)
```

输出解析

希望模型的返回按照一定格式，比如包含多个字段。下面代码中，要求模型输出description和reason，通过StructuredOutputParser.from_response_schemas方法创建了一个输出解析器，通过输出解析器对象的get_format_instructions()方法获取输出的格式说明，根据原始的字符串模板和输出解析器格式说明创建新的提示模板

```
# 导入结构化输出解析器和ResponseSchema
from langchain.output_parsers import StructuredOutputParser, ResponseSchema
# 定义我们想要接收的响应模式
response_schemas = [
    ResponseSchema(name="description", description="鲜花的描述文案"),
    ResponseSchema(name="reason", description="问什么要这样写这个文案")
]
# 创建输出解析器
output_parser = StructuredOutputParser.from_response_schemas(response_schemas)

# 获取格式指示
format_instructions = output_parser.get_format_instructions()
# 根据原始模板创建提示，同时在提示中加入输出解析器的说明
prompt = PromptTemplate.from_template(prompt_template,
                partial_variables={"format_instructions":
format_instructions})
```

总结一下使用LangChain框架的好处，你会发现它有这样几个优势。

1. 模板管理：在大型项目中，可能会有许多不同的提示模板，使用 LangChain 可以帮助你更好地管理这些模板，保持代码的清晰和整洁。
2. 变量提取和检查：LangChain 可以自动提取模板中的变量并进行检查，确保你没有忘记填充任何变量。
3. 模型切换：如果你想尝试使用不同的模型，只需要更改模型的名称就可以了，无需修改代码。
4. 输出解析：LangChain的提示模板可以嵌入对输出格式的定义，以便在后续处理过程中比较方便地处理已经被格式化了的输出。