# 4.1 Connect LLM to LangChain

The source files involved in the text can be obtained from the following path:

> - LLM access to LangChain.ipynb
> - wenxin_llm.py
> - zhipuai_llm.py

---

LangChain provides an efficient development framework for developing custom applications based on LLM, which allows developers to quickly stimulate the powerful capabilities of LLM and build LLM applications. LangChain also supports a variety of large models, and has built-in calling interfaces for large models such as OpenAI and LLAMA. However, LangChain does not have all large models built in. It provides strong scalability by allowing users to customize LLM types.

## 1. Calling ChatGPT based on LangChain

LangChain provides encapsulation for a variety of large models. Based on the LangChain interface, ChatGPT can be easily called and integrated into personal applications built on the LangChain framework. Here we briefly describe how to use the LangChain interface to call ChatGPT.

Note that calling ChatGPT based on the LangChain interface also requires configuring your personal key, and the configuration method is the same as above.

### 1.1 Models

`langchain.chat_models` Import `OpenAI` the dialogue model from `ChatOpenAI` . In addition to OpenAI, `langchain.chat_models` other dialogue models are also integrated. For more details, please refer to **the Langchain official documentation** .

python

```python
import os
import openai
from dotenv import load_dotenv, find_dotenv

# 读取本地/项目的环境变量。

# find_dotenv()寻找并定位.env文件的路径
# load_dotenv()读取该.env文件，并将其中的环境变量加载到当前的运行环境中
# 如果你设置的是全局的环境变量，这行代码则没有任何作用。
_ = load_dotenv(find_dotenv())

# 获取环境变量 OPENAI_API_KEY
openai_api_key = os.environ['OPENAI_API_KEY']
```

If langchain-openai is not installed, please run the following code first!

python

```python
from langchain_openai import ChatOpenAI
```

Next you need to instantiate a ChatOpenAI class, where you can pass in hyperparameters to control the answer, such as `temperature` parameters.

python

```python
# 这里我们将参数temperature设置为0.0，从而减少生成答案的随机性。
# 如果你想要每次得到不一样的有新意的答案，可以尝试调整该参数。
llm = ChatOpenAI(temperature=0.0)
llm
```

markup

```
ChatOpenAI(client=<openai.resources.chat.completions.Completions object a
```

The above cell assumes that your OpenAI API key is set in an environment variable. If you wish to manually specify your API key, use the following code:

python

```python
llm = ChatOpenAI(temperature=0, openai_api_key="YOUR_API_KEY")
```

As you can see, the ChatGPT-3.5 model is used by default. In addition, several commonly used hyperparameter settings include:

```markup
· model_name：所要使用的模型，默认为 ‘gpt-3.5-turbo’，参数设置与 OpenAI 原生接口
· temperature：温度系数，取值同原生接口。
· openai_api_key：OpenAI API key，如果不使用环境变量设置 API Key，也可以在实例化
· openai_proxy：设置代理，如果不使用环境变量设置代理，也可以在实例化时设置。
· streaming：是否使用流式传输，即逐字输出模型回答，默认为 False，此处不赘述。
· max_tokens：模型输出的最大 token 数，意义及取值同上。
```

Once we have initialized your selection `LLM` , we can try using it! Let's ask "Please tell me about yourself!"

```python
output = llm.invoke("请你自我介绍一下自己！")
```

```python
output
```

```markup
AIMessage(content='你好，我是一个智能助手，专注于为用户提供各种服务和帮助。我可以回
```

## 1.2 Prompt (Prompt Template)

When we develop large model applications, most of the time we don't pass user input directly to the LLM. Usually, they will add the user input to a larger text, called a prompt 提示模板 , which provides additional context about the specific task at hand. PromptTemplates help with this! They bundle all the logic from user input to a fully formatted prompt. This can start very simple - for example, the prompt that generates the above string is:

We need to construct a personalized Template first:

```python
from langchain_core.prompts import ChatPromptTemplate

# 这里我们要求模型对给定文本进行中文翻译
prompt = """请你将由三个反引号分割的文本翻译成英文！\
text: ```{text}```
"""
```

Next, let's take a look at the complete prompt template that has been constructed:

```python
text = "我带着比身体重的行李，\
游入尼罗河底，\
经过几道闪电 看到一堆光圈，\
不确定是不是这里。\
"
prompt.format(text=text)
```

```markup
'请你将由三个反引号分割的文本翻译成英文！text: ```我带着比身体重的行李，游入尼罗河底，
```

We know that the chat model interface is based on messages, not raw text. PromptTemplates can also be used to generate a list of messages, in this case `prompt` containing not only the input content information, but also each `message` message (role, position in the list, etc.). Usually, a `ChatPromptTemplate` is a `ChatMessageTemplate` list of . Each `ChatMessageTemplate` contains instructions for formatting the chat message (its role and content).

Let's look at an example:

```python
from langchain.prompts.chat import ChatPromptTemplate

template = "你是一个翻译助手，可以帮助我将 {input_language} 翻译成 {output_lang
human_template = "{text}"

chat_prompt = ChatPromptTemplate.from_messages([
```

```
    ("system", template),
    ("human", human_template),
])

text = "我带着比身体重的行李，\
游入尼罗河底，\
经过几道闪电 看到一堆光圈，\
不确定是不是这里。\
"
messages  = chat_prompt.format_messages(input_language="中文", output_lang
messages
```

markup

```
[SystemMessage(content='你是一个翻译助手，可以帮助我将 中文 翻译成 英文.'),
 HumanMessage(content='我带着比身体重的行李，游入尼罗河底，经过几道闪电 看到一堆光圈
```

Next, let's call the defined sum `llm` to `messages` output the answer:

python

```
output  = llm.invoke(messages)
output
```

markup

```
AIMessage(content='I carried luggage heavier than my body and dived into
```

## 1.3 Output parser

OutputParsers convert the raw output of a language model into a format that can be used downstream. There are several main types of OutputParsers, including:

- Convert LLM text to structured information (e.g. JSON)
- Convert ChatMessage to string
- Converts extra information returned by a call other than a message (such as an OpenAI function call) to a string

Finally, we pass the model output to StrOutputParser `output_parser`, which is a StrOutputParser `BaseOutputParser`, meaning it accepts **either a string or a BaseMessage as input**. The StrOutputParser in particular simply converts any input to a string.

python

```
from langchain_core.output_parsers import StrOutputParser

output_parser = StrOutputParser()
output_parser.invoke(output)
```

markup

```
'I carried luggage heavier than my body and dived into the bottom of the
```

From the above results, we can see that we successfully `ChatMessage` parsed the output of type into `字符串`

## 1.4 Complete Process

We can now combine all of this into a chain. This chain will take input variables, pass those variables to a prompt template to create a prompt, pass the prompt to a language model, and then pass the output through an (optional) output parser. Next we'll use the LCEL syntax to quickly implement a chain. Let's see it in action!

python

```
chain = chat_prompt | llm | output_parser
chain.invoke({"input_language":"中文", "output_language":"英文","text": te
```

markup

```
'I carried luggage heavier than my body and dived into the bottom of the
```

Let's test another example:

python

```
text = 'I carried luggage heavier than my body and dived into the bottom
```

```
chain.invoke({"input_language":"英文", "output_language":"中文","text": te
```

```
markup
```

```
'我扛着比我的身体还重的行李，潜入尼罗河的底部。穿过几道闪电后，我看到一堆光环，不确定这是
```

> **What is LCEL? LCEL (LangChain Expression Language) is a new syntax and an important addition to the LangChain toolkit. It has many advantages, making it easier and more convenient to handle LangChain and proxies.**

- LCEL provides asynchronous, batch, and stream processing support, allowing code to be quickly ported across different servers.
- LCEL has a fallback measure to solve the problem of LLM format output.
- LCEL increases the parallelism of LLM and improves its efficiency.
- LCEL has built-in logging, which helps to understand the operation of complex chains and agents even if the agents become complex.

Example usage:

```
chain = prompt | model | output_parser
```

In the code above, we use LCEL to piece together different components into a chain, in which the user input is passed to the prompt template, and then the prompt template output is passed to the model, and then the model output is passed to the output parser. The | symbol is similar to the Unix pipe operator, which links different components together, passing the output of one component as the input of the next component.

# 2. Use LangChain to call Baidu Wenxin Yiyan

We can also call Baidu Wenxin model through LangChain framework to integrate the Wenxin model into our application framework.

## 2.1 Customize LLM to access langchain

In the old version, LangChain does not directly support Wenxin calls, so we need to customize an LLM that supports Wenxin model calls. In order to show users how to customize LLM, we briefly

describe this method in "Appendix 1. LangChain Custom LLM", and you can also refer to **the source document** .

Here, we can directly call the customized Wenxin_LLM. See how to encapsulate Wenxin_LLM for details `wenxin_llm.py` .

**Note: The following code needs to download our encapsulated code wenxin_llm.py to the same directory as this Notebook before it can be used directly. Because the new version of LangChain can directly call the Wenxin Qianfan API, we recommend using the next part of the code to call the Wenxin Yiyan model**

python

```
# 需要下载源码
from wenxin_llm import Wenxin_LLM
```

We want to store the secret key directly in the .env file and load it into the environment variable like calling ChatGPT, so as to hide the specific details of the secret key and ensure security. Therefore, we need to configure `QIANFAN_AK` and in the .env file `QIANFAN_SK` and load it using the following code:

python

```
from dotenv import find_dotenv, load_dotenv
import os

# 读取本地/项目的环境变量。

# find_dotenv()寻找并定位.env文件的路径
# load_dotenv()读取该.env文件，并将其中的环境变量加载到当前的运行环境中
# 如果你设置的是全局的环境变量，这行代码则没有任何作用。
_ = load_dotenv(find_dotenv())

# 获取环境变量 API_KEY
wenxin_api_key = os.environ["QIANFAN_AK"]
wenxin_secret_key = os.environ["QIANFAN_SK"]
```

python

```
llm = Wenxin_LLM(api_key=wenxin_api_key, secret_key=wenxin_secret_key, sy
```

python

```
llm.invoke("你好，请你自我介绍一下！")
```

markup

```
[INFO] [03-31 22:12:53] openapi_requestor.py:316 [t:27812]: requesting ll
```

```
1
2
```

```
'你好！我是助手，负责协助您完成各种任务。我具备快速响应、高效执行和灵活适应的能力，致力
```

python

```
# 或者使用
llm(prompt="你好，请你自我介绍一下！")
```

markup

```
[INFO] [03-31 22:12:41] openapi_requestor.py:316 [t:27812]: requesting ll
```

```
1
2
```

```
'你好！我是助手，负责协助您完成各种任务。我具备快速学习和处理信息的能力，能够根据您的需
```

Therefore, we can add the Wenxin model to the LangChain architecture and implement the call of the Wenxin model in the application.

## 2.2 Call Wenxinyiyan directly in langchain

We can also use the new version of LangChain to directly call the Wenxin Yiyan model.

python

```python
from dotenv import find_dotenv, load_dotenv
import os

# 读取本地/项目的环境变量。

# find_dotenv()寻找并定位.env文件的路径
# load_dotenv()读取该.env文件，并将其中的环境变量加载到当前的运行环境中
# 如果你设置的是全局的环境变量，这行代码则没有任何作用。
_ = load_dotenv(find_dotenv())

# 获取环境变量 API_KEY
QIANFAN_AK = os.environ["QIANFAN_AK"]
QIANFAN_SK = os.environ["QIANFAN_SK"]
```

python

```python
# Install required dependencies
%pip install -qU langchain langchain-community
```

python

```python
from langchain_community.llms import QianfanLLMEndpoint

llm = QianfanLLMEndpoint(streaming=True)
res = llm("你好，请你自我介绍一下！")
print(res)
```

markup

```
d:\Miniconda\miniconda3\envs\llm2\lib\site-packages\langchain_core\_api\d
  warn_deprecated(
[INFO] [03-31 22:40:14] openapi_requestor.py:316 [t:3684]: requesting llm
[INFO] [03-31 22:40:14] oauth.py:207 [t:3684]: trying to refresh access_t
[INFO] [03-31 22:40:15] oauth.py:220 [t:3684]: sucessfully refresh access

你好！我是文心一言，英文名是ERNIE Bot。我是一款人工智能语言模型，可以协助你完成范围广泛
```

# 3. Use LangChain to call iFlytek Spark

We can also call iFlytek Spark LLM through LangChain framework. For more information, refer to [SparkLLM]

We want to store the secret key directly in the .env file and load it into the environment variable like calling ChatGPT, so as to hide the specific details of the secret key and ensure security. Therefore, we need to configure , `IFLYTEK_SPARK_APP_ID` and `IFLYTEK_SPARK_API_KEY` in the .env file `IFLYTEK_SPARK_API_SECRET` and load it using the following code:

python

```
from dotenv import find_dotenv, load_dotenv
import os

# 读取本地/项目的环境变量。

# find_dotenv()寻找并定位.env文件的路径
# load_dotenv()读取该.env文件，并将其中的环境变量加载到当前的运行环境中
# 如果你设置的是全局的环境变量，这行代码则没有任何作用。
_ = load_dotenv(find_dotenv())

# 获取环境变量 API_KEY
IFLYTEK_SPARK_APP_ID = os.environ["IFLYTEK_SPARK_APP_ID"]
IFLYTEK_SPARK_API_KEY = os.environ["IFLYTEK_SPARK_API_KEY"]
IFLYTEK_SPARK_API_SECRET = os.environ["IFLYTEK_SPARK_API_SECRET"]
```

python

```
def gen_spark_params(model):
    '''
    构造星火模型请求参数
    '''

    spark_url_tpl = "wss://spark-api.xf-yun.com/{}/chat"
    model_params_dict = {
        # v1.5 版本
        "v1.5": {
            "domain": "general", # 用于配置大模型版本
            "spark_url": spark_url_tpl.format("v1.1") # 云端环境的服务地址
        },
        # v2.0 版本
        "v2.0": {
            "domain": "generalv2", # 用于配置大模型版本
```

```
            "spark_url": spark_url_tpl.format("v2.1") # 云端环境的服务地址
        },
        # v3.0 版本
        "v3.0": {
            "domain": "generalv3", # 用于配置大模型版本
            "spark_url": spark_url_tpl.format("v3.1") # 云端环境的服务地址
        },
        # v3.5 版本
        "v3.5": {
            "domain": "generalv3.5", # 用于配置大模型版本
            "spark_url": spark_url_tpl.format("v3.5") # 云端环境的服务地址
        }
    }
    return model_params_dict[model]
```

python

```
from langchain_community.llms import SparkLLM

spark_api_url = gen_spark_params(model="v1.5")["spark_url"]

# Load the model(默认使用 v3.0)
llm = SparkLLM(spark_api_url = spark_api_url)  #指定 v1.5版本
```

python

```
res = llm("你好，请你自我介绍一下！")
print(res)
```

markup

```
您好，我是科大讯飞研发的认知智能大模型，我的名字叫讯飞星火认知大模型。我可以和人类进行自
```

Therefore, we can add the Spark model to the LangChain architecture and implement the call of the Wenxin model in the application.

# 4. Use LangChain to call Zhipu GLM

We can also call the Zhipu AI big model through the LangChain framework to connect it to our application framework. Since **the ChatGLM** provided in langchain is no longer available, we need

to customize a LLM.

If you are using Zhipu GLM API, you need to download our encapsulated code **zhipuai_llm.py** to the same directory as this Notebook before you can run the following code to use GLM in LangChain.

According to Zhipu's official announcement, the following models will be deprecated soon. After these models are deprecated, they will be automatically routed to new models. Please note that before the deprecation date, update your model code to the latest version to ensure a smooth transition of services. For more information about the model, please visit **model**

markup

```
| 模型编码 |弃用日期|指向模型|
| ---- | ---- | ---- |
|chatglm_pro|2024 年 12 月 31 日|glm-4|
|chatglm_std|2024 年 12 月 31 日|glm-3-turbo|
|chatglm_lite|2024 年 12 月 31 日|glm-3-turbo|
```

## 4.1 Customize chatglm to access langchain

python

```
# 需要下载源码
from zhipuai_llm import ZhipuAILLM
```

python

```
from dotenv import find_dotenv, load_dotenv
import os

# 读取本地/项目的环境变量。

# find_dotenv()寻找并定位.env文件的路径
# load_dotenv()读取该.env文件，并将其中的环境变量加载到当前的运行环境中
# 如果你设置的是全局的环境变量，这行代码则没有任何作用。
_ = load_dotenv(find_dotenv())

# 获取环境变量 API_KEY
api_key = os.environ["ZHIPUAI_API_KEY"] #填写控制台中获取的 APIKey 信息
```

python

```
zhipuai_model = ZhipuAILLM(model = "glm-4", temperature = 0.1, api_key =
```

python

```
zhipuai_model("你好，请你自我介绍一下！")
```

markup

```
'你好！我是智谱清言，是清华大学 KEG 实验室和智谱 AI 公司于 2023 年共同训练的语言模型。
```

## The source file acquisition path involved above is:

- 1.LLM access to LangChain.ipynb
- wenxin_llm.py
- zhipuai_llm.py

Next Chapter
2. Build a retrieval question-answer chain