

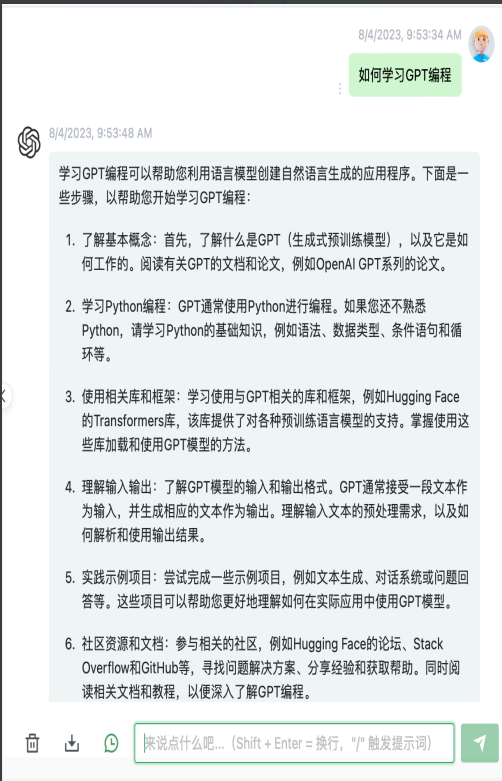
# AI 大模型企业应用实战

蔡超

# 内容综述

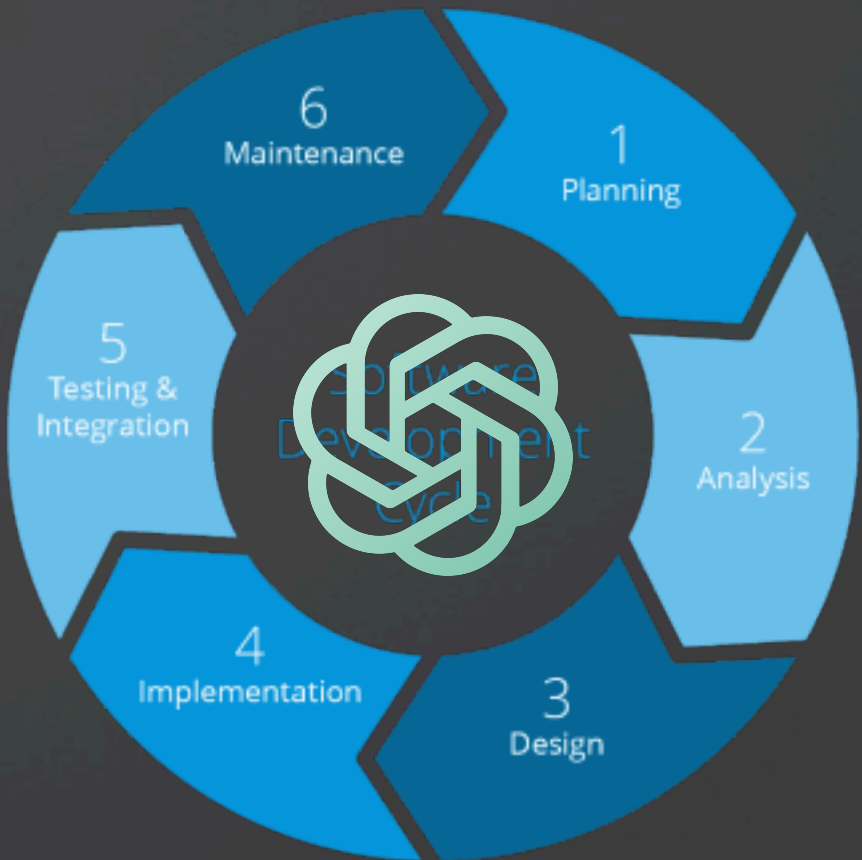
# 不只是聊天机器人

聊天机器人



企业智能应用的推理内核

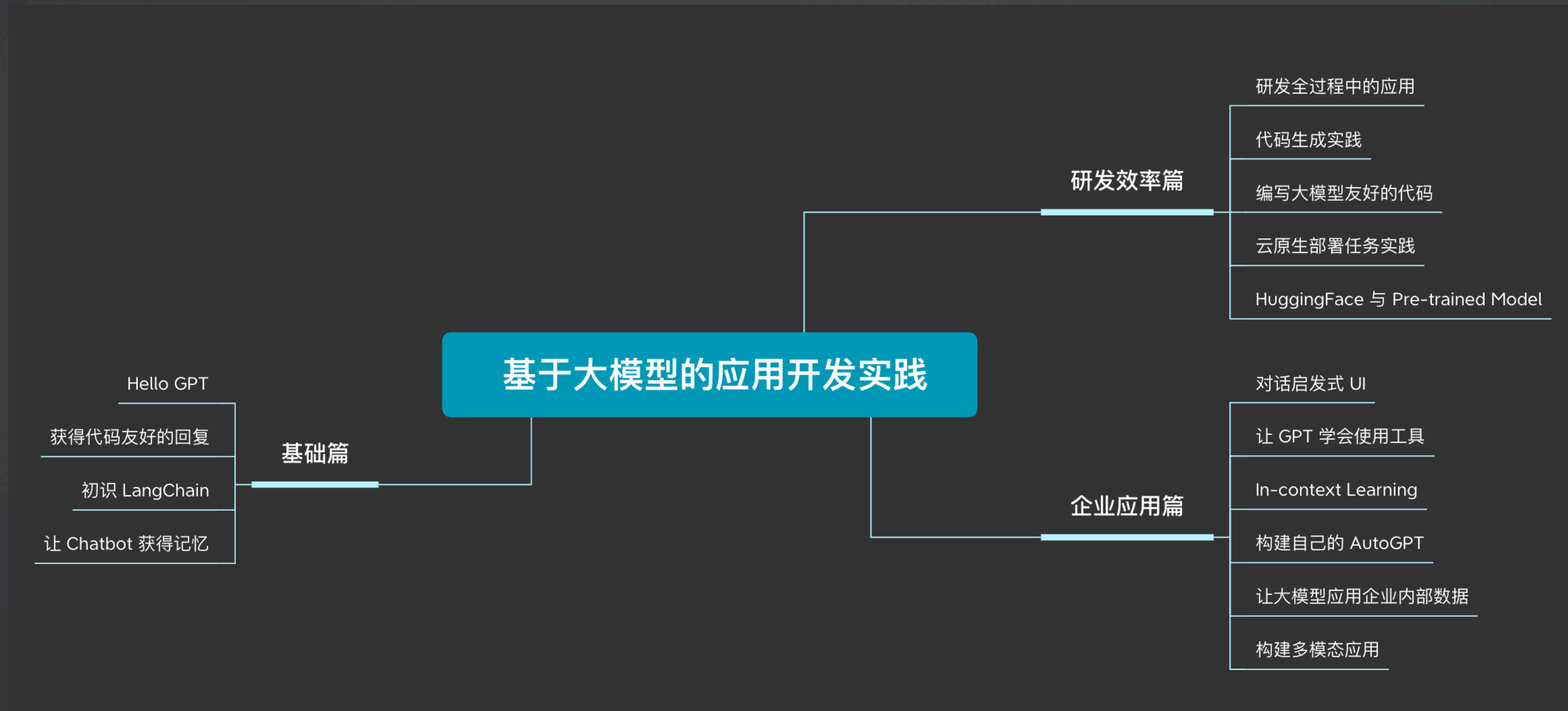
软件生命周期



软件开发中的硅基工程师



# 课程内容



# 适合人群

1. 希望在企业应用中落地大语言模型，并借此提升团队研发效率的软件工程师和架构师
2. 初中级开发者可以学到如何通过编程方式使用大模型，提高自身开发效率
3. 高级软件开发者和架构师，可以获得对未来软件的新思考，了解大模型带来的软件架构变革及新的基于大语言模型的开发范式
4. 需要了解一门编程语言，后续课程里会使用 Python 语言做演示

也欢迎你关注我的公众号：蔡超谈软件

# 第一章 | 基础知识篇



# 1 | 第一个大模型程序

Hello GPT



# 目录

- 0 构建测试环境
- 1 “Hello GPT”
- 2 Token: GPT 的基本文本处理单元

# 目录

- 0 构建测试环境
- 1 “Hello GPT”
- 2 Token: GPT 的基本文本处理单元

# 在 Azure 中部署模型

## 部署

通过部署，可以针对所提供的基本模型或微调模型进行完成和搜索调用。还可以通过修改缩放单元来轻松地纵向扩展和减少部署。

[+ 新建部署](#) [✎ 编辑部署](#) [🗑 删除部署](#) [🔑 列选项](#) [🔄 刷新](#) [🔗 在操场中打开](#)

部署名 ▾	模型名称 ▾	模.. ▾	De... ▾	Capac...	状态 ▾	Model dep... ▾
<input checked="" type="checkbox"/> <a href="#">txt</a>	text-davinci-003	1	Standard	60	✅ 已成	2024/7/5
<a href="#">embedding</a>	text-embedding-ada-002	2	Standard	120	✅ 已成	2025/2/2
<a href="#">gpt-35</a>	gpt-35-turbo	0613	Standard	120	✅ 已成	2024/7/5
<a href="#">gpt4</a>	gpt-4	0613	Standard	10	✅ 已成	2024/1/15



# Azure Deployments


## 部署

通过部署，可以针对所提供的基本模型或微调模型进行完成和搜索调用。还可以通过修改缩放单元来轻松地纵向扩展和减少部署。

[+ 新建部署](#) [✎ 编辑部署](#) [🗑 删除部署](#) [🔑 列选项](#) [🔄 刷新](#) [🔗 在操场中打开](#)

部署名 ▾	模型名称 ▾	模.. ▾	De... ▾	Capac...	状态 ▾	Model dep... ▾
<input checked="" type="checkbox"/> <a href="#">txt</a>	text-davinci-003	1	Standard	60	✅ 已成	2024/7/5
<a href="#">embedding</a>	text-embedding-ada-002	2	Standard	120	✅ 已成	2025/2/2
<div><div><a href="#">gpt-35</a></div><div>engine</div></div>	<div><div><a href="#">gpt-35-turbo</a></div><div>model</div></div>	0613	Standard	120	✅ 已成	2024/7/5
<a href="#">gpt4</a>	gpt-4	0613	Standard	10	✅ 已成	2024/1/15

# KEY 和 Endpoint

 **chao-openai** | Keys and Endpoint ☆ ...  
Azure OpenAI

[重新生成密钥1](#) [重新生成密钥2](#)

概述

活动日志


访问控制(标识和访问管理)

标记

诊断并解决问题


资源管理

- Keys and Endpoint**
- Model deployments
- 定价层
- Networking
- Identity
- Cost analysis
- 属性
- 锁


 这些订阅密钥用于访问认知服务 API。请勿共享密钥。安全地存储这些密钥，例如，使用 Azure Key Vault。另外，建议定期重新生成这些密钥。进行 API 调用只需要使用一个密钥。重新生成第一个密钥时，可以使用第二个密钥继续访问服务。

[Show Keys](#)


密钥 1

..... OPENAI\_API\_KEY 


密钥 2

..... 

Location/Region ⓘ

eastus 

终结点

https://chao-openai.openai.azure.com/ OPENAI\_API\_BASE 

# 环境构建

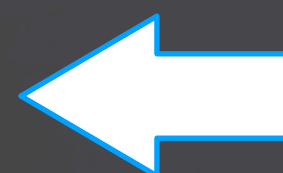
## 环境变量设置

```
export OPENAI_API_TYPE=azure
```

```
export OPENAI_API_VERSION=2023-08-01-preview
```

```
export OPENAI_API_BASE=https://<Your EndPoint>/
```

```
export OPENAI_API_KEY=<Your Key>
```



如果直接访问 OpenAI 提供的 GPT 服务仅需此配置

可以将以上环境变量加入到系统的启动配置中，如 mac OS 系统的 ~/.zshrc 文件中



# 构建测试环境

## 1. 安装 Jupyter Notebook

```
pip install notebook
```

## 2. 获取代码

```
git clone https://github.com/chaocai2001/gpt_in_practice.git
```

或者

```
git clone https://gitee.com/chao-superman/llm-in-practice.git
```

## 3. 运行 Jupyter Notebook

进入代码路径，执行：`jupyter notebook`

# 目录

- 0 构建测试环境
- 1 “Hello GPT”
- 2 Token: GPT 的基本文本处理单元

# Hello GPT

```
import openai
```

如果直接访问 OpenAI GPT 服务的同学，  
这里不要使用 engine 这个参数，  
要使用 model，如： model="gpt-4"

```
response = openai.ChatCompletion.create(  
    engine=deployment, # engine = "deployment_name"  
    messages=[  
        {"role": "system", "content": "You are an AI assistant."},  
        {"role": "user", "content": "How are you?"}  
    ],  
    temperature = 0.9,  
    max_tokens = 1000  
)  
print(response.choices[0].message.content)
```

system: 用于在这个会话中设置 AI 助手的  
个性或提供有关其在整个对话过程中  
应如何表现的具体说明

temperature:  
0-2 控制结果的随机性，  
值越大随机性越大

User: 用于向 AI 助理提出需求



# max\_tokens 限制

max\_token 用于限制生成内容的最大 token 数

另外，提示词的 token 数加上 max\_tokens 的值不能大于模型支持的内容长度

```
{
  "id": "chatcmpl-7x2VcDVyflmT0bxjphYMbgMd9O28l",
  "object": "chat.completion",
  ...
  "choices": [
    {
      "index": 0,
      "finish_reason": "length",
      "message": {
        "role": "assistant",
        "content": "\u4f60\u597d"
      },
      ...
    }
  ],
  "usage": {
    "completion_tokens": 2,
    "prompt_tokens": 30,
    "total_tokens": 32
  }
}
```

finish\_reason:

当达到 max\_tokens 限制，未完整输出生成的内容时值为“length”

输入，输出相关的 tokens 数

# 目录

- 0 构建测试环境
- 1 “Hello GPT”
- 2 Token: GPT 的基本文本处理单元

# Token

“OpenAI GPT tokens”是指OpenAI GPT 模型（包括ChatGPT）用于计算文本长度的基本单位。它们是一组字符，Token 并不总是和单词对应。

GPT-3

Codex

I think your codebase is exceedingly overpythonized

Clear

Show example

Tokens

Characters

10

51

I think your codebase is exceedingly overpythonized

TEXT

TOKEN IDS



GPT-4

With broad general knowledge and domain expertise, GPT-4 can follow complex instructions in natural language and solve difficult problems with accuracy.

[Learn about GPT-4](#)

Model	Input	Output
8K context	\$0.03 / 1K tokens	\$0.06 / 1K tokens
32K context	\$0.06 / 1K tokens	\$0.12 / 1K tokens

GPT-3.5 Turbo

GPT-3.5 Turbo is optimized for dialogue.

[Learn about GPT-3.5 Turbo ↗](#)

Model	Input	Output
4K context	\$0.0015 / 1K tokens	\$0.002 / 1K tokens
16K context	\$0.003 / 1K tokens	\$0.004 / 1K tokens

# Quota 和 Rate Limit

## 配额和限制参考

以下部分提供适用于 Azure OpenAI 默认配额和限制的快速指南：

限制名称	限制值
每个 Azure 订阅中每个区域的 OpenAI 资源	30
每个模型和地区的默认配额（以每分钟令牌为单位） <sup>1</sup>	Text-Davinci-003： 120 K GPT-4： 20 K GPT-4-32K： 60 K 其他： 240 K

# 课后作业

1. 写一句中文，然后通过 API 让 GPT 输出英语译文
2. 分别计算问题 1 中英语句子及对应中文译文的 token 数



# 课后参考

1. OpenAI API Chat Completion: <https://platform.openai.com/docs/api-reference/completions/create>
2. Azure OpenAI Chat Completion: <https://learn.microsoft.com/en-us/azure/ai-services/openai/reference#chat-completions>

# THANKS

## 2 | 提示词技巧

获得代码友好的回复



# 回顾

```
def translate(text):
    messages = []
    messages.append( {"role": "system",
                      "content": "You are a translate. Please, translate the user's request to English."})
    messages.append( {"role": "user", "content": text})
    response = openai.ChatCompletion.create(
        engine=deployment,
        model = "gpt-4", # for OpenAI
        messages=messages,
        temperature=0.5,
        max_tokens = 100
    )
    return response["choices"][0]["message"]["content"]
```

# 英语提示词通常更省 Token

在未来还没有到来的时候，总要有人把它创造出来，那个人应该是我们。	35 Tokens
Before the future arrives, there always needs to be someone to create it, and that person should be us.	22 Tokens

# 目录

1 典型应用场景

2 提示词技巧



# 目录

1 典型应用场景

2 提示词技巧

# 意图识别

```
import openai
response = openai.ChatCompletion.create(
    engine=deployment, # engine = "deployment_name".
    model=model,
    temperature = 0,
    messages=[
        {"role": "system", "content": ""
        Recognize the intent from the user's input
        ""},
        #{"role": "user", "content": "订明天早5点北京到上海的飞机"}
        {"role": "user", "content": "提醒我明早8点有会议"}
    ]
)
print(response.choices[0].message.content)
```

# 生成 SQL



```
system_prompt = """
```

```
You are a software engineer, you can write a SQL string as the answer according to the user request.
```

```
    The user's requirement is based on the given tables:
```

```
        table "students" with the columns [studentID, name, course_ID, score];
```

```
        table "courses" with the columns [courseID, name]."""
```

```
#prompt = system_prompt
```

```
response = openai.ChatCompletion.create(
```

```
    engine=deployment,
```

```
    temperature = 0.9,
```

```
    messages=[
```

```
        {"role": "system", "content": prompt},
```

```
        {"role": "user", "content": "列出英语成绩大于80分的学生"},
```

```
    ],
```

```
    max_tokens = 200
```

```
)
```

```
print(response.choices[0].message.content)
```



# 目录

1 典型应用场景

2 提示词技巧

# 提示词的一些小技巧

- 利用反向提示词（如：不使用.....）
- 规范输出的格式（便于程序处理）
- 文本规范异常输出的格式（便于程序处理）
- 不断迭代
- 提示词尽量使用英语

# 规范输出的格式

“列出英语成绩大于80分的学生, **仅返回SQL语句**”



# 规范输出的格式

When you cannot create the SQL query for the user's request based on the given tables, please, only return "**invalid request**"

# 课后作业

1. 用 GPT 编写一个用户评语判断程序，输入用户评论，输出评论是正向的还是反向的，分别用 Y 和 N 来表示

THANKS



# 3 | 初识 LangChain

你的瑞士军刀

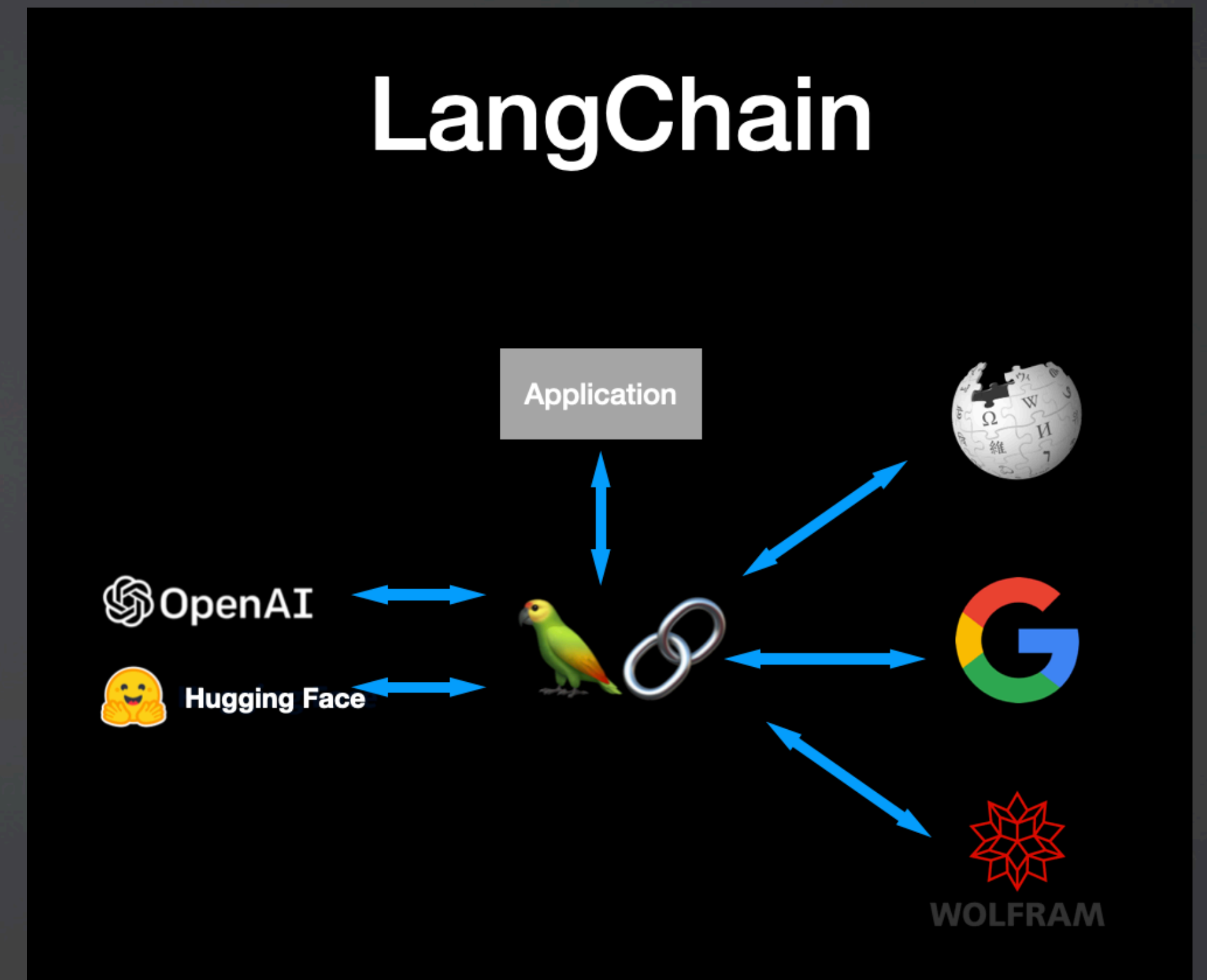
# 回顾

```
import openai
def analyze_user_review(text):
    messages = []
    messages.append( {"role": "system",
                      "content": """
                        You are an assistant.
                        Please, analyze the user reviews according to the following instruction:
                        If the review is positive, you should output 'Y', otherwise output 'N'
                        """})
    messages.append( {"role": "user", "content": text})
    response = openai.ChatCompletion.create(
        engine=deployment,
        messages=messages,
        temperature=0.5,
        max_tokens = 100
    )
    return response["choices"][0]["message"]["content"]
```

# LangChain

LangChain's flexible abstractions and extensive toolkit enables developers to harness the power of LLMs.

连接大语言模型和各种应用





# 目录

1 提示词模版

2 与 API 交互

3 链式请求

# 目录

1 提示词模版

2 与 API 交互

3 链式请求

# 提示词模版

```
from langchain import PromptTemplate, OpenAI, LLMChain

from langchain.chat_models import AzureChatOpenAI
# from langchain.chat_models import ChatOpenAI #For Direct GPT client

prompt_template = "What is a good name for a company that makes {product}?"

#llm = ChatOpenAI(temperature=0, model_name="gpt-4") #For Direct GPT client
llm = AzureChatOpenAI(deployment_name = deployment, model_name=model, temperature=0, max_tokens=200)
llm_chain = LLMChain(
    llm=llm,
    prompt=PromptTemplate.from_template(prompt_template)
)
```



# 目录

1 提示词模版

2 与 API 交互

3 链式请求

# 与 API 交互

通过 API 获取信息，然后通过 LLM 给出最终答复。

例如：

HTTP request chain

使用请求库从 URL 获取 HTML 结果，然后使用 LLM 分析结果。

# 目录

1 提示词模版

2 与 API 交互

3 链式请求

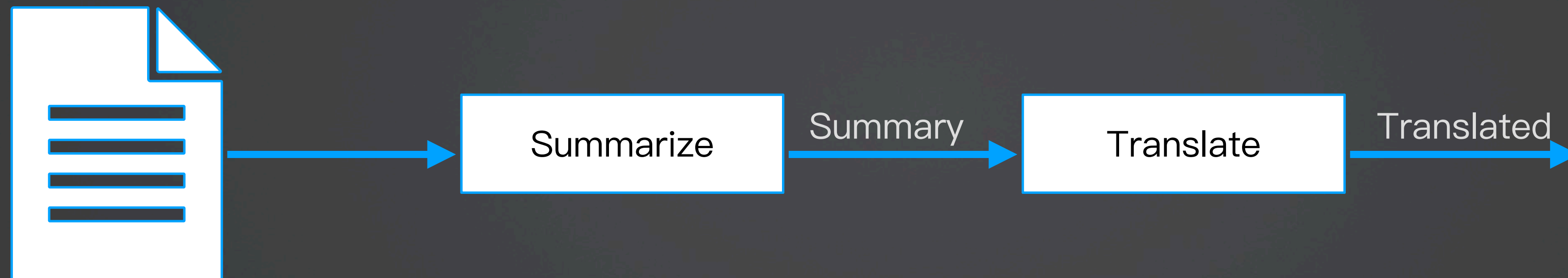


# 链式请求

## 将复杂任务分解为更简单的子任务

正如在软件工程中，将复杂系统分解为一组模块化组件是一种良好实践一样，对于提交给 GPT 的任务也是如此。复杂任务的错误率往往高于简单任务。此外，复杂任务通常可以重新定义为简单任务的工作流程，其中较早任务的输出用于构建较晚任务的输入。

# 链式请求



# 总结

- 1 提示词模版
- 2 与 API 交互 (LangChain 对提示词模版的应用举例)
- 3 链式请求 (重点)



# 课后作业

1. 先将用户问题利用搜索引擎进行检索，然后再翻译成英文

# 课后参考

1. LangChain Chain 参考: <https://python.langchain.com/docs/modules/chains/>
2. Azure OpenAI Chat Completion 参考:  
<https://learn.microsoft.com/en-us/azure/ai-services/openai/reference#chat-completions>

THANKS



# 4 | 保持会话状态

让 Chatbot 获得记忆

# 回顾

```
def overall(question):
    inputs = {
        "query": question,
        "url": "http://www.baidu.com/s?wd=" + question.replace(" ", "+")
    }

    overall_chain = SequentialChain(
        chains=[query_chain, translating_chain],
        input_variables=["query", "url"],
        output_variables=["translated"],
        verbose=True
    )

    return overall_chain(inputs)["translated"]
```

# 目录

1 没有记忆的模型

2 利用 Gradio 快速构建原型

3 构建有状态的对话



# 目录

1 没有记忆的模型

2 利用 Gradio 快速构建原型

3 构建有状态的对话

# 没有记忆模型

模型能够独立处理每个输入查询，而不受之前交互的影响。  
无状态有助于提高模型的可扩展性和性能。

# 目录

1 没有记忆的模型

2 利用 Gradio 快速构建原型

3 构建有状态的对话



# 利用 Gradio 快速构建原型

```
import gradio as gr

def respond(message, chat_history):
    bot_message = get_response(message)
    chat_history.append((message, bot_message))
    return "", chat_history

with gr.Blocks() as demo:
    chatbot = gr.Chatbot(height=240) #对话框
    msg = gr.Textbox(label="Prompt") #输入框
    btn = gr.Button("Submit") #提交按钮
    #提交
    btn.click(respond, inputs=[msg, chatbot], outputs=[msg, chatbot])
    msg.submit(respond, inputs=[msg, chatbot], outputs=[msg, chatbot])
gr.close_all()
demo.launch()
```

A screenshot of a Gradio chatbot interface. At the top, there's a tab labeled "Chatbot". Below it, a yellow message bubble contains the text "我是蔡超，你叫什么名字?". Below that, a light blue message bubble contains the text "您好，蔡超。我是OpenAI的人工智能助手，我没有具体的名字。有什么可以帮助您的吗?". Under the chat bubbles, there's a section labeled "Prompt" containing a text input field with a vertical cursor. At the bottom, there's a wide, light blue button labeled "提交".

# 目录

1 没有记忆的模型

2 利用 Gradio 快速构建原型

3 构建有状态的对话

# 构建有状态的对话

```
def history_to_prompt(chat_history):
    msg = [{"role": "system", "content": "You are an AI assistant."}]
    i = 0
    for round_trip in chat_history:
        msg.append({"role": "user", "content": round_trip[0]})
        msg.append({"role": "assistant", "content": round_trip[1]})
    return msg

def get_response(msg):
    # print(msg)
    response = openai.ChatCompletion.create(
        engine=deployment, # engine = "deployment_name".
        messages=msg,
        temperature = 0.9,
        max_tokens = 600
    )
    return response.choices[0].message.content
```





# 构建有状态的对话

```
llm = AzureChatOpenAI(deployment_name="gpt4", temperature=0.3, max_tokens=1000,  
                      streaming=True)
```

```
memory = ConversationBufferWindowMemory(k=10) #k 表示保存的会话次数
```

```
def get_response(input):  
    conversation_with_memory = ConversationChain(  
        llm=llm,  
        memory=memory,  
        verbose=False  
    )  
    return conversation_with_memory.predict(input=input)
```

# 总结

- 1 GPT 是无状态的服务（没有记忆的模型）
- 2 利用 Gradio 快速构建原型
- 3 构建有状态的对话（手动组装或者使用 LangChain）

# 课后作业

1. 在聊天机器人中尝试使用 LangChain 中不同的 memory



# 课后参考

1. LangChain Chain memory 参考: <https://python.langchain.com/docs/modules/memory/types/>

THANKS