

占位符

# 大模型提示工程快速入门

九天Hector

# 什么是提示工程？

- 提示工程（Prompt engineering）：用于制定和优化用于与大型语言模型（如 GPT-3 或 GPT-4）交互的“提示”或“查询”；
- 例如：Ch 1.2中简单借助GPT API向GPT模型进行提问，其中的messages其实就是提示，而提示工程则是指编写提示的方法，其目的在于能够更加高效精准的获取答案；

```
response = openai.ChatCompletion.create(  
    model="gpt-3.5-turbo",  
    messages=[  
        {"role": "user", "content": "请问什么是机器学习? "}  
    ]  
)
```

```
response.choices[0].message['content']
```

'机器学习是一种人工智能的分支，通过使用数学和统计方法来自动分析和理解数据，并使计算机能够自主学习和改进性能。它是通过从数据中学习模式和规律，而不是通过明确编程来实现任务的一种方法。机器学习可以帮助计算机系统通过从大量数据中提取规律来预测结果、进行分类、聚类、识别图像和语音、自动驾驶等。'

占位符

**重要性被严重低估的技术：提示工程**



# 大模型工程师的基本功

## 提示工程

占位符

- 提示是和大模型交互的唯一方法，**每时每刻都会用到提示技巧**；
- 提示效果好坏直接**决定大模型运行效果**；
- 微调（或其他技术手段）无法代替提示工程技术，相反，**提示工程是最为简单的提升大模型应用效果的方法**；
- 提示工程可以看成是使用**大模型工具熟练度**的直接证明；

### Prompt engineering

This guide shares strategies and tactics for getting better results from large language models (sometimes referred to as GPT models) like GPT-4. The methods described here can sometimes be deployed in combination for greater effect. We encourage experimentation to find the methods that work best for you.

Some of the examples demonstrated here currently work only with our most capable model, `gpt-4`. In general, if you find that a model fails at a task and a more capable model is available, it's often worth trying again with the more capable model.

You can also explore example prompts which showcase what our models are capable of:



#### Prompt examples

Explore prompt examples to learn what GPT models can do



Explore prompt examples to learn what GPT models can do

Prompt examples

占位符

提示工程实战应用

提升模型推理能力与自然语言编程



# 一组典型的模型推理能力测试

- **Q1:**罗杰有五个网球，他又买了两盒网球，每盒有3个网球，请问他现在总共有多少个网球？
- **Q2:**食堂总共有23个苹果，如果他们用掉20个苹果，然后又买了6个苹果，请问现在食堂总共有多少个苹果？
- **Q3:**杂耍者可以杂耍16个球。其中一半的球是高尔夫球，其中一半的高尔夫球是蓝色的。请问总共有多少个蓝色高尔夫球？
- **Q4:**艾米需要4分钟才能爬到滑梯顶部，她花了1分钟才滑下来，水滑梯将在15分钟后关闭，请问在关闭之前她能滑多少次？

## Q1:“请帮我编写一个多轮对话函数”

```
response = openai.ChatCompletion.create(  
    model="gpt-3.5-turbo-0613",  
    messages=[  
        {"role": "user", "content": "请帮我编写一个多轮对话函数。"}  
    ]  
)
```

```
response.choices[0].message['content']
```

'当然，我可以帮你编写多轮对话函数。请告诉我具体的需求和要实现的功能，然后我们可以一起开始编写代码。'

代码详见：  
《Ch 1.3 提示工程实操》

## 基础提示技巧一：清晰准确

**用户需求：**编写一个能够实现多轮对话的函数

**简单提问：**“请帮我编写一个多轮对话函数”

**存在问题：**缺乏背景信息，例如用哪个函数编写多轮对话？

**修改之后：**请帮我编写一个**基于OpenAI ChatCompletion.create**函数的多轮对话函数（其他必要时候可以添加背景信息）



# 占位符

```
q1 = "请帮我编写一个基于OpenAI ChatCompletion.create函数的多轮对话函数"
```

```
response = openai.ChatCompletion.create(  
    model="gpt-3.5-turbo-0613",  
    messages=[  
        {"role": "user", "content": q1}  
    ]  
)
```

```
response.choices[0].message['content']
```

```
'当然可以! 以下是一个基于OpenAI ChatCompletion.create函数的多轮对话函  
数的示例: \n\n```\npython\nimport openai\n\ndef start_chat(prompt):\n    # 设置OpenAI API\n    openai.api_key = 'YOUR_API_KEY_HERE'\n    # 初始化对话历史\n    chat_history = []\n    # 发送初始提示\n    response = openai.ChatCompletion.create(\n        model="gpt-3.5-turbo",\n        messages=[\n            {"role": "system", "content": "You are a helpful assistant."},\n            {"role": "user", "content": prompt}\n        ]\n    )\n    # 提取模型的回复\n    model_reply = response['choices'][0]['message']['content']\n    # 将模型的回复添加到对话历史中\n    chat_history.append(model_reply)\n    # 返回模型的回复作为初始对话回复\n    return model_reply\n\ndef continue_chat(prompt):\n    # 检查对话历史是否为空\n    if len(chat_history) == 0:\n        return start_chat(prompt)\n    # 向聊天历史中添加用户输入\n    chat_history.append(prompt)\n    # 发送完整的对话历史给模型\n    response = openai.ChatCompletion.create(\n        model="gpt-3.5-turbo",\n        messages=[\n            {"role": "system", "content": "You are a helpful assistant."},\n            {"role": "user", "content": prompt}\n        ]\n    )\n    # 提取模型的回复\n    model_reply = response['choices'][0]['message']['content']\n    # 将模型的回复添加到对话历史中\n    chat_history.append(model_reply)\n    # 返回模型的回复\n    return model_reply\n\nif __name__ == '__main__':\n    prompt = input("请输入您的问题: ")  
    reply = start_chat(prompt)\n    while True:\n        prompt = input("请输入您的问题: ")  
        reply = continue_chat(prompt)
```

## 提示技巧一：清晰准确

需要完整详细的描述用户需求，才能让大模型了解用户意图

占位符

# 最终完成多轮对话函数编写

当然可以！以下是一个基于OpenAI ChatCompletion.create函数的多轮对话函数的示例：

```
import openai

def start_chat(prompt):
    # 设置OpenAI API
    openai.api_key = 'YOUR_API_KEY_HERE'

    # 初始化对话历史
    chat_history = []

    # 发送初始提示
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": prompt}
        ]
    )

    # 提取模型的回复
    model_reply = response['choices'][0]['message']['content']

    # 将模型的回复添加到对话历史中
    chat_history.append(model_reply)

    # 返回模型的回复作为初始对话回复
    return model_reply

def continue_chat(prompt):
    # 检查对话历史是否为空
    if len(chat_history) == 0:
        return start_chat(prompt)

    # 向聊天历史中添加用户输入
    chat_history.append(prompt)
```

```
# 发送完整的对话历史给模型
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": chat_history[-2]},
        {"role": "assistant", "content": chat_history[-1]}
    ]
)

# 提取模型的回复
model_reply = response['choices'][0]['message']['content']

# 将模型的回复添加到对话历史中
chat_history.append(model_reply)

# 返回模型的回复
return model_reply

# 示例对话
chat_prompt = "What's the weather like today?"

# 开始第一次对话
chat_reply = start_chat(chat_prompt)
print("Assistant:", chat_reply)

# 第一次对话后，继续对话
while True:
    user_input = input("User: ")
    chat_reply = continue_chat(user_input)
    print("Assistant:", chat_reply)
```

代码详见：  
《Ch 1.3 提示工程实操》



## 基础提示技巧二：设置角色

```
messages=[  
    {"role": "system", "content": "你是一位擅长编写代码的大模型技术专  
    {"role": "user", "content": q1}  
]
```

```
response = openai.ChatCompletion.create(  
    model="gpt-3.5-turbo-0613",  
    messages=messages  
)
```

- 角色会较大程度影响GPT系列模型输出结果，当我们输入“**你是一位擅长写代码的技术专家**”时，模型便会“以专家水平”要求自己，此时往往编写的**代码会更加简洁高效**；



当然可以！以下是一个基于OpenAI ChatCompletion.create函数的多轮对话函数的示例：

```
import openai

def start_chat(prompt):
    # 设置OpenAI API
    openai.api_key = 'YOUR_API_KEY_HERE'

    # 初始化对话历史
    chat_history = []

    # 发送初始提示
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": prompt}
        ]
    )

    # 提取模型的回复
    model_reply = response['choices'][0]['message']['content']

    # 将模型的回复添加到对话历史中
    chat_history.append(model_reply)

    # 返回模型的回复作为初始对话回复
    return model_reply

def continue_chat(prompt):
    # 检查对话历史是否为空
    if len(chat_history) == 0:
        return start_chat(prompt)

    # 向聊天历史中添加用户输入
    chat_history.append(prompt)

    # 发送完整的对话历史给模型
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": chat_history[-2]},
            {"role": "assistant", "content": chat_history[-1]}
        ]
    )

    # 提取模型的回复
    model_reply = response['choices'][0]['message']['content']

    # 将模型的回复添加到对话历史中
    chat_history.append(model_reply)

    # 返回模型的回复
    return model_reply

# 示例对话
chat_prompt = "What's the weather like today?"

# 开始第一次对话
chat_reply = start_chat(chat_prompt)
print("Assistant:", chat_reply)

# 第一次对话后，继续对话
while True:
    user_input = input("User: ")
    chat_reply = continue_chat(user_input)
    print("Assistant:", chat_reply)
```

```
import openai

def chat_with_openai(prompt):
    # 设置OpenAI的API密钥
    openai.api_key = 'your-api-key'

    # 初始化对话历史
    chat_history = []

    while True:
        # 构建聊天输入，将对话历史 and 用户输入拼接起来
        user_input = input("User: ")
        chat_history.append(f"User: {user_input}")
        chat_history_str = '\n'.join(chat_history)

        # 使用OpenAI API进行聊天
        response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[
                {"role": "system", "content": "You are a helpful assistant."},
                {"role": "user", "content": chat_history_str}
            ],
            max_tokens=50
        )

        # 解析API响应，提取助手的回复
        assistant_reply = response.choices[0].message.content
        print("Assistant:", assistant_reply)

        # 将助手的回复添加到对话历史中
        chat_history.append(f"Assistant: {assistant_reply}")

        # 当用户输入"结束对话"时退出循环
        if user_input.lower() == "end conversation":
            break

    chat_with_openai("Let's start the conversation!")
```

代码详见：  
《Ch 1.3 提示工程实操》

占位符

## 不同身份编写代码效果

- 左侧代码为未设置身份时编写结果
- 右侧代码为设置身份时编写结果
- 专家身份编写代码约减少50%左右

# 多轮函数运行结果

```
chat_with_openai("Let's start the conversation!")
```

User: 请问什么是机器学习

Assistant: 机器学习是一种人工智能领域的技术，它使计算机能够从数据中学习和改进性能，而不需要进行明确的程序编写。通过利用统计学和算法模型，机器学习可以自动地从大量的数据中发现规律、模式和趋势，并且基于这些发现做出预测和决策。机器学习常被应用于各种领域，如图像识别、语音识别、自然语言处理、推荐系统等，以帮助人们解决复杂的问题和改进现有的系统。

User: 我的上一个问题是？

Assistant: 您之前的问题是关于什么是机器学习的。

User: end conversation

Assistant: Assistant: If you have any more questions in the future, feel free to ask. Have a great day!

这种引导大模型进行编程方法，也被称为“**自然语言编程**”，是目前非常重要的大模型应用方向；



# 完整自然语言编程 提示方法

占位符

system message保持不变

user message如下：

我现在已经获取了OpenAI API-KEY，并已赋值给openai.api\_key。我现在想要编写一个基于ChatCompletion.create函数的多轮对话函数，函数要求如下：

- 1.函数无任何必选参数，用户无需输入任何参数即可并开启多轮对话；
- 2.当用户输入退出时退出多轮对话；
- 3.ChatCompletion.create函数运行过程不设置任何其他参数；
- 4.请在函数编写过程中，帮我编写详细的函数说明文档，用于说明函数功能、函数参数情况以及函数返回结果等信息；



# 效果展示

占位符

```
def chat_with_ai():  
    """  
    基于OpenAI GPT-3 API的多轮对话函数  
  
    1. 该函数无任何必选参数，用户无需输入任何参数即可开始多轮对话。  
    2. 当用户输入"退出"时，函数将退出多轮对话。  
    3. 函数使用ChatCompletion.create函数运行AI模型，内部不设置任何其他参数。  
  
    Returns:  
        response: AI的回复  
  
    Raises:  
        Exception: 当与API通信发生错误时，将引发异常。  
  
    """  
  
    # 初始化对话过程  
    user_input = ""  
    conversation_history = []  
  
    while user_input.lower() != "退出":  
        # 获取用户输入  
        user_input = input("> ")  
  
        # 创建聊天记录，在历史记录中保存用户输入和AI回复  
        conversation_history.append({"role": "user", "content": user_input})  
  
        # 发送请求到 GPT-3 API  
        try:  
            response = openai.ChatCompletion.create(  
                model="gpt-3.5-turbo",  
                messages=conversation_history  
            )  
        except Exception as e:  
            raise Exception("与 GPT-3 API 进行通信时发生错误:" + str(e))
```

chat\_with\_ai()

> 你好

AI: 你好! 有什么我可以帮助你的吗?

> 请问什么是机器学习?

AI: 机器学习是一种人工智能领域的技术，通过使用算法和模型让计算机系统从经验数据中学习，并根据学习的知识进行预测和决策。它的目标是让计算机具备从数据中获取知识、理解模式和进行自主学习的能力，而无需明确地编程指令。机器学习的过程涉及数据的收集、预处理、特征提取、模型构建、模型训练和评估等步骤。通过机器学习，计算机可以自动发现数据中的规律和模式，从而对未知数据做出预测和判断。机器学习在各个领域都有广泛的应用，如推荐系统、图像识别、自然语言处理等。

> 请问我的上一个问题是?

AI: 你上一个问题是“请问什么是机器学习?”

> 退出

AI: 好的，如果你有任何其他问题，随时可以来问我。再见!

'好的，如果你有任何其他问题，随时可以来问我。再见!'