GPT models

> 原文： https://platform.openai.com/docs/guides/gpt

OpenAI的GPT（生成式预训练变换器）模型已经经过训练，可以理解自然语言和代码。GPT可以根据输入生成文本输出。GPT的输入也被称为“提示”。设计一个提示本质上就是如何“编程”一个GPT模型，通常是通过提供指令或一些成功完成任务的示例来实现。

使用GPT，您可以构建以下应用程序：

起草文件
编写计算机代码
回答关于知识库的问题
分析文本
创建对话代理
为软件提供自然语言界面
在各种学科中提供辅导
翻译语言
为游戏模拟角色
...等等！

要通过OpenAI API使用GPT模型，您需要发送一个包含输入和API密钥的请求，并接收一个包含模型输出的响应。我们最新的模型，gpt-4和gpt-3.5-turbo，可以通过聊天完成API端点进行访问。目前，只有较旧的遗留模型可以通过完成API端点进行访问。

	MODEL FAMILIES	API ENDPOINT
Newer models (2023–)	gpt-4, gpt-3.5-turbo	https://api.openai.com/v1/chat/completions
Updated base models (2023)	babbage-002, davinci-002	https://api.openai.com/v1/completions
Legacy models (2020–2022)	text-davinci-003, text-davinci-002, davinci, curie, babbage, ada	https://api.openai.com/v1/completions

您可以在Playground中尝试使用GPT模型。如果您不确定要使用哪个模型，可以使用gpt-3.5-turbo或gpt-4。

Chat completions API

聊天模型以消息列表作为输入，并返回模型生成的消息作为输出。尽管聊天格式旨在使多轮对话变得简单，但对于没有对话的单轮任务同样有用。

一个示例的API调用如下所示：

```
import openai

openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
        {"role": "user", "content": "Where was it played?"}
    ]
)
```

See the full API reference documentation here.

主要的输入是消息参数。消息必须是一个消息对象的数组，每个对象都有一个角色（可以是"system"、"user"或"assistant"）和内容。对话可以短到只有一条消息，也可以有很多轮回合。

通常，一个对话的格式是先有一个系统消息，然后是交替的用户和助手消息。

系统消息有助于设定助手的行为。例如，您可以修改助手的个性，或者提供关于它在整个对话中应该如何行为的具体指示。但请注意，系统消息是可选的，如果没有系统消息，模型的行为可能类似于使用一个通用的消息，如"你是一个乐于助人的助手"。

用户消息为助手提供了请求或评论以供回应。助手消息存储了先前的助手回应，但也可以由您编写，以给出期望行为的示例。

当用户指令引用先前消息时，包含对话历史记录非常重要。在上述示例中，用户的最后一个问题"它在哪里进行的？"只有在关于2020年世界系列赛的先前消息的上下文中才有意义。因为模型对过去的请求没有记忆，所有相关的信息必须作为每个请求中的对话历史的一部分提供。如果对话无法适应模型的令牌限制，那么它将需要以某种方式进行缩短。

> 要模仿在ChatGPT中看到的文本逐步返回的效果，将stream参数设置为true。

Chat completions response format

An example Chat completions API response looks as follows:

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
```

```
        "content": "The 2020 World Series was played in Texas at Globe Life Field in
Arlington.",
        "role": "assistant"
      }
    }
  ],
  "created": 1677664795,
  "id": "chatcmpl-7QyqpwdfhqwajicIEznoc6Q47XAyW",
  "model": "gpt-3.5-turbo-0613",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 17,
    "prompt_tokens": 57,
    "total_tokens": 74
  }
}
```


在Python中，可以用response['choices'][0]['message']['content']来提取助手的回复。

每个响应都会包含一个finish_reason。finish_reason的可能值有：

stop：API返回了完整的消息，或者由于提供的stop参数中的一个停止序列终止了消息
length：由于max_tokens参数或令牌限制，模型输出不完整
function_call：模型决定调用一个函数
content_filter：由于我们的内容过滤器的标记，省略了内容
null：API响应仍在进行中或不完整
根据输入参数（如下面所示的提供函数），模型响应可能包含不同的信息。

Function calling

在API调用中，您可以向gpt-3.5-turbo-0613和gpt-4-0613描述函数，并让模型智能地选择输出一个包含调用这些函数的参数的JSON对象。聊天完成API并不调用函数；相反，模型生成的JSON可以用于在您的代码中调用函数。

最新的模型（gpt-3.5-turbo-0613和gpt-4-0613）已经进行了微调，既可以检测何时应该调用函数（取决于输入），也可以响应符合函数签名的JSON。这种能力也带来了潜在的风险。我们强烈建议在代表用户进行影响世界的行动（发送电子邮件，在线发布内容，进行购买等）之前，建立用户确认流程。

> 在底层，函数以模型已经训练过的语法注入到系统消息中。这意味着函数会计入模型的上下文限制，并作为输入令牌进行计费。如果遇到上下文限制，我们建议限制函数的数量或者您为函数参数提供的文档的长度。

函数调用允许您更可靠地从模型中获取结构化数据。例如，您可以：

通过调用外部API创建回答问题的聊天机器人（例如，像ChatGPT插件那样）
例如，定义像send_email(to: string, body: string)或get_current_weather(location: string, unit: 'celsius' | 'fahrenheit')这样的函数
将自然语言转换为API调用
例如，将"谁是我的顶级客户？"转换为get_customers(min_revenue: int, created_before: string, limit: int)并调用您的内部API
从文本中提取结构化数据
例如，定义一个名为extract_data(name: string, birthday: string)的函数，或者sql_query(query: string)
...等等！

函数调用的基本步骤序列如下：

使用用户查询和在functions参数中定义的一组函数调用模型。
模型可以选择调用一个函数；如果是这样，内容将是一个符合您自定义模式的字符串化的JSON对象（注意：模型可能会生成无效的JSON或产生参数幻觉）。
在您的代码中将字符串解析为JSON，并且如果存在提供的参数，则使用这些参数调用您的函数。
通过将函数响应作为新消息追加，再次调用模型，并让模型将结果总结回用户。
您可以通过下面的示例看到这些步骤的实际操作：

```
import openai
import json


# Example dummy function hard coded to return the same weather
# In production, this could be your backend API or an external API
def get_current_weather(location, unit="fahrenheit"):
    """Get the current weather in a given location"""
    weather_info = {
        "location": location,
        "temperature": "72",
        "unit": unit,
        "forecast": ["sunny", "windy"],
    }
    return json.dumps(weather_info)


def run_conversation():
    # Step 1: send the conversation and available functions to GPT
    messages = [{"role": "user", "content": "What's the weather like in Boston?"}]
    functions = [
        {
            "name": "get_current_weather",
            "description": "Get the current weather in a given location",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "The city and state, e.g. San Francisco, CA",
                    },
                    "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]},
```

```
                },
                "required": ["location"],
            },
        }
    ]
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo-0613",
        messages=messages,
        functions=functions,
        function_call="auto",
    )

    response_message = response["choices"][0]["message"]

    # Step 2: check if GPT wanted to call a function
    if response_message.get("function_call"):
        # Step 3: call the function
        # Note: the JSON response may not always be valid; be sure to handle errors
        available_functions = {
            "get_current_weather": get_current_weather,
        }  # only one function in this example, but you can have multiple
        function_name = response_message["function_call"]["name"]
        fuction_to_call = available_functions[function_name]
        function_args = json.loads(response_message["function_call"]["arguments"])
        function_response = fuction_to_call(
            location=function_args.get("location"),
            unit=function_args.get("unit"),
        )

        # Step 4: send the info on the function call and function response to GPT
        messages.append(response_message)  # extend conversation with assistant's reply
        messages.append(
            {
                "role": "function",
                "name": function_name,
                "content": function_response,
            }
        )  # extend conversation with function response
        second_response = openai.ChatCompletion.create(
            model="gpt-3.5-turbo-0613",
            messages=messages,
        )  # get a new response from GPT where it can see the function response
        return second_response


print(run_conversation())
```

> 在函数调用中产生的幻觉输出通常可以通过系统消息来减轻。例如，如果您发现模型正在生成没有提供给它的函数的函数调用，尝试使用一个系统消息，说："只使用你被提供的函数。"

在上面的例子中，我们将函数响应发送回模型，并让它决定下一步。它回应了一个面向用户的消息，告诉用户波士顿的温度，但是根据查询的不同，它可能选择再次调用函数。

例如，如果你问模型“找出这个周末波士顿的天气，预订周六的两人晚餐，并更新我的日历”，并为这些查询提供相应的函数，它可能选择连续调用它们，只在最后创建一个面向用户的消息。

如果你想强制模型调用一个特定的函数，你可以通过设置function_call: {"name": ""}来实现。你也可以通过设置function_call: "none"来强制模型生成一个面向用户的消息。请注意，默认行为（function_call: "auto"）是让模型自己决定是否调用函数，如果是的话，调用哪个函数。

你可以在OpenAI烹饪书中找到更多关于函数调用的例子：

> Function calling
>
> https://github.com/openai/openai-cookbook/blob/main/examples/How_to_call_functions_with_chat_models.ipynb

Completions API

Completions API端点在2023年7月接收到最后的更新，并且与新的聊天完成端点的接口不同。输入不再是一系列消息，而是一个被称为提示的自由形式的文本字符串。

一个API调用的例子如下：

```
import openai

response = openai.Completion.create(
  model="text-davinci-003",
  prompt="Write a tagline for an ice cream shop."
)
```

See the full API reference documentation to learn more.

令牌对数概率
Completions API可以提供与每个输出令牌最可能的令牌相关的有限数量的对数概率。这个功能通过使用logprobs字段来控制。在某些情况下，这可以用来评估模型对其输出的信心。

插入文本
Completions端点还支持通过提供一个后缀来插入文本，除了标准的提示，这被视为一个前缀。当编写长篇文本，过渡段落，遵循大纲，或引导模型朝向一个结尾时，这种需求自然会出现。这也适用于代码，可以用来在函数或文件的中间插入。

> 为了说明后缀上下文如何影响生成的文本，考虑这个提示：“今天我决定做一个大的改变。”有很多种方式可以想象完成这句话。但是，如果我们现在提供故事的结尾：“我得到了很多关于我的新发型的赞美！”，那么预期的完成就变得清晰了。
>
> 我在波士顿大学上大学。拿到学位后，我决定做出改变。一个大的改变！

> 我打包行李，搬到了美国的西海岸。
>
> 现在，我对太平洋简直爱不释手！
>
> 通过为模型提供额外的上下文，它可以被更好地控制。然而，这对模型来说是一个更有约束性和挑战性的任务。为了获得最好的结果，我们建议以下操作：
>
> 使用max_tokens > 256。模型更擅长插入较长的完成。如果max_tokens太小，模型可能在连接到后缀之前就被切断。注意，即使使用较大的max_tokens，你只会被收取实际产生的令牌数量。
>
> 更喜欢finish_reason == "stop"。当模型到达一个自然的停止点或用户提供的停止序列时，它会将finish_reason设置为"stop"。这表明模型已经很好地连接到后缀，是完成质量的一个好信号。这对于在使用n > 1或重新采样（参见下一点）时在几个完成之间进行选择尤其相关。
>
> 重新采样3-5次。虽然几乎所有的完成都连接到前缀，但模型可能在更难的情况下难以连接到后缀。我们发现，重新采样3或5次（或使用best_of，k=3,5），并选择以"stop"为其finish_reason的样本，在这种情况下可能是一种有效的方法。在重新采样时，你通常希望有更高的温度以增加多样性。
>
> 注意：如果所有返回的样本的finish_reason都是"length"，那么很可能max_tokens太小，模型在自然地连接提示和后缀之前就用完了令牌。在重新采样之前，考虑增加max_tokens。
>
> 尝试给出更多的线索。在某些情况下，为了更好地帮助模型的生成，你可以通过给出一些模型可以遵循的模式的例子，来决定一个自然的停止点。
>
> 如何制作美味的热巧克力：
>
> 1. 烧开水
> 2. 把热巧克力放在杯子里
> 3. 把开水加入杯子
> 4. 享受热巧克力
>
> 狗是忠诚的动物。
> 狮子是凶猛的动物。
> 海豚是好玩的动物。
> 马是雄伟的动物。

Completions response format

An example completions API response looks as follows:


```
{
  "choices": [
    {
      "finish_reason": "length",
      "index": 0,
      "logprobs": null,
      "text": "\n\n\"Let Your Sweet Tooth Run Wild at Our Creamy Ice Cream Shack"
    }
  ],
  "created": 1683130927,
```

```
  "id": "cmpl-7C9Wxi9Du4j1lQjdjhxBlO22M61LD",
  "model": "text-davinci-003",
  "object": "text_completion",
  "usage": {
    "completion_tokens": 16,
    "prompt_tokens": 10,
    "total_tokens": 26
  }
}
```

In Python, the output can be extracted with `response['choices'][0]['text']`.

The response format is similar to the response format of the Chat completions API but also includes the optional field `logprobs`.

Chat completions vs. Completions

通过使用单个用户消息构造请求，可以使聊天完成格式与完成格式相似。例如，可以使用以下完成提示从英语翻译成法语：

```
Translate the following English text to French: "{text}"
```

And an equivalent chat prompt would be:

```
[{"role": "user", "content": 'Translate the following English text to French: "{text}"'}]
```

同样，完成API可以通过相应地格式化输入来模拟用户和助手之间的聊天。

这些API之间的区别主要来自于每个API中可用的底层GPT模型。聊天完成API是我们最有能力的模型（gpt-4）和我们最具成本效益的模型（gpt-3.5-turbo）的接口。作为参考，gpt-3.5-turbo的性能与text-davinci-003相似，但每个令牌的价格只有10%！请在这里查看价格详情。

Which model should I use?

我们通常建议您使用gpt-4或gpt-3.5-turbo。您应该使用哪一个取决于您使用模型的任务的复杂性。gpt-4通常在广泛的评估中表现得更好。特别是，gpt-4更能够仔细地遵循复杂的指令。相比之下，gpt-3.5-turbo更可能只遵循复杂多部分指令的一部分。gpt-4比gpt-3.5-turbo更不可能编造信息，这种行为被称为"幻觉"。gpt-4还有一个更大的上下文窗口，最大大小为8192个令牌，而gpt-3.5-turbo为4096个令牌。然而，gpt-3.5-turbo返回的输出延迟更低，每个令牌的成本也更低。

我们建议在游乐场中进行实验，以研究哪些模型为您的使用提供了最佳的价格性能折衷。一个常见的设计模式是使用几种不同的查询类型，每种查询类型都被分派给适合处理它们的模型。

GPT best practices

了解使用GPT的最佳实践可以显著提高应用程序的性能。GPT表现出的失败模式以及解决或纠正这些失败模式的方法并不总是直观的。与GPT一起工作有一种技巧，被称为“提示工程”，但随着该领域的发展，其范围已经超出了仅仅是工程提示，发展到使用模型查询作为组件的系统工程。要了解更多信息，请阅读我们关于GPT最佳实践的指南，其中包括改进模型推理，减少模型幻觉的可能性等方法。您还可以在OpenAI Cookbook中找到许多有用的资源，包括代码样本。

> GPT best practices
>
> OpenAI Cookbook.

Managing tokens

语言模型以被称为令牌的块来读写文本。在英语中，一个令牌可以短到一个字符，长到一个单词（例如，a或apple），在某些语言中，令牌甚至可以比一个字符还短，或者比一个单词还长。

例如，字符串"ChatGPT is great!"被编码成六个令牌：["Chat", "G", "PT", " is", " great", "!"]。

API调用中的令牌总数会影响：

您的API调用的成本，因为您是按令牌付费的
您的API调用需要多长时间，因为写入更多的令牌需要更多的时间
您的API调用是否能够工作，因为总令牌数必须低于模型的最大限制（对于gpt-3.5-turbo，是4096个令牌）
输入和输出的令牌都计入这些数量。例如，如果您的API调用在消息输入中使用了10个令牌，并在消息输出中收到了20个令牌，那么您将被收取30个令牌的费用。但请注意，对于某些模型，输入和输出中的令牌的价格是不同的（有关更多信息，请参阅价格页面）。

要查看API调用使用了多少令牌，可以检查API响应中的使用字段（例如，response['usage']['total_tokens']）。

像gpt-3.5-turbo和gpt-4这样的聊天模型与完成API中可用的模型一样使用令牌，但由于它们基于消息的格式，计算对话将使用多少令牌更为困难。

Counting tokens for chat API calls

以下是一个用于计算传递给gpt-3.5-turbo-0613的消息令牌数量的示例函数。
消息被转换成令牌的确切方式可能会因模型而异。因此，当未来的模型版本发布时，这个函数返回的答案可能只是近似的。ChatML文档解释了OpenAI API如何将消息转换成令牌，可能对编写您自己的函数有所帮助。

```
def num_tokens_from_messages(messages, model="gpt-3.5-turbo-0613"):
  """Returns the number of tokens used by a list of messages."""
  try:
      encoding = tiktoken.encoding_for_model(model)
  except KeyError:
      encoding = tiktoken.get_encoding("cl100k_base")
```

```
  if model == "gpt-3.5-turbo-0613":  # note: future models may deviate from this
      num_tokens = 0
      for message in messages:
          num_tokens += 4  # every message follows <im_start>{role/name}\n{content}<im_end>\n
          for key, value in message.items():
              num_tokens += len(encoding.encode(value))
              if key == "name":  # if there's a name, the role is omitted
                  num_tokens += -1  # role is always required and always 1 token
      num_tokens += 2  # every reply is primed with <im_start>assistant
      return num_tokens
  else:
      raise NotImplementedError(f"""num_tokens_from_messages() is not presently implemented
for model {model}.
  See https://github.com/openai/openai-python/blob/main/chatml.md for information on how
messages are converted to tokens.""")
```

接下来，创建一条消息并将其传递给上面定义的函数，以查看令牌计数，这应该与API使用参数返回的值匹配：

```
messages = [
  {"role": "system", "content": "You are a helpful, pattern-following assistant that
translates corporate jargon into plain English."},
  {"role": "system", "name":"example_user", "content": "New synergies will help drive top-
line growth."},
  {"role": "system", "name": "example_assistant", "content": "Things working well together
will increase revenue."},
  {"role": "system", "name":"example_user", "content": "Let's circle back when we have more
bandwidth to touch base on opportunities for increased leverage."},
  {"role": "system", "name": "example_assistant", "content": "Let's talk later when we're
less busy about how to do better."},
  {"role": "user", "content": "This late pivot means we don't have time to boil the ocean for
the client deliverable."},
]

model = "gpt-3.5-turbo-0613"

print(f"{num_tokens_from_messages(messages, model)} prompt tokens counted.")
# Should show ~126 total_tokens
```

为了确认我们上面的函数生成的数字与API返回的数字相同，创建一个新的聊天完成：

```
# example token count from the OpenAI API
import openai


response = openai.ChatCompletion.create(
    model=model,
    messages=messages,
    temperature=0,
)

print(f'{response["usage"]["prompt_tokens"]} prompt tokens used.')
```

如果要在不进行API调用的情况下查看一个文本字符串中有多少个令牌，可以使用OpenAI的tiktoken Python库。示例代码可以在OpenAI Cookbook的如何使用tiktoken计数令牌的指南中找到。

传递给API的每条消息都会消耗内容、角色和其他字段中的令牌数量，再加上一些用于幕后格式化的额外令牌。这可能在未来会有所改变。

如果一个对话的令牌数量超过了模型的最大限制（例如，对于gpt-3.5-turbo，超过4096个令牌），你将不得不截断、省略或以其他方式缩小你的文本，直到它适合为止。请注意，如果一条消息从消息输入中被移除，模型将失去对它的所有知识。

请注意，非常长的对话更有可能收到不完整的回复。例如，一个4090个令牌长的gpt-3.5-turbo对话，在只有6个令牌后就会被切断。

Parameter details

频率和存在惩罚

在聊天完成API和遗留完成API中找到的频率和存在惩罚可以用来降低采样重复令牌序列的可能性。它们通过直接修改logits（未标准化的对数概率）的加法贡献来工作。

```
mu[j] -> mu[j] - c[j] * alpha_frequency - float(c[j] > 0) * alpha_presence
```

其中：

mu[j]是第j个令牌的logits
c[j]是在当前位置之前已经采样的令牌的频率
float(c[j] > 0)如果c[j] > 0则为1，否则为0
alpha_frequency是频率惩罚系数
alpha_presence是存在惩罚系数
我们可以看到，存在惩罚是一个一次性的加法贡献，适用于所有至少被采样一次的令牌，频率惩罚是一个与特定令牌已经被采样的频率成比例的贡献。

如果目标只是稍微减少重复的样本，那么惩罚系数的合理值大约在0.1到1之间。如果目标是强烈抑制重复，那么可以将系数增加到2，但这可能会明显降低样本的质量。负值可以用来增加重复的可能性。

FAQ

为什么模型输出不一致?
API默认是非确定性的。这意味着即使你的提示保持不变，每次调用它时你可能会得到稍微不同的完成。将温度设置为0将使输出大部分确定，但仍会保留少量的变异性。

我应该如何设置温度参数?
温度的较低值会导致输出更一致，而较高的值会生成更多样化和富有创造性的结果。根据你的特定应用对一致性和创造性之间的期望权衡来选择一个温度值。

最新的模型是否可以进行微调?
是的，对于一些模型可以。目前，你只能微调gpt-3.5-turbo和我们更新的基础模型（babbage-002和davinci-002）。查看微调指南以获取更多关于如何使用微调模型的详细

你们会存储传入API的数据吗?
自2023年3月1日起，我们会保留你的API数据30天，但不再使用通过API发送的数据来改进我们的模型。在我们的数据使用政策中了解更多。一些端点提供零保留。

我如何使我的应用更安全?
如果你想在Chat API的输出中添加一个审查层，你可以按照我们的审查指南来防止显示违反OpenAI使用政策的内容。

我应该使用ChatGPT还是API?
ChatGPT为OpenAI API中的模型提供了一个聊天界面，并且有一系列内置功能，如集成浏览、代码执行、插件等。相比之下，使用OpenAI的API提供了更多的灵活性。