代码生成小程序

今天给大家介绍的是我大模型兴趣组任务汇报的成果--代码生成小程序。希望和大家分享我在前段时间内的一些学习收获。首先需要说明的是，由于本人也是新手，此程序仍然是基于OpenAI开放的接口进行开发，实现功能也可以由chatGPT完成，但此工具针对代码生成部分进行了提示词优化，具体功能在下方介绍。本工具的使用需要用户自行准备OpenAI密钥！！！

介绍的主要内容有以下几个方面：

- 小程序的获取方式；
- 小程序的功能介绍和使用教程；
- 程序python源码运行环境介绍；
- 程序源码解读；

小程序的获取方式

本次分享中提到的小程序以及源码大家可以在github仓库中找到。exe文件可以直接点击打开，同时大家也可以对源码进行修改后自行生成可执行文件（exe）。

形成可执行文件的步骤为：

- 安装pyinstaller包：
  ```
  pip install -i https://pypi.tuna.tsinghua.edu.cn/simple
  ```
- 打开python IDE终端或者cmd命令行，进入到需要打包的代码文件所在文件夹；
- 输入命令行指令生成可执行文件：
  ```
  pyinstaller -F 要打包的文件名.py -n name
  ```

生成可执行文件还有其他个性化设置，比如设置exe的图标等，具体请参照pyinstaller

完成以上步骤后，在代码文件所在文件夹或自定义生成位置的文件夹下会生成相对应的exe文件，点击即可打开。

小程序的功能介绍以及使用教程

功能介绍

- 按照指令生成代码；

- 可以自行指定GPT对话角色；自行选择GPT模型（需要用户自己的API key支持）；
- 可以进行连续多轮对话，对话内容自动保存在本地；
- 可以自行设置GPT上下文记忆能力强弱；

使用教程

以下将分别介绍执行exe可执行文件和执行源文件两方面进行介绍。如果需要自定义设置，参照下面的*源文件执行*。

程序requirment

1. 点击进入程序后，需要提供自己的OPENAI_API_KEY;
2. 需要网络代理（VPN）

前端程序（简易）

- 连接VPN
- 点击进入DSSTL.exe应用程序
- 看到提示输入OPENAI_API_KEY
- 进行提问，可以进行多轮对话
- 对话完毕后，输入'exit'结束对话，界面会显示对话日志存储位置
- 关闭程序

源文件执行

在提供的文件中，找到api.py文件，即为程序执行源文件，可以进入源文件进行执行，并可以自行修改。

执行过程较为简单，此处不进行赘述。以下主要说明如何进行模型修改，文件地址等的修改。

模型修改

本程序默认使用'gpt-3.5-turbo'模型，如需修改模型，此处提供GPT3.5所支持的部分模型：

LATEST MODEL	DESCRIPTION	MAX REQUEST	TRAINING DATA
gpt-3.5-turbo	Most capable GPT-3.5 model and optimized for chat at 1/10th the cost of text-davinci-003. Will be updated with our latest model iteration.	4,096 tokens	Up to Sep 2021
gpt-3.5-turbo-0301	Snapshot of gpt-3.5-turbo from March 1st 2023. Unlike gpt-3.5-turbo, this model will not receive updates, and will only be supported for a three month period ending on June 1st 2023.	4,096 tokens	Up to Sep 2021
text-davinci-003	Can do any language task with better quality, longer output, and consistent instruction-following than the curie, babbage, or ada models. Also supports inserting completions within text.	4,097 tokens	Up to Jun 2021
text-davinci-002	Similar capabilities to text-davinci-003 but trained with supervised fine-tuning instead of reinforcement learning	4,097 tokens	Up to Jun 2021
code-davinci-002	Optimized for code-completion tasks	8,001 tokens	Up to Jun 2021

CSDN @神州数码云基地

但目前最新的GPT3.5模型即为'gpt-3.5-turbo'模型，如果你的账号为plus账号并且申请了GPT4模型使用权限，那么也可以使用GPT4模型。

文件存储位置

```
f = Path("你希望存储的文件位置") # api.py文件的 *line22*
```

模型角色改变

模型角色改变，是指可以人为为模型定义其在对话中的角色，模型在不同的角色下的回答会有所不同。

角色修改步骤：

```
'''
修改'role':'system' 后面的'content'内容
'''
 messages=[
                            {'role':'system','content':'you are a python rookie, can not code pyth
                            {"role": "user", "content": prompt},
                        ]     # api.py文件的 *line45*
```

程序python源码运行环境

如果需要进行上述的文件自定义操作，可能部分读者电脑中的python环境无法运行程序。因此这里简要提示一下大家python环境配置。

首先需要注意你的python环境的版本，python版本不要低于3.8，因为部分3.7版本无法安装最新版的openAI包。

接着需要安装OpenAI的安装包，进入到python环境中，直接通过命令行安装:

```
pip install openai
```

显示安装完成即可。

程序源码解读

接下来，我将和大家分享一下程序源码的细节。特别关注的是，如何简易实现GPT在多轮对话中的上下文记忆功能。

首先我们用到了OpenAI开放的API,调用函数openai.ChatCompletion.create(),键入参数为prompt，返回值为GPT模型生成的回复。

函数返回类型示例如下所示：

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "Hi there! How can I assist you today?",
        "role": "assistant"
      }
    }
  ],
  "created": 1683710982,
  "id": "chatcmpl-7EaQgUFQwFZEzqcVirm8Edae83peM",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 10,
    "prompt_tokens": 32,
    "total_tokens": 42
  }
}
```


其中需要进行展示的答复为$massage[content]$,通过字典索引可以得到具体的答复。同时我们需要注意的是，在$usage$中对prompt等的token数计数。通过对$prompt_tokens$和$total_tokens$的了解，我们可以粗略估计每次对话所可能需要的token数。

那么，我们可以看到，一次简略的对话差不多需要50个tokens。而GPT-3.5-turbo的最大token数为3.2k（现在更多了），那么我们是不是可以利用这个token数进行多轮带有上下文记忆能力的对话呢？

因为目前并没有任何关于GPT模型内部是如何训练使得ChatGPT能够在多轮对话中保持不同对话间的记忆传输（不同于同一次对话中的上下文理解和记忆），在看了一些文章和相应的代码后，我发现，利用较大的token容量进行记忆的储存是常用的一种手段，也是在GPT没有开源情况下实现记忆功能最直接的方式。

这个功能的实现则主要来自这里：

```
prompt = self.bot + self.text + self.user + question
```

在上文中，我们提到，调用openai.ChatCompletion.create()时，我们键入的是prompt，在这里，我们发现，整个的prompt不仅仅是人为输入的指示或者问题，有着之前的信息。

self.bot和self.user分别表明后续内容对应主体。而self.text表示的是前几轮对话的问题以及回答信息。question则才是本次对话所输入的问题。因此prompt表示，每次对话都会将前几轮对话的信息也作为prompt输入。这一方法可行的基础则是模型可容纳token数远远大于单对话token数。但是此方法不能持续进行，因为当轮次过多时，就会出现token超限的现象。因此我们可以自行设置每次截取的之前信息的token长度，自行设置记忆对话长度，将距离当前对话轮次过远的对话信息舍去，这样就使得模型对话可以一直进行，但记忆长度一定。

修改记忆长度的代码为：

```
self.turns += [question] + [result]
# 记忆功能所能支持的token数
if len(self.turns) <= 10:
    self.text = " ".join(self.turns)
else:
    self.text = " ".join(self.turns[-50:]) #50为当前设置的截取之前对话信息token总数
```

在完成上述工作后，在老师的补充下，才看到longchain关于LLM记忆机制的教学视频memory|langchain，收获匪浅，在这里也和大家一起分享。

首先，LLM本身是不具备对话记忆功能的，其具有的是，一次长文本输入上下文的信息记忆功能。这种记忆功能的实现是transformer的功劳。LLM实现对话记忆功能的方式和我们上述的基本一致。

那么针对对话轮次过多，导致输入token过长的问题，目前主要的处理方式有以下三种：

- window memory。这个其实很好理解：窗口记忆，即对于每轮对话，只记住此次对话前有限轮对话内容，添加到本轮对话的输入中。
- token memory。这种记忆模式和第一种记忆模式相近，但是其进行内容截断的方式不同。第一种采用的是根据对话轮次进行截断，而这种是根据token数进行截断。我们上面所使用的就是token memory方式。
- summary memory。这种主要针对长对话内容设计的。将长对话内容先进行一次summary，然后将summary后的内容存储下来，作为后续对话的记忆输入。值得注意的是，summary的精辟程度是通过设置summary的最大token数实现的。由此可见，token确实是LLM开销的关键。

除此之外，还有很多不同的记忆存储方式，这些存储不再仅限于多轮对话中，他们单独开辟数据库用于记忆存储，可以有效的摆脱token限额的约束。

- vector database：向量数据库；将记忆向量化，通过表征信息在向量空间中的数据

分布情况，在记忆调用时候，通过关键词优先获取相邻空间位置上的信息作为记忆返回。详细的介绍请参照what-are-vector-databases|Medium

- structure database:例如SQL，将记忆分成两部分，一部分当作key，另外一部分当作value，key中保存具有标识性的内容，而具体的记忆内容放在value中，通过'key-value'的键值对，在需要特定记忆的时候从数据库中获取相应的value；

写完上述内容后，我最大的感想是，记忆的本质是存储。如何进行信息的存储和调用，方式多种多样，存储需要的是空间，调用需要的是索引。不论是输入中添加之前的对话信息，还是利用数据库存储对话信息，均是为了满足以上两个功能。直接加入输入中，就是最直接的索引，模型token容量提供存储空间；使用数据库，则需要key或者关键词进行索引，而database使用的是另外的空间存储，因此可以有效解决token限额的问题。

以上就是本次分享的全部内容。我也一直在学习，自己的发言也比较片面，希望大家能够谅解，希望我们一起变强！