

功能在演示视频里已经说明了，模式里的 **English** 是指的 **cmd**，**Chinese** 是指的中文和 **ai** 交互，这点可能视频里说的不是很清楚。

然后是具体实现：

一. 核心

用 `subprocess.Popen` 打开 `cmd`，然后用线程读取管道里的数据。

具体实现如下：

`subprocess.Popen` 打开 `cmd`

```
self.process = subprocess.Popen(
    "cmd.exe",
    shell=True,
    stdin=subprocess.PIPE,
    stdout=subprocess.PIPE,
    stderr=subprocess.PIPE,
    text=True,
    bufsize=1,  # 行缓冲
    universal_newlines=True
)
```

1. `universal_newlines=True` 表示将换行符自动处理为统一的 `\n`，程序每次从外部进程读取输出时，会等待直到一行数据被完整地输出后才返回给程序。
2. 在 `subprocess.Popen()` 中，管道用于传递数据，其中：
 - **stdin (标准输入)**：用来向进程发送输入数据。
 - **stdout (标准输出)**：用来接收进程的输出数据。
 - **stderr (标准错误)**：用来接收进程的错误输出数据。

通过管道，父进程（如 Python 程序）可以与子进程（如 `cmd.exe`）进行双向通信。管道本质上是内存中的一个缓冲区，数据通过该缓冲区流动。在 `subprocess.Popen()` 的上下文中，管道允许：

- 向 `cmd.exe` 传递命令（通过 `stdin`）。
- 从 `cmd.exe` 获取命令的执行结果（通过 `stdout` 和 `stderr`）。

线程读取管道里的数据

```
threading.Thread(target=self.read_stdout, daemon=True).start()
threading.Thread(target=self.read_stderr, daemon=True).start()
```

用两个线程去读取管道，然后每 0.1 毫秒获取一下有没有最新输出。

二. API 与 AI 交互设置

1. 交互

先给 AI 一段 **prompt**，然后把 GUI 里用户输入的东西给 AI，最后把 AI 返回的东西用正则表达式筛出来就行。

2. 大模型选择

其实只能用免费的两个 **speed 模型** (`ernie-speed-128k`, `ernie-speed-8k`)，其他的模型只能用一次，然后就欠费了。

三. 不同模式和 `if-else` 设计

核心在 `shot_cmd_button` 这个函数里，本质是检测输入框里输入的啥，然后用正则表达式提取特征，最后加几个嵌套的 `if-else`，在不同的条件下改改变量，触发下函数就行。

四. GUI 设计

使用了 `tkinter` 的升级版 `customtkinter`，占了代码的绝大部分行数，不过和 OS 的授课内容关系不大，这里也不再细说。值得注意的是因为 `tkinter` 的低刷新率，导致在拉滑动条的时候可能有一种粘滞感。

五. 潜在问题

两个线程无法完全捕捉 `cmd` 的输出，比如在这个 GUI 里运行代码，代码的输出好像是无法捕获的。