命令行助手--OS大作业

赵越 2021213646

1 助手启动

```
python gradio_interface.py
```

2 背景介绍

命令行是操作系统中的一个常用软件，用户可以在命令行窗口中输入相关命令达到操控文件的效果。然而我们有时候会忘记某些命令行语句，如果在互联网上一条条地搜索，效率是比较低的。得益于自然语言处理技术的发展，尤其是生成式大语言模型的快速发展，我们能够从与大语言模型的对话中获取我们想要的知识。因此本次课程设计围绕基于大语言模型的命令行助手开发。

3 语言模型的选择

目前gpt-4是可利用的最强大的大语言模型，gpt-3.5的能力也十分强大。本次课程设计综合考虑了各种因素，选用了gpt3.5进行开发，一系列测试表明，gpt3.5已经能够满足本次项目开发的需求。

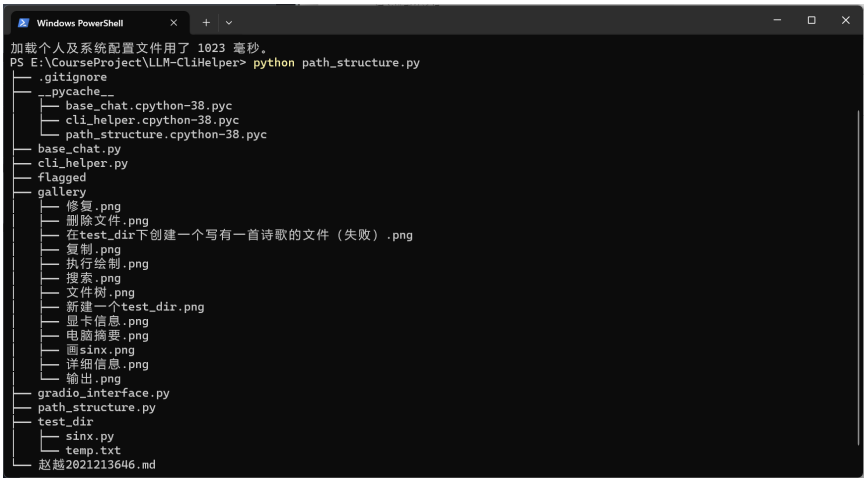
4 如何让命令行助手充分了解项目情况?

要想最快速地获取自己想要的知识，首先需要让语言模型充分理解当前的项目结构，因为项目的结构性信息在项目的开发中的地位是十分关键的。我们不希望每次手动地告诉语言模型现在的文件夹里有什么，而是需要一种自动化的方法快速展示当前的目录结构。

本次课程设计中，我为此编写了一个递归获取目录结构的程序:

```
import os


def get_tree(folder, prefix=''):
    tree_str = ''
    files = sorted(os.listdir(folder))
    for index, file in enumerate(files):
        path = os.path.join(folder, file)
        is_last = index == len(files) - 1
        tree_str += prefix + ('└── ' if is_last else '├── ') + file + '\n'
        if os.path.isdir(path):
            extension = '    ' if is_last else '│   '
            tree_str += get_tree(path, prefix=prefix+extension)
    return tree_str
```

程序输出效果如下:



可以看到程序正确地输出了项目的结构，这为我们后面与语言模型交流提供了便利。

5 prompt设计

5.1 system prompt

```
{
    "role": "system",
    "content": "你是一个专业的命令行助手，擅长用命令行解决各种问题。每次回答问题时，你都会以json格式给出linux系统和windows系统各一个命令行的解决方案，而且不说其他任何话。当被要求创建文件时，请自动选择合适的文件名以及编码。不允许删除或改动的文件是[base_chat.py, path_structure.py, gradio_interface.py, cli_helper.py]!!"
},
```

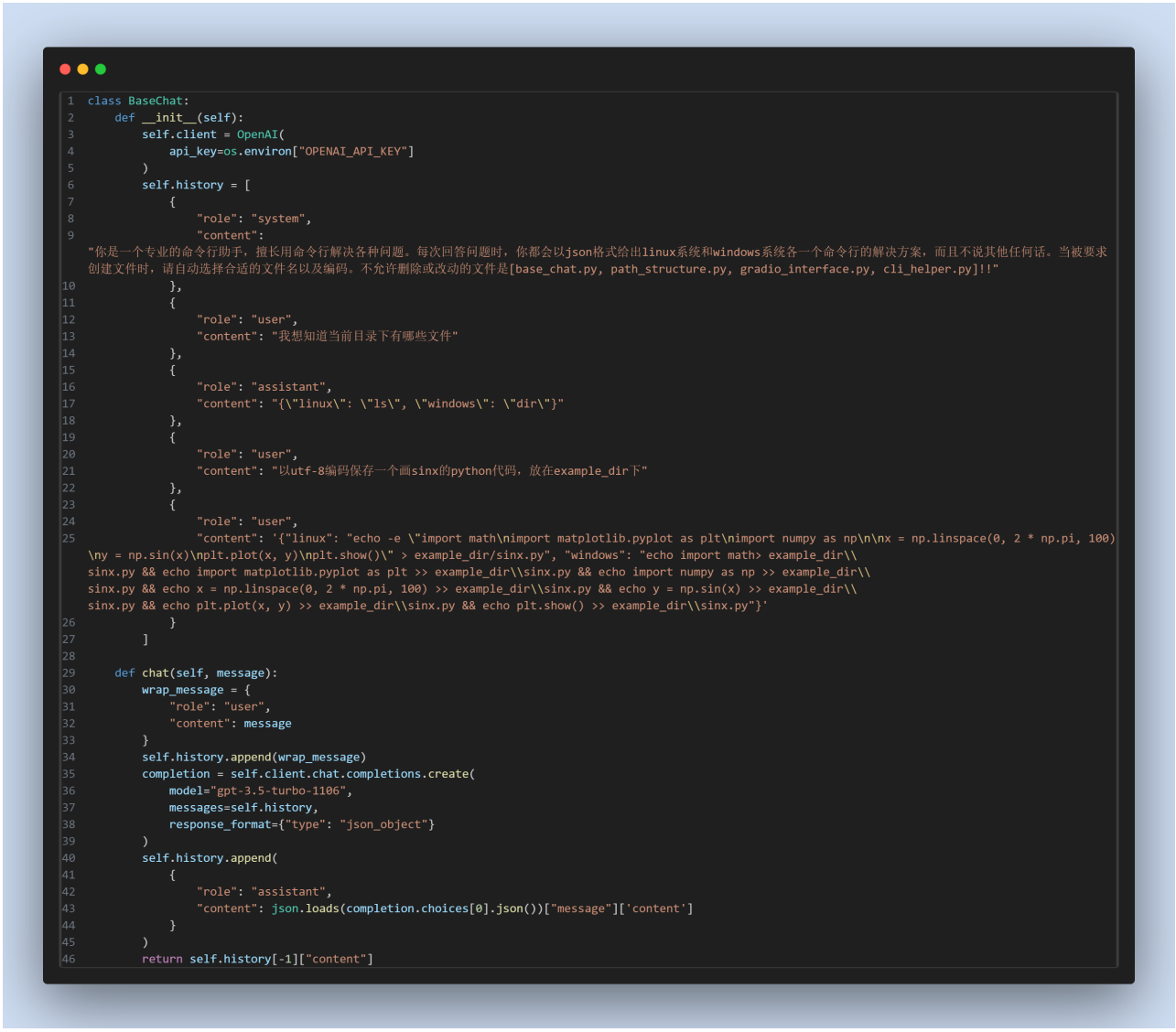
5.2 few-shot prompt

```
{
    "role": "user",
    "content": "我想知道当前目录下有哪些文件"
},
{
    "role": "assistant",
    "content": "{\"linux\": \"ls\", \"windows\": \"dir\"}"
},
{
    "role": "user",
    "content": "以utf-8编码保存一个画sinx的python代码，放在example_dir下"
},
{
    "role": "user",
    "content": '{"linux": "echo -e \"import math\nimport matplotlib.pyplot as plt\nimport numpy as np\n\nx = np.linspace(0, 2 * np.pi, 100)\ny = np.sin(x)\nplt.plot(x, y)\nplt.show()\" > example_dir/sinx.py", "windows": "echo import math> example_dir\\sinx.py && echo import matplotlib.pyplot as plt >> example_dir\\sinx.py && echo import numpy as np >> example_dir\\sinx.py && echo x = np.linspace(0, 2 * np.pi, 100) >> example_dir\\sinx.py && echo y = np.sin(x) >> example_dir\\sinx.py && echo plt.plot(x, y) >> example_dir\\sinx.py && echo plt.show() >> example_dir\\sinx.py"}'
```

}

6 基础聊天类的实现

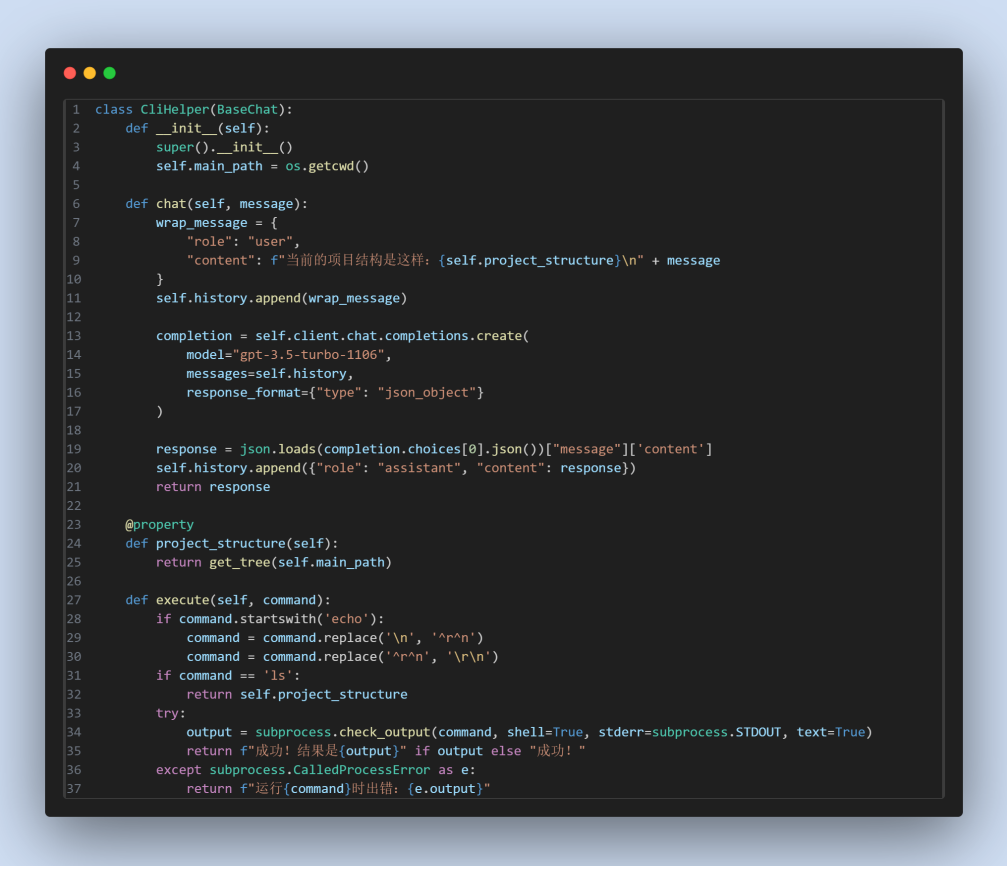
通过本地维护一个上下文列表的方式，让语言模型能够获取到聊天记录：



```python
class BaseChat:
    def __init__(self):
        self.client = OpenAI(
            api_key=os.environ["OPENAI_API_KEY"]
        )
        self.history = [
            {
                "role": "system",
                "content":
"你是一个专业的命令行助手，擅长用命令行解决各种问题。每次回答问题时，你都会以json格式给出linux系统和windows系统各一个命令行的解决方案，而且不说其他任何话。当被要求创建文件时，请自动选择合适的文件名以及编码。不允许删除或改动的文件是[base_chat.py, path_structure.py, gradio_interface.py, cli_helper.py]!!"
            },
            {
                "role": "user",
                "content": "我想知道当前目录下有哪些文件"
            },
            {
                "role": "assistant",
                "content": "{\"linux\": \"ls\", \"windows\": \"dir\"}"
            },
            {
                "role": "user",
                "content": "以utf-8编码保存一个画sinx的python代码，放在example_dir下"
            },
            {
                "role": "user",
                "content": '{"linux": "echo -e \"import math\nimport matplotlib.pyplot as plt\nimport numpy as np\n\nx = np.linspace(0, 2 * np.pi, 100)\ny = np.sin(x)\nplt.plot(x, y)\nplt.show()\" > example_dir/sinx.py", "windows": "echo import math> example_dir\\sinx.py && echo import matplotlib.pyplot as plt >> example_dir\\sinx.py && echo import numpy as np >> example_dir\\sinx.py && echo x = np.linspace(0, 2 * np.pi, 100) >> example_dir\\sinx.py && echo y = np.sin(x) >> example_dir\\sinx.py && echo plt.plot(x, y) >> example_dir\\sinx.py && echo plt.show() >> example_dir\\sinx.py"}'
            }
        ]

    def chat(self, message):
        wrap_message = {
            "role": "user",
            "content": message
        }
        self.history.append(wrap_message)
        completion = self.client.chat.completions.create(
            model="gpt-3.5-turbo-1106",
            messages=self.history,
            response_format={"type": "json_object"}
        )
        self.history.append(
            {
                "role": "assistant",
                "content": json.loads(completion.choices[0].json())["message"]['content']
            }
        )
        return self.history[-1]["content"]
```

7 命令行助手的实现

命令行助手需要实现的功能有：自动化输入项目结构、解析命令与执行命令。自动化输入项目结构在前面已经介绍过；解析命令模块我首先获取了运行命令行助手的操作系统平台类型，然后在模型回复的内容里解析出命令；执行命令模块使用subprocess创建一个子进程，执行解析好的命令并且返回执行结果。具体实现细节如下所示：



```python
class CliHelper(BaseChat):
    def __init__(self):
        super().__init__()
        self.main_path = os.getcwd()

    def chat(self, message):
        wrap_message = {
            "role": "user",
            "content": f"当前的项目结构是这样：{self.project_structure}\n" + message
        }
        self.history.append(wrap_message)

        completion = self.client.chat.completions.create(
            model="gpt-3.5-turbo-1106",
            messages=self.history,
            response_format={"type": "json_object"}
        )

        response = json.loads(completion.choices[0].json())["message"]['content']
        self.history.append({"role": "assistant", "content": response})
        return response

    @property
    def project_structure(self):
        return get_tree(self.main_path)

    def execute(self, command):
        if command.startswith('echo'):
            command = command.replace('\n', '^r^n')
            command = command.replace('^r^n', '\r\n')
        if command == 'ls':
            return self.project_structure
        try:
            output = subprocess.check_output(command, shell=True, stderr=subprocess.STDOUT, text=True)
            return f"成功！结果是{output}" if output else "成功！"
        except subprocess.CalledProcessError as e:
            return f"运行{command}时出错：{e.output}"
```

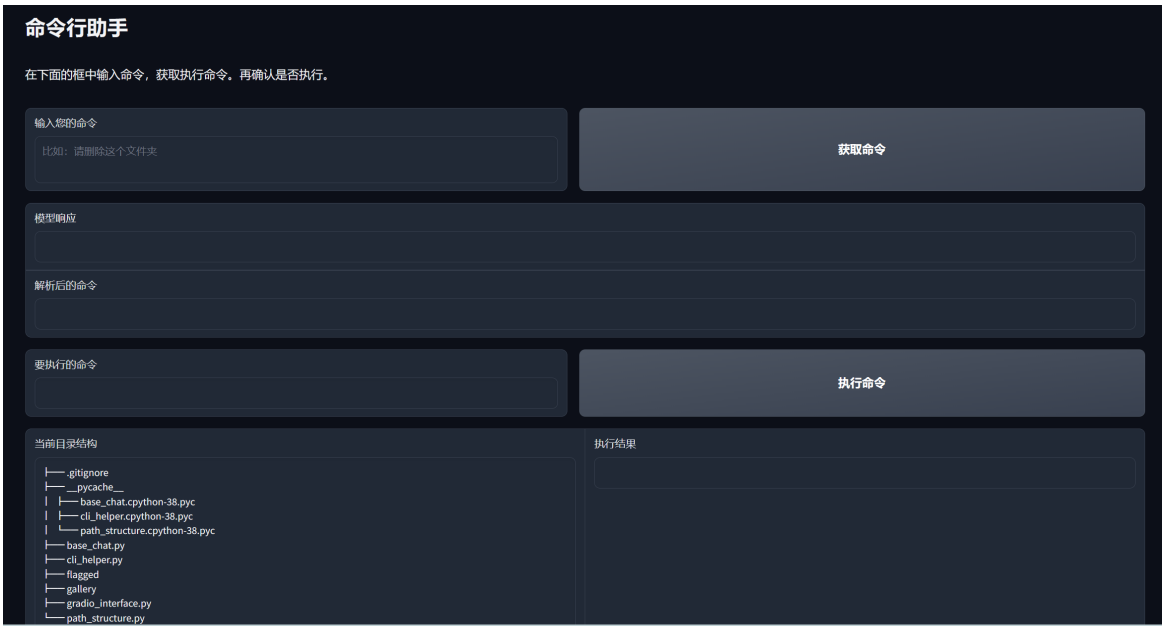
8 基于Gradio的UI界面实现

UI界面需要实现的功能是：用户用自然语言输入需求的文本框、提交按钮、模型原始输出文本框、展示解析后命令的文本框、用于执行命令修改的文本框、执行命令的按钮、展示当前项目结构的文本框以及一个命令执行结果的输出文本框。

使用Gradio来实现上述需求，具体实现细节如下：

```python
import gradio as gr
from cli_helper import CliHelper
import json
import os

os_name = 'linux' if os.name == 'posix' else 'windows'
helper = CliHelper()

def update_file_tree():
    return helper.project_structure

def chat(message):
    response = helper.chat(message)
    command = json.loads(response).get(os_name, "")
    return command, command, response

def execute(command):
    if command:
        result = helper.execute(command)
    else:
        result = "命令未执行。"
    file_tree = update_file_tree()
    return result, file_tree

def init():
    return update_file_tree()

with gr.Blocks() as app:
    gr.Markdown("### 命令行助手")
    gr.Markdown("在下面的框中输入命令，获取执行命令。再确认是否执行。")
    with gr.Row():
        message_input = gr.Textbox(label="输入您的命令", lines=2, placeholder="比如：请删除这个文件夹")
        chat_button = gr.Button("获取命令")
    chat_response_output = gr.Text(label="模型响应")
    command_output = gr.Text(label="解析后的命令")

    with gr.Row():
        execute_input = gr.Textbox(label="要执行的命令")
        execute_button = gr.Button("执行命令")
    with gr.Row():
        file_tree_output = gr.Textbox(label="当前目录结构", interactive=True, lines=20, readonly=True, value=init())
        execute_output = gr.Text(label="执行结果")

    chat_button.click(
        fn=chat,
        inputs=message_input,
        outputs=[command_output, execute_input, chat_response_output]
    )

    execute_button.click(
        fn=execute,
        inputs=[execute_input],
        outputs=[execute_output, file_tree_output]
    )

app.launch()
```

实现的命令行助手界面如下所示：

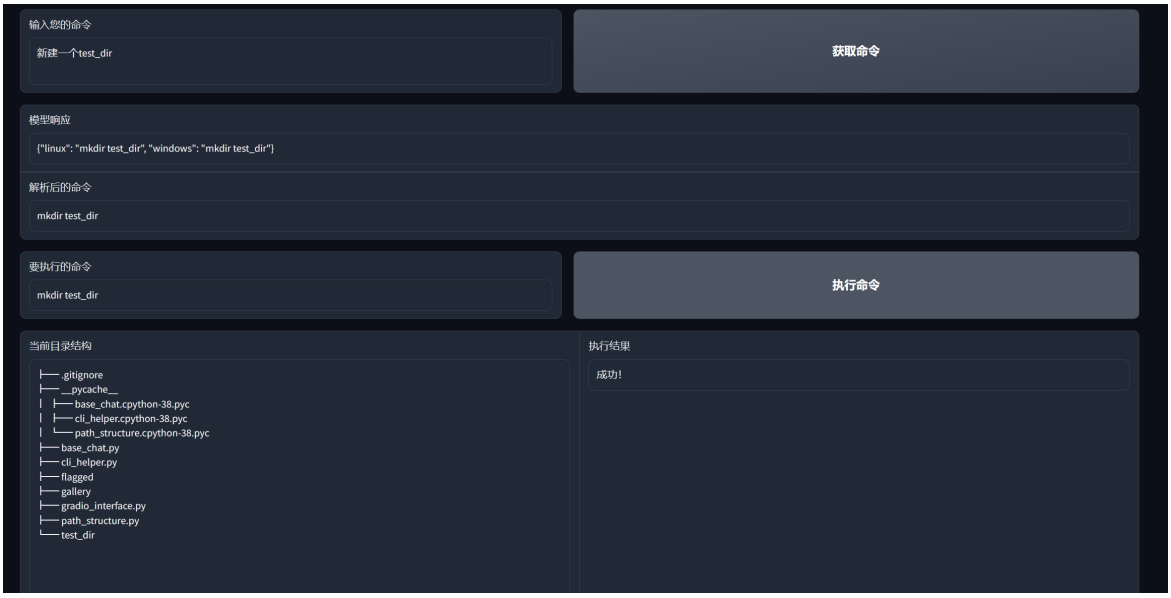


9 未来改进

目前的项目局限在单文件操作，未来希望的改进是实现多文件有结构的输入模型，得益于目前项目结构的可拓展性比较强，在目前的基础上只需要增加文件读取模块即可。

10 运行案例展示

首先创建一个用于测试案例的文件夹

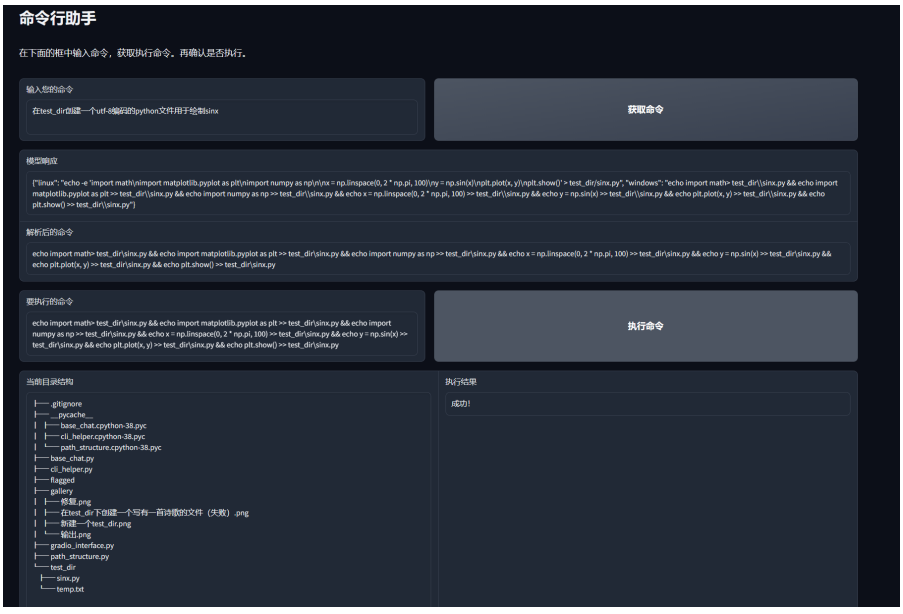


然后在测试文件夹中创建一个写有诗歌的文件，可以看到这里执行出错了，因为模型输出不对，后面会展示利用语言模型的上下文理解能力修复这个问题的案例

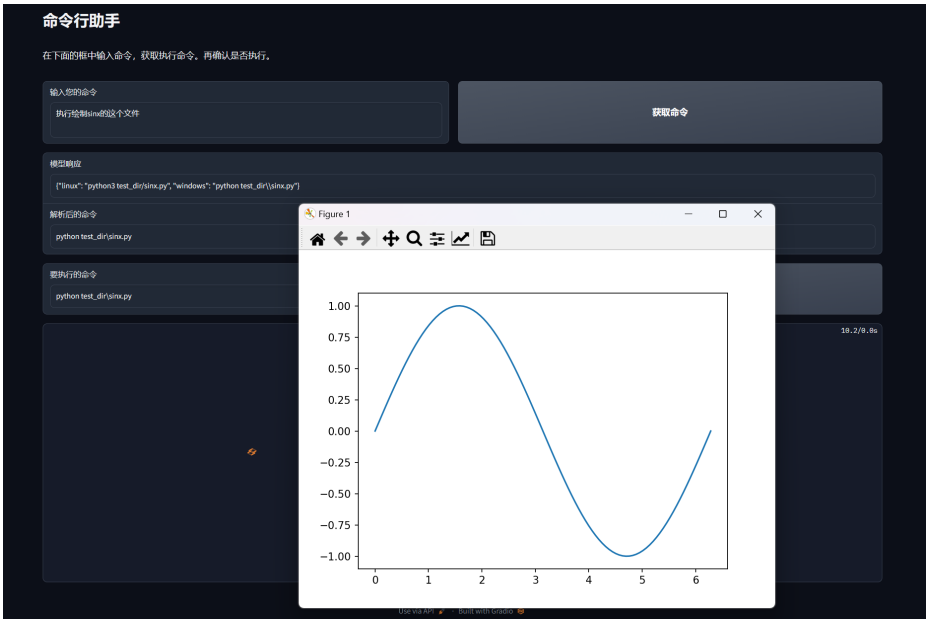
在这个测试中，我简单地把上一个案例的报错信息发送给命令行助手，命令行助手正确理解了上下文含义并且纠正了之前的错误，可以看到这里的测试结果是正确的，左边的目录结构也更新出了诗歌对应的文件temp.txt

要求命令行助手给出命令来输出刚刚创建的文件内容，甚至不需要说清楚是哪个文件，助手依据上下文自动判断

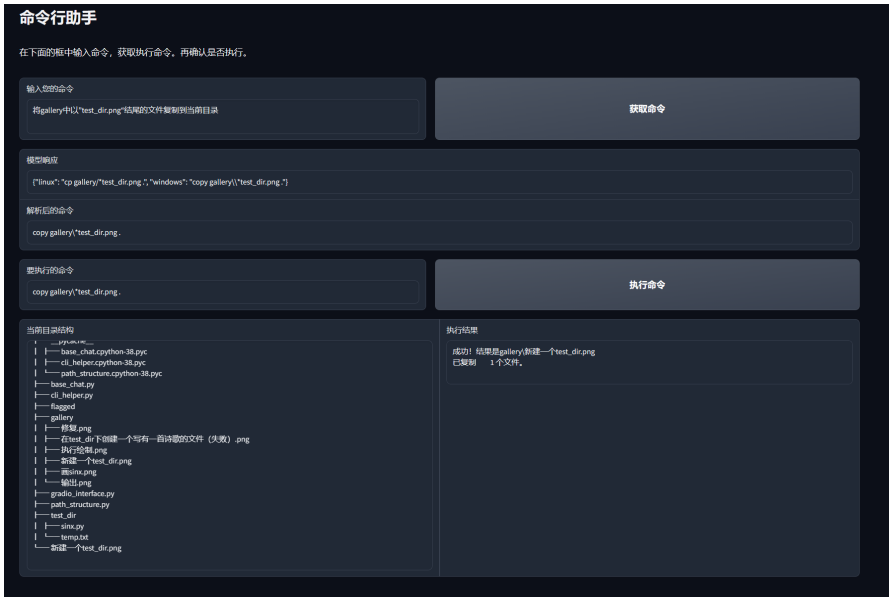
创建一个python文件用于绘制sinx图像



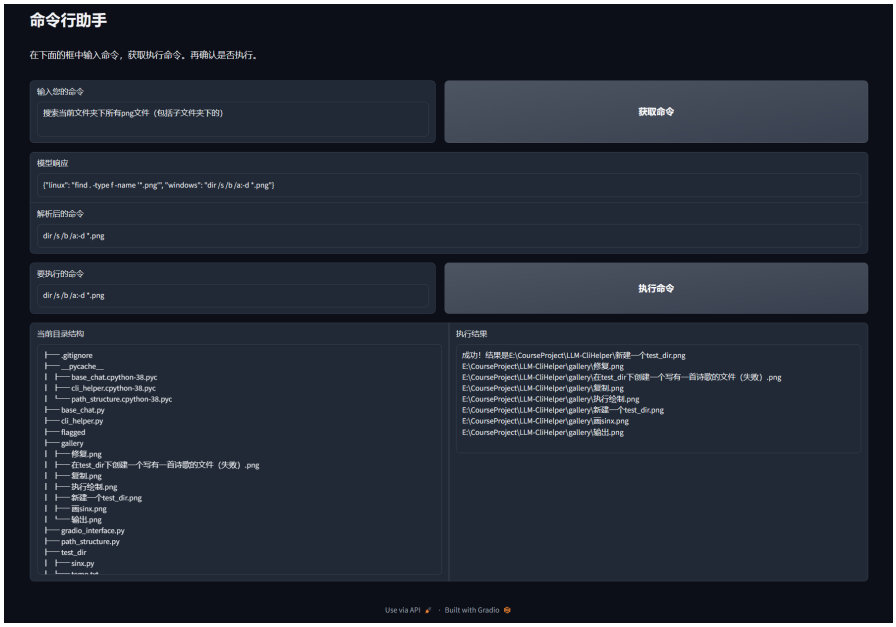
执行上个案例创建的python文件，可以看到正确调用了python程序来执行文件



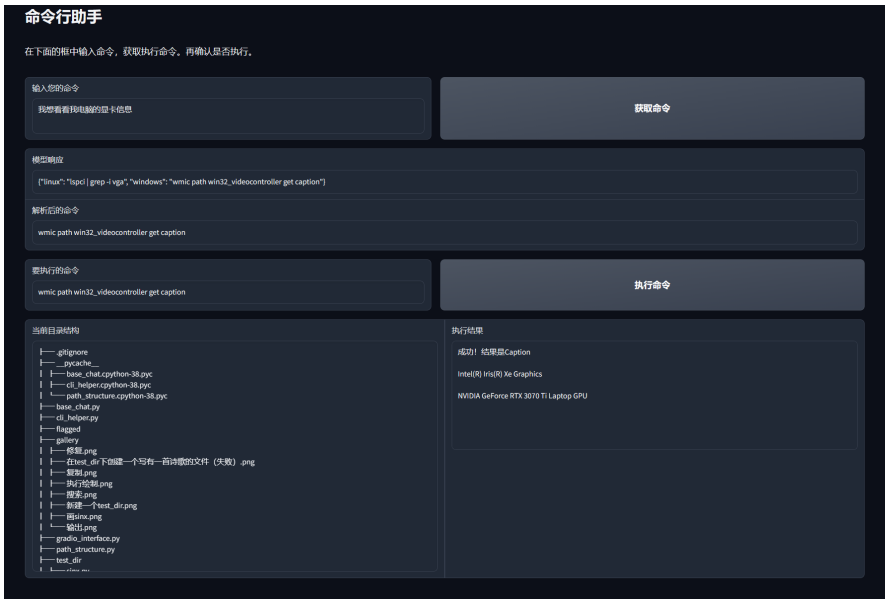
测试命令行助手的复制功能，这里复制了一张图片



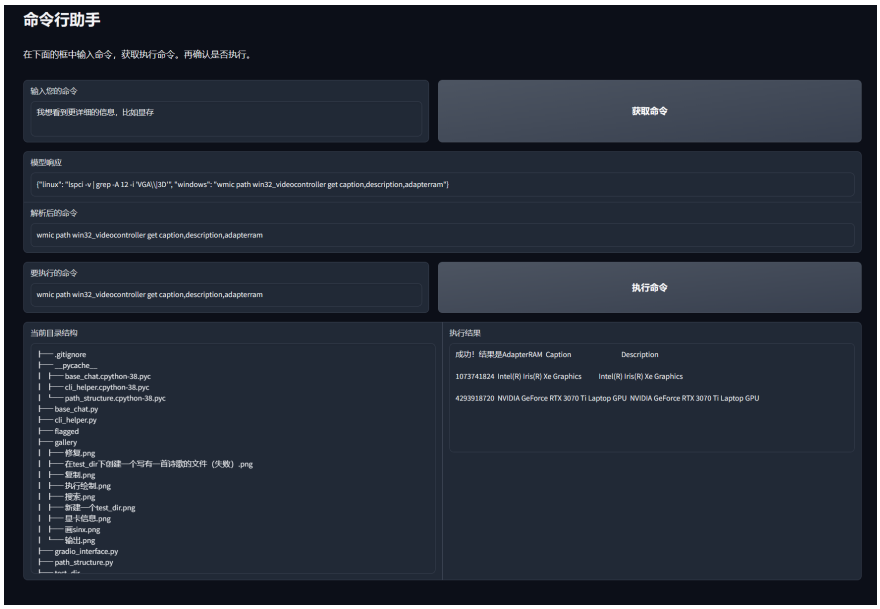
测试命令行助手搜索的能力，并且这个案例还需要命令行助手能够从项目结构中抽取符合用户需求的正则模式，可以看到执行结果是正确的



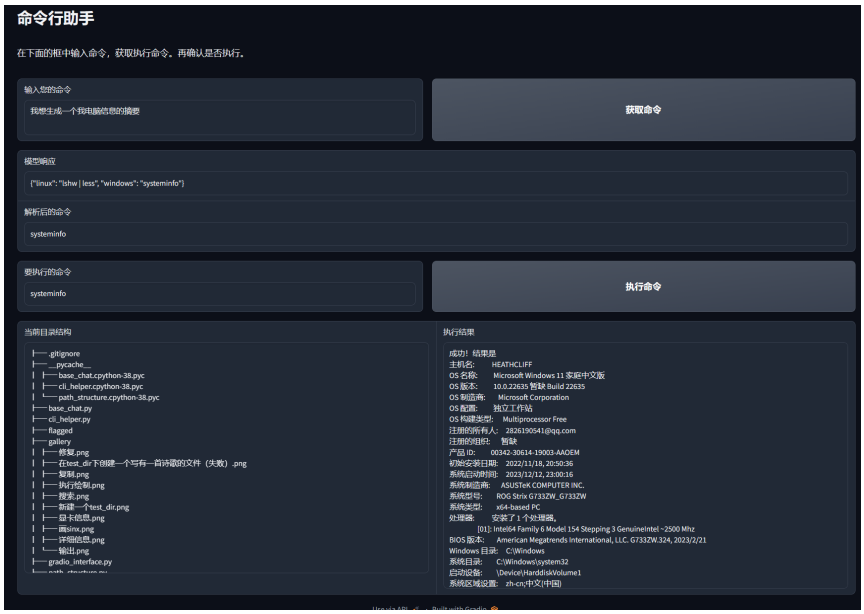
测试更加专业的命令，如这里的显卡信息



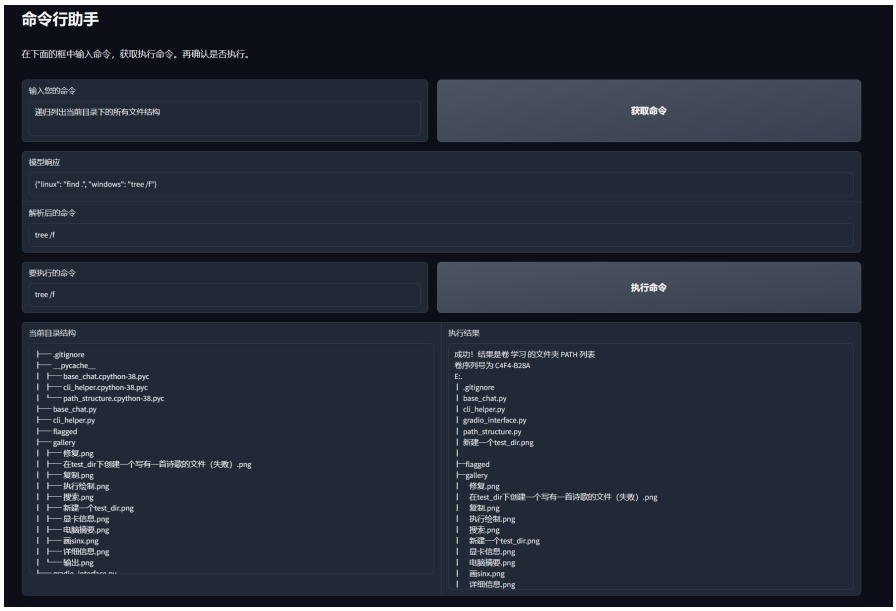
要求命令行助手给出一个能获取更加详细信息的命令



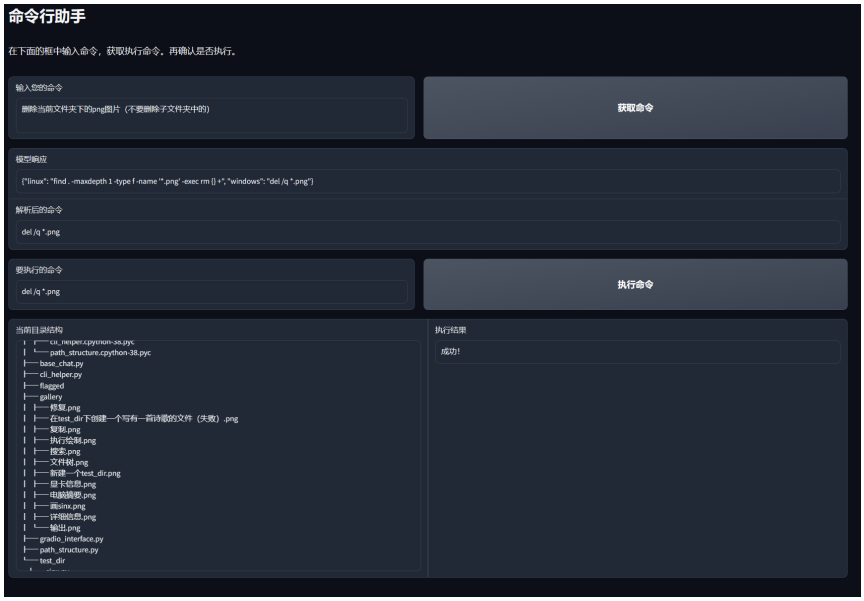
要求命令行助手给出一个命令生成“电脑信息的摘要”，这个需求比较模糊，但是语言模型正确理解了需求，找到了符合需求的命令，可以看到执行结果是正确的



要求命令行助手打印项目结构，和左边我编程实现的对比，可以看到结果是正确的



最后测试删除功能，删除前面案例复制到项目根目录的那张图片，成功执行。


命令行助手
在下面的框中输入命令，获取执行命令。再确认是否执行。
输入您的命令
删除当前文件夹下的png图片（不要删除子文件夹中的）
获取命令
模型响应
{"linux": "find . -maxdepth 1 -type f -name '*.png' -exec rm {} +", "windows": "del /q *.png"}
解析后的命令
del /q *.png
要执行的命令
del /q *.png
执行命令
当前目录结构
cli_helper.cpython-38.pyc
path_structure.cpython-38.pyc
base_chat.py
cli_helper.py
flagged
gallery
修复.png
在test_dir下创建一个写有一首诗歌的文件（失败）.png
复制.png
执行控制.png
搜索.png
文件树.png
新建一个test_dir.png
显卡信息.png
电脑配置.png
画sinx.png
详细信息.png
输出.png
gradio_interface.py
path_structure.py
test_dir
执行结果
成功!