项目 3

慧 答 助 手

本项目以 AI Agent 作为核心控制逻辑基础，借助 API 分别接入 SerpApi Google 搜索与智谱 AI 的 GLM-4 模型服务。通过智谱客户端的推理模块，负责处理 GLM-4 模型的调用以及响应生成工作。最终，依据 conversation_history 列表，达成知识库问答以及实时信息查询功能。

3.1 总体设计

本部分包括整体框架和系统流程。

3.1.1 整体框架

整体框架如图 3-1 所示。

图 3-1 整体框架

3.1.2 系统流程

系统流程如图 3-2 所示。运行流程对应代码中的 process_query()函数和主循环逻辑。步骤如下：①通过 determine_search_needed()函数判断是否需要搜索，如需搜索，则调用

web_search()函数获取实时信息，否则直接使用模型知识库。②通过 get_model_response()函数生成包含思考链的结构化回答，回答格式包含 Thought、Action、Action Input、Observation 和 Final Answer。③通过对话历史管理机制维护上下文，并将响应展示给用户。整个流程包含完整的错误处理机制，确保系统稳定运行。

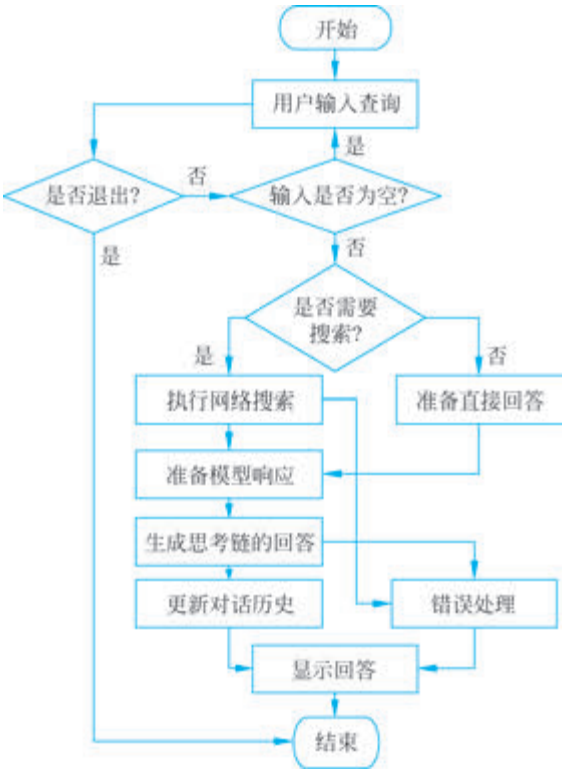

图 3-2　系统流程

3.2　开发环境

本部分介绍 Python 的安装过程，给出大模型 API 的申请步骤。

3.2.1　安装 Python

安装 Python 参见 1.2.1 节。

使用 VSCode＋Python 插件方式实现 Python 的编译，如图 3-3 所示。

访问 Anaconda，创建 Conda 环境，为以后执行任务需要不同版本的 Python 提供一个安全的环境配置的方法。

```
# 创建 Conda 环境(配置 d2l 环境所用的 3.9.20 版本,因此可以直接跳到激活环境)
conda create -n d2l python=3.9.20
```

```
#激活环境
conda activate d2l
```

图 3-3 安装 VSCode 的 Python 扩展

进入虚拟环境后，如图 3-4 所示。

图 3-4 进入虚拟环境

打开项目：File→Open Folder→D:\studydata\courses\ai_agent。选择 Python 解释器：Ctrl+Shift+P，输入 Python：Select Interpreter，选择创建的 Conda 环境。创建 Python 文件，保存为.py 格式，便可以在对应的 Python 文件中进行代码编译，本系统在文件夹中命名为 myagent.py，如图 3-5 所示。

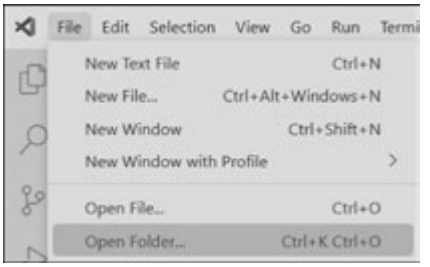

图 3-5 打开项目

3.2.2 安装 Python 包

本系统是通过调用大模型 API 实现的，实现编译软件安装后，只将必要的 Python 包进行安装即可实现系统的编译。必须安装的 Python 包如下。

```
pip install zhipuai                    #智谱 AI 的 SDK
pip install google-search-results      #SerpApi Google 搜索接口
pip install aiohttp                    #异步 HTTP 客户端/服务器端
pip install tenacity                   #重试机制库
```

3.2.3 智谱 API 申请

打开智谱 AI 大模型开放平台首页，如图 3-6 所示。

图 3-6　智谱 AI 大模型开放平台首页

单击右上角的“登录/注册”按钮，如图 3-7 所示。

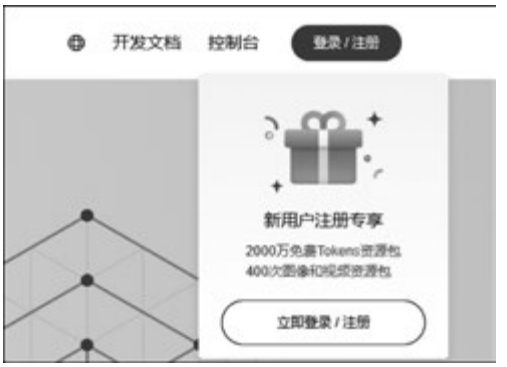

图 3-7　登录界面

新用户可以免费试用 API，但若未被赠送，则可充值小额后激活 API 实现调用。登录后在界面右上角找到 API Key 获取入口，单击进入。API Key 界面有默认项目的 API Key，起展示作用，无法复制和试用，需要添加新的 API Key，在给项目命名后复制自己的 API Key 并可开始满足调用需求。注意：不可公开泄露自己的 API。API Key 获取入口和新建按钮如图 3-8 所示。

图 3-8　API Key 获取入口和新建按钮

调用 API 需要判断 API Key 是否得到激活，从而实现正确调用，若未得到激活，会报错，若正确激活得到调用，则可以对代码错误进行其他问题的分析，如图 3-9 所示。

图 3-9　API Key 被正确激活调用的显示

3.2.4　SerpAPI 申请

打开 SerpAPI 官网首页，如图 3-10 所示。

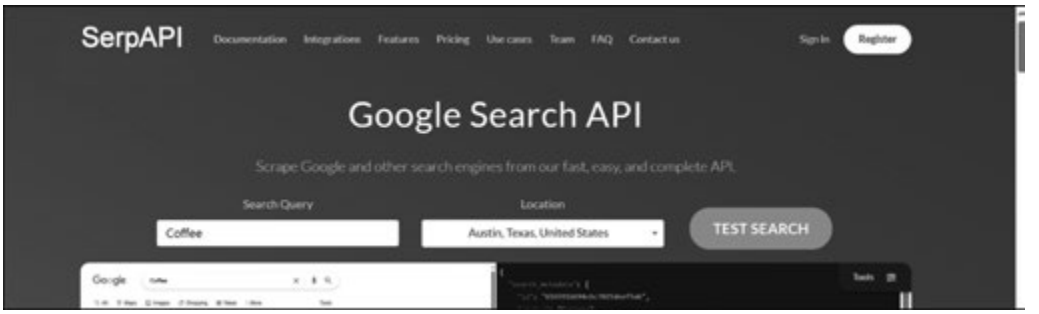

图 3-10　SerpAPI 官网首页

用谷歌邮箱进行登录，每人/月可以查询 100 次。首次申请需要进行邮箱验证和手机验证等。登录成功界面如图 3-11 所示。登录成功后即可直接进入，获取 API Key。

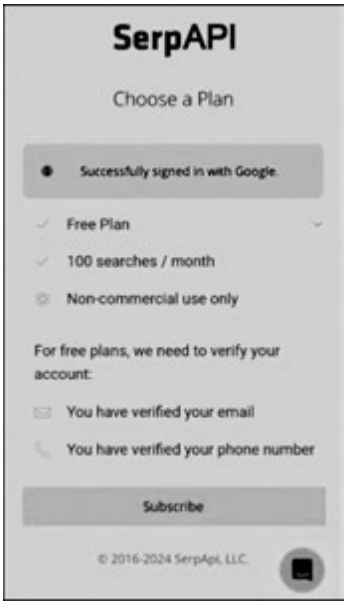

图 3-11　登录成功界面

3.3　系统实现

项目的组件包括 AI 模型交互、网络搜索功能和主程序入口。

3.3.1　AI Agent 类的初始化

初始化相关代码如下。

```
def _ _init_ _(self, api_key: str):
    self.client = ZhipuAI(api_key = api_key)        # 初始化智谱 AI 客户端
    self.serpapi_key = serpapi_key                  # 初始化 SerpApi Google 客户端
    self.conversation_history = []                  # 初始化对话历史列表
    self.max_history_length = 10                    # 设置最大对话历史长度
                                                    # 设置系统提示词,定义 AI 助手的行为模式
    self.system_prompt = """
    你是一个 AI 助手。请按照以下格式输出你的思考过程:
    Thought: [你对问题的思考]
    Action: [如果需要则是"Search",如果不需要则是"Direct Answer"]
    Action Input: [搜索关键词或直接回答的依据]
    Observation: [搜索结果或知识库信息]
    Final Answer: [最终的完整回答]
    """
```

3.3.2　网络搜索功能

网络搜索功能相关代码如下。

```
@retry(stop = stop_after_attempt(3), wait = wait_exponential(multiplier = 1, min = 4, max = 10))
    async def web_search(self, query: str) -> str:
        try:
            params = {
            "q": query,
            "api_key": self.serpapi_key,
            "engine": "google",
            "num": 3,                    # 限制 3 条结果
            "hl": "zh-cn",               # 设置语言为中文
          }
          search = GoogleSearch(params)
          results = search.get_dict()
          search_results = []
          # 处理搜索结果
          if "organic_results" in results:
              for result in results["organic_results"][:3]:
                  title = result.get('title', '').strip()
                  snippet = result.get('snippet', '').strip()
                  link = result.get('link', '').strip()
                  if title and snippet:
                      formatted_result = (
```

```
                    f" - 标题: {title}\n"
                    f" 摘要: {snippet}\n"
                    f" 来源: {link}"
                )
                search_results.append(formatted_result)
        return "\n\n".join(search_results) if search_results else ""
    except Exception as e:
        return f"搜索错误: {str(e)}"
```

3.3.3 模型响应处理

模型响应处理相关代码如下。

```
async def get_model_response(self, query: str, action: str, action_input: str, observation: str) -> str:
    try:
        #构建基础消息列表
        messages = [
            {"role": "system", "content": self.system_prompt},
            {"role": "user", "content": query}
        ]
        #构建思考链提示
        chain_prompt = f"""
        请基于以下信息,生成完整的思考链回答:
        问题: {query}
        执行动作: {action}
        动作输入: {action_input}
        观察结果: {observation}
        请严格按照以下格式输出:
        Thought: (思考过程)
        Action: {action}
        Action Input: {action_input}
        Observation: {observation}
        Final Answer: (最终回答)
        """
        messages.append({"role": "user", "content": chain_prompt})
        #调用 GLM-4 模型生成回答
        response = self.client.chat.completions.create(
            model = "glm-4",
            messages = messages,
            stream = True,
            temperature = 0.7,
            max_tokens = 2000
        )
        #处理流式响应
        full_response = ""
        for chunk in response:
          if chunk.choices[0].delta.content is not None:
              full_response += chunk.choices[0].delta.content
        return full_response
```

```
    except Exception as e:
        return f"生成回答时出错：{str(e)}"
```

3.3.4　搜索需求判断方法

搜索需求判断方法相关代码如下。

```
async def determine_search_needed(self, query: str) -> bool:
    #定义需要触发搜索的关键词
    explicit_search_keywords = {
        "最新", "现在", "最近", "新闻", "目前",
        "最新进展", "最新情况", "最新消息"
    }
    return any(keyword in query for keyword in explicit_search_keywords)
```

3.3.5　查询处理逻辑

查询处理逻辑相关代码如下。

```
async def process_query(self, query: str) -> str:
    try:
        #处理清除历史命令
        if query.lower() in ['清除历史', 'clear history']:
            self.conversation_history = []
            return "对话历史已清除"
        #判断是否需要搜索
        need_search = await self.determine_search_needed(query)
        if need_search:
            #执行搜索流程
            action = "Search"
            action_input = query
            observation = await self.web_search(query)
        else:
            #直接回答流程
            action = "Direct Answer"
            action_input = "使用模型知识库"
            observation = "基于内置知识"
        #生成回答
        response = await self.get_model_response(
            query=query,
            action=action,
            action_input=action_input,
            observation=observation
        )
        #更新对话历史
        self.conversation_history.append({"role": "user", "content": query})
        self.conversation_history.append({"role": "assistant", "content": response})
        #维护历史长度
        if len(self.conversation_history) > self.max_history_length:
            self.conversation_history = self.conversation_history[-self.max_history_
```

```
length:]
            return response
        except Exception as e:
            return f"处理查询时出错: {str(e)}"
```

3.3.6 主程序入口

主程序入口相关代码如下。

```
async def main():
    API_KEY = "0da40f0……3UpQyU"
    SERPAPI_KEY = "fd229d……d3c58d7" #添加 SerpApi API Key
    agent = AIAgent(api_key=API_KEY, serpapi_key=SERPAPI_KEY)
#修改初始化参数
    print("AI 助手已启动!输入'退出'结束对话")
    print("-" * 50)
    while True:
        try:
            #获取用户输入
            query = input("\n 请输入你的问题: ").strip()
            if not query:
                continue
            #处理退出命令
            if query.lower() in ['退出', 'exit', 'quit']:
                print("谢谢使用,再见!")
                break
            print("\n 思考中...")
            #处理查询并输出结果
            response = await agent.process_query(query)
            print(f"\n{response}")
            print("-" * 50)
        except KeyboardInterrupt:
            print("\n 程序已被用户中断。再见!")
            break
        except Exception as e:
            print(f"发生错误: {str(e)}")
if __name__ == "__main__":
    asyncio.run(main())
```

3.4 功能测试

本部分包括启动项目、发送问题及响应。

3.4.1 启动项目

在命令行运行主程序 main.py。虚拟环境和编译环境的确认在 VSCode 右下角,如图 3-12 所示。

图 3-12　当前所在编译环境(虚拟环境)

启动界面如图 3-13 所示。

图 3-13　启动界面

3.4.2　发送问题及响应

向大模型进行提问，输入问题后按回车键，如图 3-14～图 3-19 所示。

图 3-14　问题请求与回复

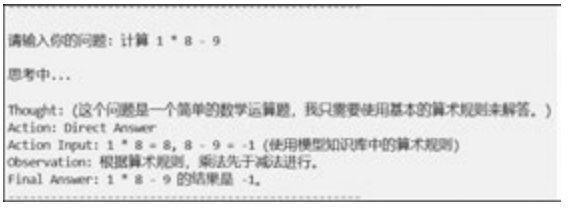

图 3-15　回答数学问题

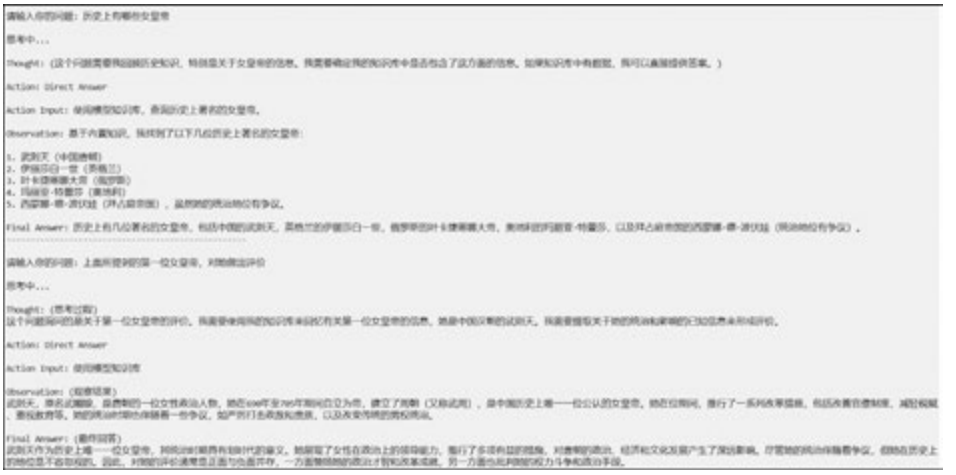

图 3-16　回答历史问题

图 3-17　实现计划制订

图 3-18　实现中英文翻译

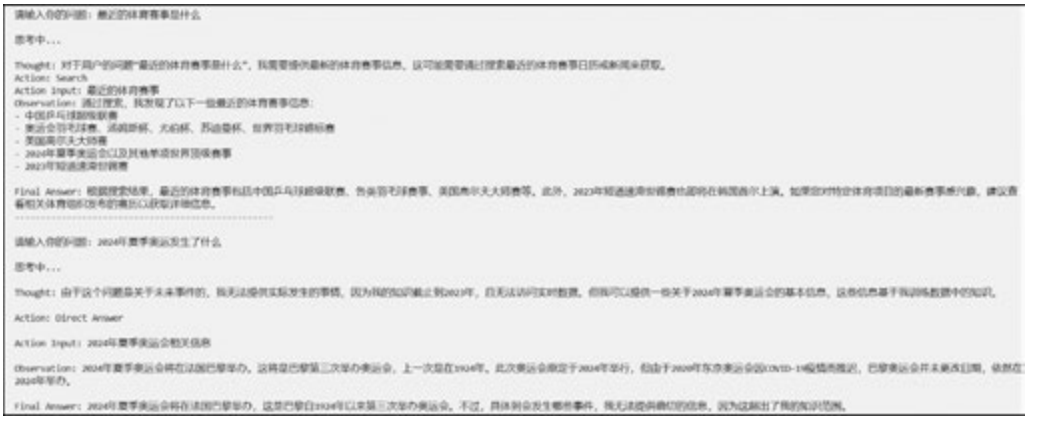

图 3-19　实现实时搜索