Deepseek企业级Agent项目开发实战

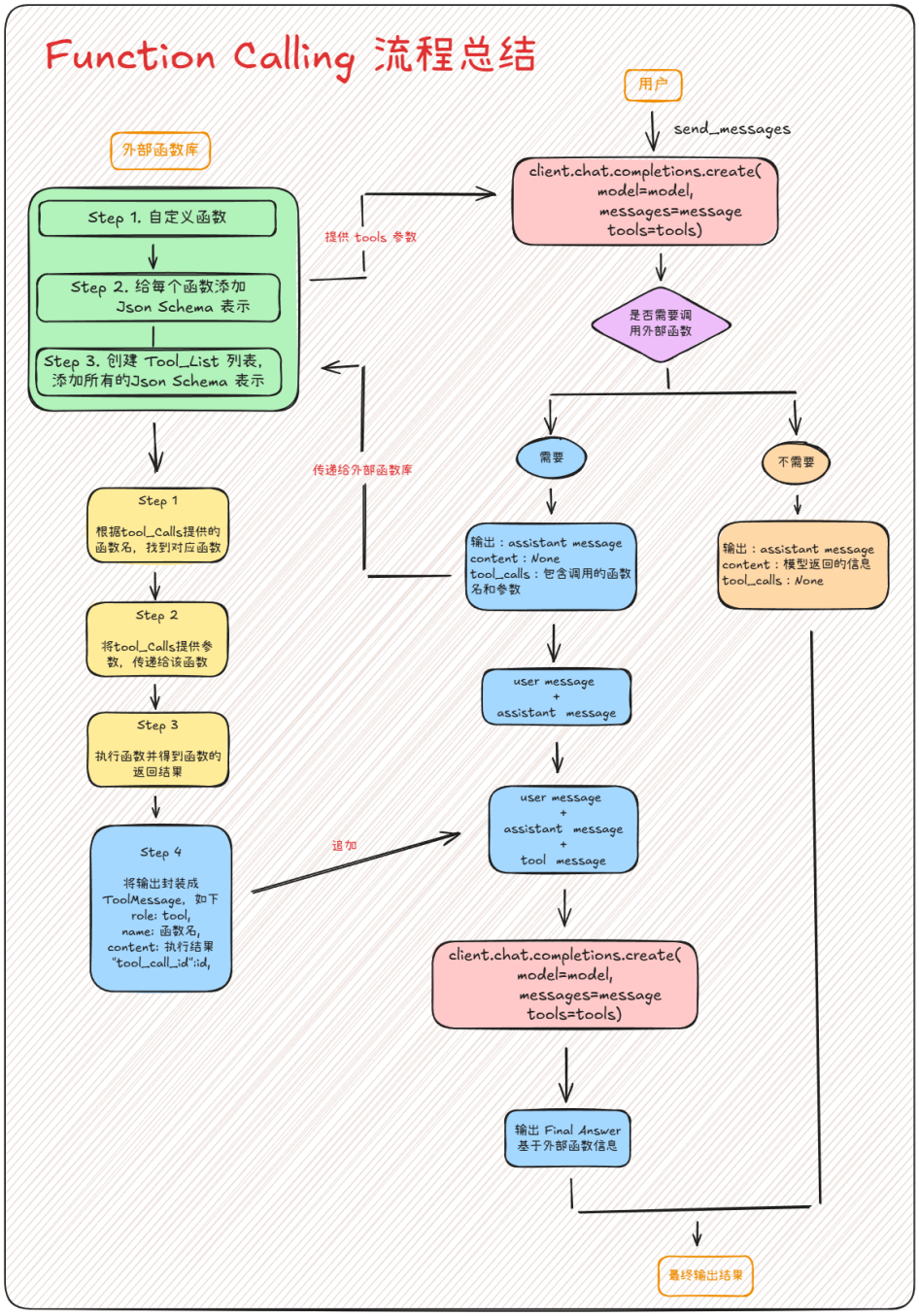
Part 12. DeepSeek Function Calling 功能调用与联网检索

大模型在训练时使用大量的数据进行训练，一旦训练完成，其能回答的问题范围就仅限于训练数据中的问题，无法满足实际应用中对最新、实时知识和问答需求。因此在`2023年7月`，`OpenAI`为其`GPT`系列模型推出了`Function Calling`功能，允许大模型在对话中调用外部工具，这个外部工具可以是连接数据库、搜索引擎、API等多种不同的形式。随后，基本上所有的大模型如`GLM-4`、`Qwen`等都陆续开始支持`Function Calling`功能，同样，`DeepSeek V3`也支持`Function Calling`功能，并且和`OpenAI`的`Function Calling`功能在使用规范上完全兼容。

因此这里大家要清楚的概念是：`Function Calling`指的是函数调用，也就是允许大模型调用外部的工具（比如`Python`函数封装）。而所谓的支持`Function Calling`功能，它并不是大模型完全自动实现调用 + 响应的整个过程，而是需要开发者根据大模型的输出结果，然后通过手动实现的方式来完成这个流程，具体来说：

1. 常规对话类模型无论用户输入什么问题，大模型都会根据用户的问题给出回答，这个回答是基于大模型自身的知识库给出的。具备`Function Calling`功能的大模型，当用户的问题涉及到了某个外部工具的应用需求点，那么大模型会返回一个`Function Calling`的输出，这个输出告诉开发者，需要调用哪个外部工具，以及调用这个外部工具需要传入的参数是什么。
2. 开发者根据大模型返回的`Function Calling`的输出，执行这个外部函数，并且传入这个外部函数需要用到的参数，获取到外部函数执行后的返回结果。
3. 手动获取到外部工具的返回结果后，将结果与用户的原始问题一起给到大模型，大模型根据新的输入给出最终的回答。

`Function calling`的实现完整流程如下图所示：



1. DeepSeek Function Calling 功能兼容情况

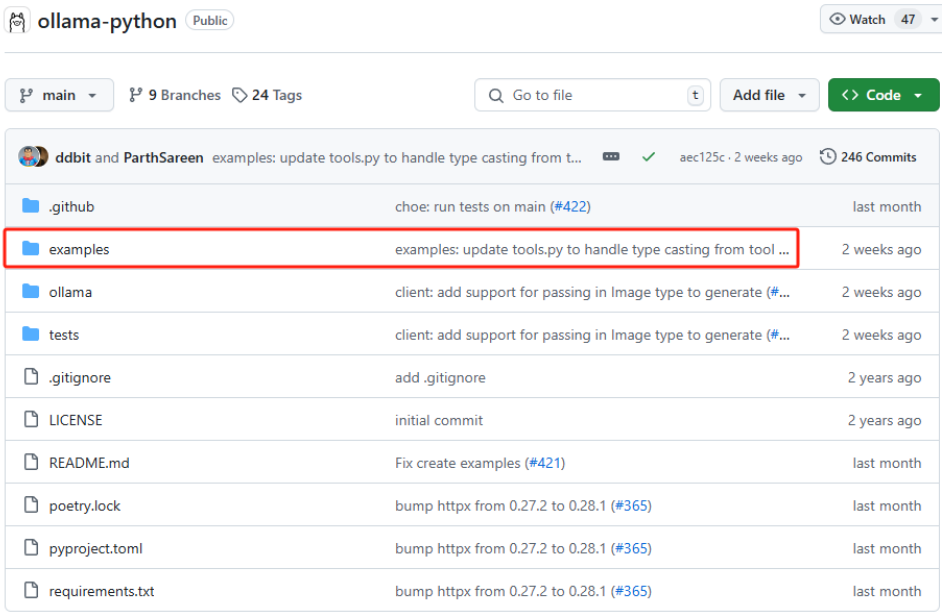
`DeepSeek` 官方的 `API` 目前包含两类模型，分别是 `DeepSeek-chat` 和 `DeepSeek-reasoner`，其中两类模型的区别与 `Function Calling` 功能支持情况如下：

Function Calling 功能支持情况

模型名称	底层模型	支持 Function Calling
DeepSeek-chat	满血版 DeepSeek v3	是
DeepSeek-reasoner	满血版 DeepSeek r1	否

除了官方的`API`之外，基于`Ollama`框架启动的本地模型也是支持`Function Calling`功能的。但是无法使用原生的`REST API`来调用，比如`/api/generate`和`/api/chat`，而是需要使用基于`Ollama`原生的`REST API`的上层封装来进行调用，主要有两种方式：

1. OpenAI compatibility 兼容 API；
2. 集成`Ollama`服务的 Python Library 或 JavaScript Library，开源地址：https://github.com/ollama/ollama-python



使用更加广泛的还是`OpenAI`兼容的`API`方式，我们对`Ollama`启动的`DeepSeek-R1`模型也进行了`Function Calling`功能的测试，在这段代码中，我们定义了一个`Chatbot`类，用于与`DeepSeek-R1`模型进行对话，同时定义了`create_tools`函数，用于创建工具定义，`get_weather`函数用于模拟获取天气信息。完整代码如下：

```
import json
from openai import OpenAI

class Chatbot:
    def __init__(self, model: str, base_url: str, api_key: str):

        # 初始化 OpenAI 客户端，该方式可以兼容在线API，同时也兼容使用 `ollama` 启动的 REST
API
        self.client = OpenAI(
            base_url=base_url,
            api_key=api_key
        )
        self.model = model

    # 定义外部工具库
    # 1. 自定义函数
    def get_weather(self, location: str) -> str:
        """获取实时的天气数据，包括天气状况、温度"""
        weather_data = {
            "北京": "晴天，气温 25°C",
            "上海": "多云，气温 22°C",
            "广州": "小雨，气温 28°C",
            "深圳": "阴天，气温 27°C",
        }
        return weather_data.get(location, "城市未找到，无法提供天气信息。")
```

```
    # 2. 创建自定义函数的 JsonSchema 表示
    def create_tools(self):
        """创建工具定义"""
        return [
            {
                "type": "function",
                "function": {
                    "name": "get_weather",
                    "description": "Get real-time weather data, including
weather conditions, temperatures",
                    "parameters": {
                        "type": "object",
                        "properties": {
                            "location": {
                                "type": "string",
                                "description": "The city name, e.g. Beijing",
                            }
                        },
                        "required": ["location"]
                    },
                }
            },
        ]

    # 对话主函数
    def chat(self, user_message: str):
        """
        对话函数
        """
        # 添加系统角色消息
        messages = [
            {"role": "system", "content": "你是一个智能助手，能够回答用户的问题并提供帮
助。"},
            {"role": "user", "content": user_message}
        ]

        print('用户输入\t:', user_message)

        response = self.client.chat.completions.create(
            model=self.model,
            messages=messages,
            tools=self.create_tools()    # 通过tools参数传递工具的 JsonSchema 表示
        )

        # 检查 finish_reason
        finish_reason = response.choices[0].finish_reason

        if finish_reason == 'stop':
            # 普通问答处理
            print("模型输出\t:", response.choices[0].message.content)

        elif finish_reason == 'tool_calls':
            assistant_message = response.choices[0].message  # 获取 assistant 的消
息
            print('参数解析\t:',
response.choices[0].message.tool_calls[0].function.arguments)

            # 将 assistant 的消息添加到 messages 列表中
```

```
            messages.append(assistant_message)

            # 处理工具调用
            tool_calls = response.choices[0].message.tool_calls
            if tool_calls:
                # 获取函数名称和参数
                function_name = tool_calls[0].function.name
                function_args = json.loads(tool_calls[0].function.arguments)

                # 定义可用的函数
                available_functions = {
                    "get_weather": self.get_weather,
                    # 可以在这里添加更多的函数
                }

                # 动态调用相应的函数
                if function_name in available_functions:
                    function_response = available_functions[function_name]
(**function_args)
                    print(f"执行结果: {function_response}")
                    # 添加工具调用的响应到消息中
                    messages.append({
                        "role": "tool",
                        "content": str(function_response),
                        "tool_call_id": tool_calls[0].id,
                    })

                    final_response = self.client.chat.completions.create(
                        model=self.model,
                        messages=messages,
                    )

                    print('最终回复\t:',
final_response.choices[0].message.content)
                else:
                    print(f"Function {function_name} is not available.")
            else:
                print("No tool calls returned from model")
        else:
            print("Unknown finish reason:", finish_reason)
```

先测试一下`DeepSeek`官方的`API`对`Function Calling`功能的支持情况，测试代码如下：

```
import os
from dotenv import load_dotenv

# 加载 .env 文件
load_dotenv()


# 直接传入模型名称、base_url 和 api_key
api_key=str(os.getenv('DEEPSEEK_API_KEY'))
base_url=str(os.getenv('DEEPSEEK_BASE_URL'))
model_name = 'deepseek-chat'  # 模型名称

user_message = '北京的天气怎么样？'  # 输入
chatbot = Chatbot(model_name, base_url, api_key)
```

```
chatbot.chat(user_message)
```

```
用户输入     : 北京的天气怎么样?
参数解析     : {"location":"北京"}
执行结果: 晴天，气温 25°C
最终回复     : 今天北京的天气是晴天，气温为25°C。
```

从测试结果可以验证，`DeepSeek` 官方的 `API` 确实是支持 `Function Calling` 功能的。接下来我们再测试 `Ollama` 启动的 `DeepSeek-R1` 模型，测试代码如下：

```
# 直接传入模型名称、base_url 和 api_key
base_url = 'http://192.168.110.131:11434/v1/'  # REST API 的 endpoint， 需要修改成可访问的 IP
api_key = 'ollama'  # 随便写，但是api_key字段一定要有
model_name = 'deepseek-r1:32b'  # 模型名称
user_message = '请告诉我北京的天气。'  # 输入

chatbot = Chatbot(model_name, base_url, api_key)
chatbot.chat(user_message)
```

从测试结果可以看到，`'message': 'registry.ollama.ai/library/deepseek-r1:32b does not support tools` 表示 `Ollama` 启动的 `DeepSeek-R1` 模型不支持 `Function Calling` 功能。

接下来可以测试 `Ollama` 启动的其他系列模型，这里我们使用 `Qwen2.5:7b` 模型，测试代码如下：

```
# 直接传入模型名称、base_url 和 api_key
base_url = 'http://192.168.110.131:11434/v1/'  # REST API 的 endpoint， 需要修改成可访问的 IP
api_key = 'ollama'  # 随便写，但是api_key字段一定要有
model_name = 'qwen2.5:32b'  # 模型名称
user_message = '请告诉我北京的天气。'  # 输入

chatbot = Chatbot(model_name, base_url, api_key)
chatbot.chat(user_message)
```

```
用户输入     : 请告诉我北京的天气。
参数解析     : {"location":"北京"}
执行结果: 晴天，气温 25°C
最终回复     : 当前北京的天气是晴朗的，温度为25°C。
```

从测试结果看，`Ollama` 集成的 `OpenAI` 兼容的 `API` 是支持 `Function Calling` 功能的。只不过 `deepseek-r1` 模型不支持，其他的主流对话模型如 `qwen2.5`、`llama3.3` 等都是支持的，这点大家需要特别注意。