01-调用ollama本地部署的模型

说明: 不使用 langchain 框架的 API

Ollama 提供了 REST API 用于运行和管理模型。

1、使用 request 库来实现调用（RESTful API）

- **使用的库**：使用 `requests` 库，直接调用 Ollama 的 RESTful API 进行通信。
- **交互方式**：通过发送 POST 请求到 `http://localhost:11434/api/generate` 端点，进行单次文本生成。
- **消息管理**：没有维护消息列表，仅发送一个固定的提示信息。
- **模型调用**：在请求数据中指定模型、提示信息，并开启流式响应。
- **输出处理**：对响应进行流式处理，将接收到的分块数据累加到缓冲区，按行解析 JSON 数据，直接输出 `response` 字段的内容。

Python

```python
#!/usr/local/bin/python3
# -*- coding: utf-8 -*-
'''
@File    :   ollama_inference.py
@Time    :   2025/02/21 23:34:14
@Author  :   jason
@Version :   0.1
@Desc    :   None
'''
import requests
import json
import time

if __name__ == '__main__':
    model_url = 'http://localhost:11434/api/generate'

    data = {
        "model": "deepseek-r1:14b",
        "prompt": "你好,帮我写一首春天的诗,直接给答案",
        "stream": True
    }

    response = requests.post(url=model_url, json=data, stream=True)

    # 初始化一个缓冲区，用于累积分块数据
    buffer = ""
    # Assuming the server sends data in chunks
    for chunk in response.iter_content(chunk_size=1024, decode_unicode=True):
        if chunk:
            buffer += chunk.decode('utf-8')                                  # 将分块数据解码为字符串，并添加到缓冲区
            while '\n' in buffer:                                            # 检查缓冲区中是否有完整的 JSON 对象（以换行符分隔）
                line, buffer = buffer.split('\n', 1)                         # 从缓冲区中提取一个完整的 JSON 对象（以换行符为界）
                if line.strip():                                             # 如果提取的行不为空，则尝试解析 JSON
                    try:
                        chunk_data = json.loads(line)                        # 解析 JSON 对象
                        if chunk_data['response']:                           # 如果 JSON 对象中包含 "response" 字段，则打印它
                            print(chunk_data['response'], end=' ',flush=True)   # Print the 'response' field from the JSON
```

```
                    time.sleep(0.05)
                except json.JSONDecodeError as e:
                    print(f"Error decoding JSON: {e}")
```

2、使用 openai来实现调用

- **使用的库**：使用 `openai` 库，通过 OpenAI API 风格的接口与 Ollama 服务通信。
- **交互方式**：模拟 OpenAI 的聊天对话模式，使用 `client.chat.completions.create` 方法
- **消息管理**：维护一个消息列表 `messages`，其中包含系统消息和用户与助手的历史对话。每次用户输入后，将其添加到消息列表中，并在获取响应后将助手的回复也添加到列表中。
- **模型调用**：指定模型为 `deepseek-r1:14b`，设置了温度和最大令牌数等参数，并开启流式响应。
- **输出处理**：对响应进行流式处理，去除 `<think></think>` 标签和空行后实时输出。

```python
#!/usr/local/bin/python3
# -*- coding: utf-8 -*-
'''
@File    :   ollama_openai_inference.py
@Time    :   2025/02/22 20:23:23
@Author  :   jason
@Version :   0.1
@Desc    :   None
'''
from openai import OpenAI
import re
import time

client = OpenAI(
    base_url = 'http://localhost:11434/v1',
    api_key = 'ollama'
)

messages =[
    {"role": "system", "content": "You are a helpful assistant."}
]

def chat_with_deepseek(user_input):
    messages.append({"role": "user", "content": user_input})
    print(f"messages : {messages}")
    response = client.chat.completions.create(
        model = 'deepseek-r1:14b',
        messages = messages[-6:],
        temperature = 0.9,
        max_tokens = 4096,
        stream = True
    )

    assistant_response = ""
    print("Deepseek:", end='', flush=True)

    for chunk in response:
        if chunk.choices[0].delta.content:
            content = chunk.choices[0].delta.content
            assistant_response += content

            # 去除<think></think>标签
            clean_content = content.replace("<think>", "").replace("</think>", "")
            # 清除空行
```

```
                clean_content = "\n".join([line for line in clean_content.splitlines() if line.strip()])
                # 输出清理后的内容
                print(clean_content, end='', flush=True)
                time.sleep(0.05)
        print("")
        # 将助手的响应添加到消息列表中
        messages.append({"role": "assistant", "content": assistant_response})

if __name__ == "__main__":
    while True:
        user_input = input("jason: ")
        if user_input.lower() in ['exit','quit']:
            print("对话结束")
            break
        chat_with_deepseek(user_input)
```

3、聊天交互模式（RestFul API）

Ollama 作为一个模型运行框架，通过 REST API 提供了两种与模型交互的主要方式：

1. **生成回复（Generate API）**
   - **端点**：`/api/generate`
   - **用途**：向模型发送一个提示并获取一次性回复，适用于简单问答、内容生成等场景
   - **特点**：
     - 只支持单次输入 – 输出
     - 没有对话历史的概念
     - 适合简单的生成任务，如文本补全、诗歌创作等
   - **参数**：
     - `model`：指定使用的模型名称
     - `prompt`：用户输入的提示文本
2. **对话交互（Chat API）**
   - **端点**：`/api/chat`
   - **用途**：实现多轮对话交互，支持上下文理解
   - **特点**：
     - 支持对话历史，模型可以基于历史消息生成回复

        - 通过 `messages` 数组维护对话上下文
        - 更适合聊天机器人、客服助手等需要连续性的场景
    - **参数**：
        - `model`：指定使用的模型名称
        - `messages`：对话历史数组，每个消息包含：
            - `role`：角色（user/assistant/system）
            - `content`：消息内容

3. 对比说明

- **适用场景**：
    - `generate` 适合一次性任务，如生成文章、代码等
    - `chat` 适合需要上下文的多轮对话，如聊天、问答系统
- **对话状态管理**：
    - `generate` 每次请求独立，不维护状态
    - `chat` 需要客户端维护完整的对话历史并在每次请求时发送

4. 使用建议

- 对于简单的生成任务，优先使用 `/api/generate`
- 对于需要多轮对话的场景，使用 `/api/chat` 并维护好 `messages` 数组
- 生产环境中建议使用流式响应（添加 `"stream": true` 参数）以获得更好的用户体验

Python

```python
#!/usr/local/bin/python3
# -*- coding: utf-8 -*-
'''
@File    :   ollama_restfulapi_chat.py
@Time    :   2025/02/22 21:49:37
@Author  :   jason
@Version :   0.1
@Desc    :   None
'''
import requests
import json
import time

messages = [
    {"role": "assistant","content": "You are a helpful assistant."}
]

def ollama_restfulapi_chat(user_input):
    api_url = 'http://localhost:11434/api/chat'
    messages.append({"role": "user","content": user_input})
    print(f"messages: {messages}")
    data = {
        "model": "deepseek-r1:14b",
        "messages": messages[-5:],
        "temperature": 0.9,
        "max_tokens": 4096,
        "stream":True
    }
    response = requests.post(api_url, json=data, stream=True)

    buffer = ""
    assistant_response = ""  # 用于存储完整的助手回复
    print('assistant: ', end='', flush=True)
    for chunk in response.iter_content(chunk_size=1024, decode_unicode=Tru
e):
        if chunk:
            buffer += chunk.decode('utf-8')
            while '\n' in buffer:
                line, buffer = buffer.split('\n', 1)
                if line.strip():
                    try:
                        chunk_data = json.loads(line)
                        if chunk_data['message']:
                            message = chunk_data['message']

```

```
                                content = message['content'].replace('<think>'
,'').replace('</think>', '')
                                #清楚空行
                                content = '\n'.join([line for line in content.
splitlines() if line.strip()])
                                print(f"{content}", end="", flush=True)
                                time.sleep(0.05)
                                assistant_response += content       # 将清理后
的内容添加到完整回复中
                    except json.JSONDecodeError as e:
                        print("Error decoding JSON:", line)

    # 将助手的回复添加到消息列表中
    messages.append({"role": "assistant", "content": assistant_response})

if __name__ == '__main__':
    while True:
        user_input = input("You: ")
        if user_input.lower() in ['exit','quit']:
            print("Chat ended.")
            break
        ollama_restfulapi_chat(user_input)
        print("")
```