实验四 智能模型驱动

1. 实验目标

- 了解智能模型驱动相关概念
- 掌握基于大语言模型辅助的建模方法，基于相关框架构建MultiAgent Workflow实现自动化需求建模。

2. 实验内容

- **实验准备：** 相关环境配置、选定目标模型
- **任务 1：** 基于纯 Restful API 的智能化需求建模
- **任务 2：** 基于 OpenAI SDK 的智能化需求建模
- **任务 3：** 基于 LLM Agent 的智能化需求建模

3. 实验材料

- BASE_URL：https://api.chatfire.cn/v1
- API_KEY：sk-zO8exlBicZh7nJeZn5GuC5X9SPuVrZzXoGyOW0i9BFvN62ON
- 注意：**API_KEY相关信息仅限本课程使用，不要提供给本课程之外的人员。**

4. 作业要求

- 相关代码上传到Github或Gitee中，仓库需要设置为**Public**可见，并将**项目链接**填写到问卷中。
- 实验报告（以项目README文件形式给出），包含**输入的Prompt**，**输出格式定义**，MultiAgent Workflow**简要说明**，**生成的需求模型说明**，不要求具体格式。
- **注意：本次实验最晚提交时间为 2025 年 6 月 23 日 0 点!**

5. 实验准备：相关环境配置、选定目标模型

- **目标环境配置：** 选择MultiAgent Workflow框架，本文档以openai agents为例，基于Python环境，安装方式参考如下，具体可阅读官方文档：OpenAI Agents SDK。

```
pip install openai-agents
```

- **选定目标模型：** 参考RM2PT Case Study（北航云盘），自选目标系统或者在已有案例基础上进行扩展，**可以继续使用实验一选择的目标模型。**

6. 任务 1

- 阅读 OpenAI 官方 API 文档，理解请求格式与响应格式。
- 设计 Prompt 以及期望的模型输出格式，输出格式可以自行设计，能够清晰的表达需求模型即可。
- 使用 Postman、curl 或 Python 等工具调用 OpenAI 的 API 生成需求模型，简单案例**参考9.1。**
- 使用Postman、curl等工具调用在报告中提供相关截图以及生成结果即可。

7. 任务 2

- 阅读 OpenAI 官方 API 文档，了解 **OpenAI SDK** 的使用。
- 设计 Prompt 以及期望的模型输出格式，输出格式可以自行设计，能够清晰的表达需求模型即可。

- 使用 **OpenAI SDK** 调用 OpenAI 的 API 生成需求模型，简单案例**参考 9.2。**

8. 任务 3

设计和定义MultiAgent Workflow、Agent对应的System Prompt以及Agent输出格式（DSL），构建MultiAgent Workflow，实现自动化领域建模。

- 设计MultiAgent协同的流程、Agent对应的System Prompt。
- 设计用于Agent输出的DSL，可以使用json或其他形式，需要包含现有需求模型完整内容，包括用例图、系统顺序图、概念类图以及OCL合约。
- 以用例图为例，可以定义如下的DSL（仅供参考）

```
{
    "name": "CoCoME",
    "usecases": [
        {
            "name": "OpenStore",
            "includes": [],
            "extends": []
        }
        {
            "name": "CloseStore",
            "includes": [],
            "extends": []
        }
        {
            "name": "manageStore"
            "includes": ["OpenStore", "CloseStore"],
            "extends": []
        }
    ]
}
```

- 定义和实现需要使用的外部tool。
- 基于相关框架实现MultiAgent Workflow，本文档以openai agents为例，后附简要使用说明，详细信息可阅读官方文档：OpenAI Agents SDK。**可以使用其他任意框架或方法实现。**

9 案例

9.1 Restful API 调用

```
curl 'base_url/chat/completions' \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer api_key" \
  -d '{
    "model": "gpt-4o",
    "messages": [
      {
        "role": "developer",
        "content": "You are a helpful assistant."
      },
      {
        "role": "user",
        "content": "Hello!"
      }
    ]
  }'
```

9.2 OpenAI SDK 调用

```python
from openai import OpenAI
from openai.types.chat import ChatCompletion

BASE_URL: str = "base_url"
API_KEY: str = "api_key"

client: OpenAI = OpenAI(base_url=BASE_URL, api_key=API_KEY)
completion: ChatCompletion = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "developer", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Hello!"},
    ],
)

print(completion.choices[0].message)
```

9.3 OpenAI Agent

9.3.1 Agent Function Call

- 定义tool

```python
import asyncio
from agents import Agent, Runner, function_tool

@function_tool
def get_weather(city: str) -> str:
    return f"The weather in {city} is sunny."
```

- 定义Agent，添加tool

```python
agent = Agent(
    name="Hello world",
    instructions="You are a helpful agent.",
    tools=[get_weather],
    model="gpt-4o"
)

provider: OpenAIProvider = OpenAIProvider(
    openai_client=AsyncOpenAI(base_url="base_url", api_key='api_key'),
    use_responses=False,
)
```

- 运行Agent

```python
async def main():
    result = await Runner.run(agent,
        input="What's the weather in Beijing?",
        run_config=RunConfig(model_provider=provider)
    )
    print(result.final_output)

if __name__ == "__main__":
    asyncio.run(main())
```

9.3.2 MultiAgent Workflow

- 定义输出格式

```python
@dataclass
class EvaluationFeedback:
    score: Literal["pass", "needs_improvement", "fail"]
    feedback: str
```

- 定义多个Agent

```
  story_outline_generator: Agent = Agent(
    name="story_outline_generator",
    model="gpt-4o",
    instructions=(
        "You generate a very short story outline based on the user's input."
        "If there is any feedback provided, use it to improve the outline."
  ),
)

evaluator: Agent = Agent(
    name="evaluator",
    model="gpt-4o",
    instructions=(
        "You evaluate a story outline and decide if it's good enough."
        "If it's not good enough, you provide feedback on what needs to be improved."
        "Never give it a pass on the first try."
    ),
    output_type=EvaluationFeedback,
)

provider: OpenAIProvider = OpenAIProvider(
    openai_client=AsyncOpenAI(
        base_url="base_url",
        api_key="api_key",
    ),
    use_responses=False,
)
```

- 定义和执行工作流

```
async def main() -> None:
    msg: str = input("What kind of story would you like to hear? ")
    input_items: list[TResponseInputItem] = [{"content": msg, "role": "user"}]
    latest_outline: str | None = None

    with trace("LLM as a judge"):
        while True:
            story_outline_result: RunResult = await Runner.run(
                story_outline_generator,
                input_items,
                model_provider=provider
            )

            input_items: list[TResponseInputItem] = story_outline_result.to_input_list()
            latest_outline = story_outline_result.final_output_as(str)

            print("Story outline generated")
            print(latest_outline)

            evaluator_result: RunResult = await Runner.run(evaluator, input_items, model_provider=provider)
            result: EvaluationFeedback = evaluator_result.final_output
            print(f"Evaluator score: {result.score}")

            if result.score == "pass":
                print("Story outline is good enough, exiting.")
                break

            print(result.feedback)
            print("Re-running with feedback")

            input_items.append({"content": f"Feedback: {result.feedback}", "role": "user"})

    print(f"Final story outline: {latest_outline}")

if __name__ == "__main__":
    asyncio.run(main())
```