项目报告

基于多智能体协作的旅行规划系统

项目报告

一、项目概述

1.1 项目背景

随着人工智能技术的快速发展，多智能体系统（Multi-Agent System, MAS）在复杂任务求解中展现出巨大潜力。旅行规划作为一个典型的复杂决策问题，涉及交通、住宿、景点、餐饮等多个维度的约束优化，非常适合采用多智能体协作的方式来解决。

本项目基于微软开源的 AutoGen 框架，设计并实现了一个智能旅行规划系统。系统通过多个专门化的 AI Agent 协作，能够根据用户需求自动生成满足预算约束和时间限制的旅行方案。

1.2 项目目标

1. **设计多智能体协作框架**：构建包含协调、研究、规划、反馈、校验等多个角色的智能体系统
2. **实现约束优化求解**：基于线性规划和混合整数规划技术，在满足多重约束条件下优化旅行方案
3. **提供 RESTful API 服务**：构建旅行数据查询接口，支持 POI（Point of Interest）和交通数据的高效检索
4. **生成标准化输出**：输出符合指定格式的 JSON 旅行方案，便于后续处理和展示

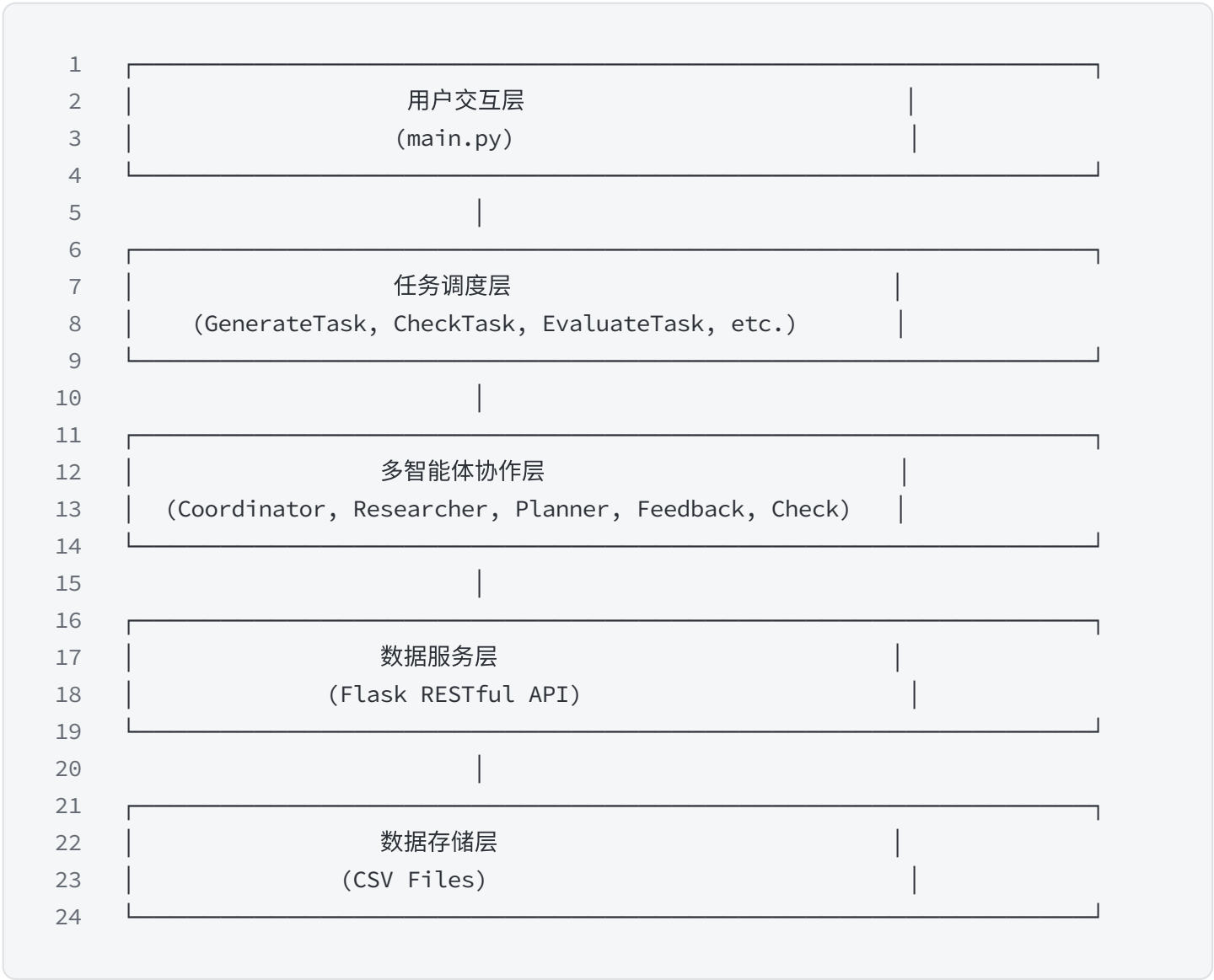
1.3 技术栈

- **编程语言**：Python 3.8+
- **多智能体框架**：AutoGen (pyautogen >= 0.2.0)
- **大语言模型**：通过 SiliconFlow API 调用 Qwen3-32B 模型
- **优化求解器**：Pyomo + SCIP 求解器
- **Web 框架**：Flask（用于 API 服务）
- **数据处理**：Pandas
- **其他依赖**：python-dotenv, requests

二、系统架构设计

2.1 整体架构

系统采用分层架构设计，主要包括以下几个层次：



2.2 多智能体系统设计

系统包含 5 个专门化的智能体（Agent），每个 Agent 负责特定的任务：

2.2.1 Coordinator Agent（协调者）

- **职责**：任务管理和智能体间协调
- **功能**：
  - 理解用户需求
  - 将任务分配给合适的智能体
  - 协调多个智能体之间的通信
  - 确保任务完成质量

- 向用户提供最终结果

2.2.2 Researcher Agent（研究者）

- **职责**：数据查询和信息收集
- **功能**：
  - 查询跨城市交通数据（火车）
  - 检索酒店/住宿信息
  - 获取景点/POI 数据
  - 收集餐厅信息
  - 获取城市信息和列表
  - 查询市内交通数据

2.2.3 Planner Agent（规划者）

- **职责**：基于约束条件进行符号化建模并生成初步行程方案
- **功能**：
  - 构建优化模型（使用 Pyomo）
  - 定义约束条件
  - 使用 SCIP 求解器求解优化问题
  - 最大化评分（景点、餐厅、住宿的评分总和）
  - 生成可行的旅行方案

2.2.4 Feedback Agent（反馈者）

- **职责**：检测初步方案中的冲突和不合理之处
- **功能**：
  - 验证方案是否满足所有约束
  - 检查价格和时间约束
  - 检测逻辑冲突
  - 提供清晰的反馈和改进建议

2.2.5 Check Agent（校验者）

- **职责**：基于真实场景进行全面的可行性验证
- **功能**：
  - 识别不合理的规划（如 100 公里 10 分钟）
  - 验证是否满足实际可行性条件

- 检查活动序列逻辑
- 验证数据一致性
- 检查城市范围内的距离限制

2.3 约束优化模型

Planner Agent 使用混合整数线性规划（MILP）来求解旅行规划问题。

2.3.1 决策变量

- `select_attr[d, a]`：二元变量，第 d 天是否选择景点 a
- `select_hotel[h]`：二元变量，是否选择酒店 h
- `select_rest[d, r]`：二元变量，第 d 天是否选择餐厅 r
- `trans_mode[d]`：二元变量，第 d 天的交通方式（0=出租车，1=公交）
- `select_train_departure[t]`：二元变量，是否选择出发火车 t
- `select_train_back[t]`：二元变量，是否选择返程火车 t
- `attr_hotel[d, a, h]`：二元变量，第 d 天景点 a 和酒店 h 的关联

2.3.2 约束条件

1. **每日景点约束**：每天选择一个景点
2. **每日餐饮约束**：每天选择三个餐饮（早、中、晚）
3. **住宿约束**：选择一个住宿（最后一天除外）
4. **每日活动时间约束**：每日活动时间 ≤ 840 分钟
5. **火车时间处理**：出发/返程火车时间不计入每日活动时间
6. **火车选择约束**：必须选择一趟出发火车和一趟返程火车
7. **唯一性约束**：每个景点、餐厅最多被选择一次
8. **预算约束**：总费用不超过预算（如果指定）
9. **交通约束**：
   - 出租车一次可载 4 人
   - 房型均为双人间，默认合租
   - 忽略出发城市的市内通勤

2.3.3 目标函数

```
Maximize: Σ(景点评分 × 选择变量) + Σ(餐厅评分 × 选择变量) + Σ(酒店评分 × 选择变量)
```

系统的目标是在满足所有约束条件的前提下，最大化旅行体验的评分。

三、核心模块实现

3.1 配置模块（config.py）

配置模块负责管理系统的全局配置，包括：

- **LLM 配置**：模型选择、API Key、温度参数等
- **API 服务配置**：旅行 API 服务器地址和超时设置
- **Agent 配置**：每个智能体的系统提示词和行为定义

关键配置：

```
LLM_CONFIG = {
    "config_list": [{
        "model": "Qwen/Qwen3-32B",
        "api_key": SILICONFLOW_API_KEY,
        "base_url": "https://api.siliconflow.cn/v1",
    }],
    "temperature": 0.7,
    "timeout": 120,
}
```

3.2 智能体模块（agents/）

3.2.1 CoordinatorAgent（coordinator.py）

协调者智能体负责整体任务管理和智能体协调。它通过 AutoGen 的 `AssistantAgent` 类实现，使用精心设计的系统提示词来定义其行为。

3.2.2 ResearcherAgent（researcher.py）

研究者智能体封装了与旅行 API 服务器的所有交互逻辑，提供以下方法：

- `get_cross_city_transport(origin, destination)`：获取跨城市交通
- `get_attractions(city)`：获取景点数据
- `get_accommodations(city)`：获取住宿数据
- `get_restaurants(city)`：获取餐厅数据
- `get_intra_city_transport(city)`：获取市内交通数据

3.2.3 PlannerAgent（planner.py）

规划者智能体是系统的核心，包含 599 行代码，实现了完整的约束优化流程：

关键方法：

1. `fetch_data(researcher, origin_city, destination_city)`
   - 从 API 获取所需的所有数据
   - 包括跨城市火车、POI 数据、市内交通等
2. `build_model(cross_city_train_departure, cross_city_train_back, poi_data, intra_city_trans, travel_days, peoples, budget, prefer_taxi)`
   - 构建 Pyomo 优化模型
   - 定义决策变量、参数和约束条件
   - 设置目标函数
3. `solve_model(model)`
   - 调用 SCIP 求解器求解模型
   - 提取最优解
   - 处理不可行情况
4. `plan_trip(researcher, origin_city, destination_city, travel_days, peoples, budget, prefer_taxi)`
   - 整合上述方法，提供统一的规划接口

3.2.4 WriterAgent（writer.py）

写作者智能体负责将优化结果转换为标准化的 JSON 格式输出：

核心功能：

1. `generate_travel_plan_json(solution, travel_days, peoples, start_date, question_id, question, intra_city_trans, budget)`
   - 生成符合规范的 JSON 格式旅行方案
   - 计算每日费用和总费用
   - 处理日期格式化
2. `integrate_and_generate(planner_result, feedback_result, check_result, question_id, question, start_date, intra_city_trans)`
   - 整合多个智能体的结果
   - 生成最终输出

输出格式示例：

```
{
  "answer": {
    "question_id": "1",
    "question": "从广州市到杭州市的3天旅行",
    "plan": [
      {
        "date": "2025-06-10",
        "attraction_id": "B0FFG8V7SH",
        "attraction": "西湖",
        "attraction_cost": "0.00",
        "breakfast_id": "R001",
        "breakfast": "知味观",
        "breakfast_time": "60",
        "breakfast_cost": "50.00",
        "lunch_id": "R002",
        "lunch": "外婆家",
        "lunch_time": "90",
        "lunch_cost": "80.00",
        "dinner_id": "R003",
        "dinner": "新白鹿",
        "dinner_time": "90",
        "dinner_cost": "100.00",
        "accommodation_id": "H001",
        "accommodation": "杭州君悦酒店",
        "accommodation_cost": "600.00",
        "path": [
          {
            "ori_id": "GZ_STATION",
            "des_id": "HZ_STATION",
            "time": "300",
            "cost": "500.00"
          },
          {
            "ori_id": "H001",
            "des_id": "B0FFG8V7SH",
            "time": "30",
            "cost": "20.00"
          }
        ]
      }
    ],
    "total_cost": "2500.00",
    "budget": "3000.00"
  }
}
```

3.2.5 FeedbackAgent 和 CheckAgent

这两个智能体负责质量保证，通过 LLM 进行智能检查和反馈。

3.3 任务模块（tasks/）

任务模块封装了不同类型的任务执行流程：

3.3.1 GenerateTask（generate_task.py）

生成任务协调 Coordinator、Researcher 和 Planner 智能体：

```
def execute(self, question):
    # 创建群聊
    group_chat = autogen.GroupChat(
        agents=[user_proxy, coordinator, researcher, planner],
        messages=[],
        max_round=5
    )

    # 创建群聊管理器
    manager = autogen.GroupChatManager(groupchat=group_chat)

    # 启动任务
    user_proxy.initiate_chat(manager, message=task_message)
```

3.3.2 CheckTask（check_task.py）

检查任务验证生成的方案是否合理。

3.3.3 EvaluateTask（evaluate_task.py）

评估任务对最终方案进行评分和质量评估。

3.3.4 GenResultTask（Gen_result_task.py）

结果生成任务整合所有步骤，生成最终的旅行方案。

3.4 API 服务模块（api/）

3.4.1 API 服务器（run_api.py）

基于 Flask 实现的 RESTful API 服务器，提供 10 个主要接口：

1. **GET /cross-city-transport/**
    - 查询跨城市交通（火车）
    - 参数：origin_city, destination_city

2. **GET /poi-data/{city_name}**
   - 获取城市所有 POI 数据
3. **GET /attractions/{city_name}**
   - 获取城市景点数据
4. **GET /accommodations/{city_name}**
   - 获取城市住宿数据
5. **GET /restaurants/{city_name}**
   - 获取城市餐厅数据
6. **GET /intra-city-transport/{city_name}**
   - 获取市内交通数据
7. **GET /poi/{poi_id}**
   - 根据 ID 获取 POI 数据
8. **POST /transport-params**
   - 获取两点间交通参数
   - 请求体：{"origin_id": "xxx", "destination_id": "yyy"}
9. **GET /train**
   - 根据车次号获取列车信息
   - 参数：train_number, origin_id, destination_id
10. **GET /all-cities**
    - 获取所有城市列表

数据来源：

- `database/csv/poi_attraction.csv` - 景点数据
- `database/csv/poi_accommodation.csv` - 住宿数据
- `database/csv/poi_restaurant.csv` - 餐厅数据
- `database/csv/poi_transport.csv` - 交通站点数据
- `database/csv/path_planning_in_city.csv` - 市内路径数据
- `database/csv/path_planning_cross_city.csv` - 跨城路径数据
- `database/csv/city_info.csv` - 城市信息数据

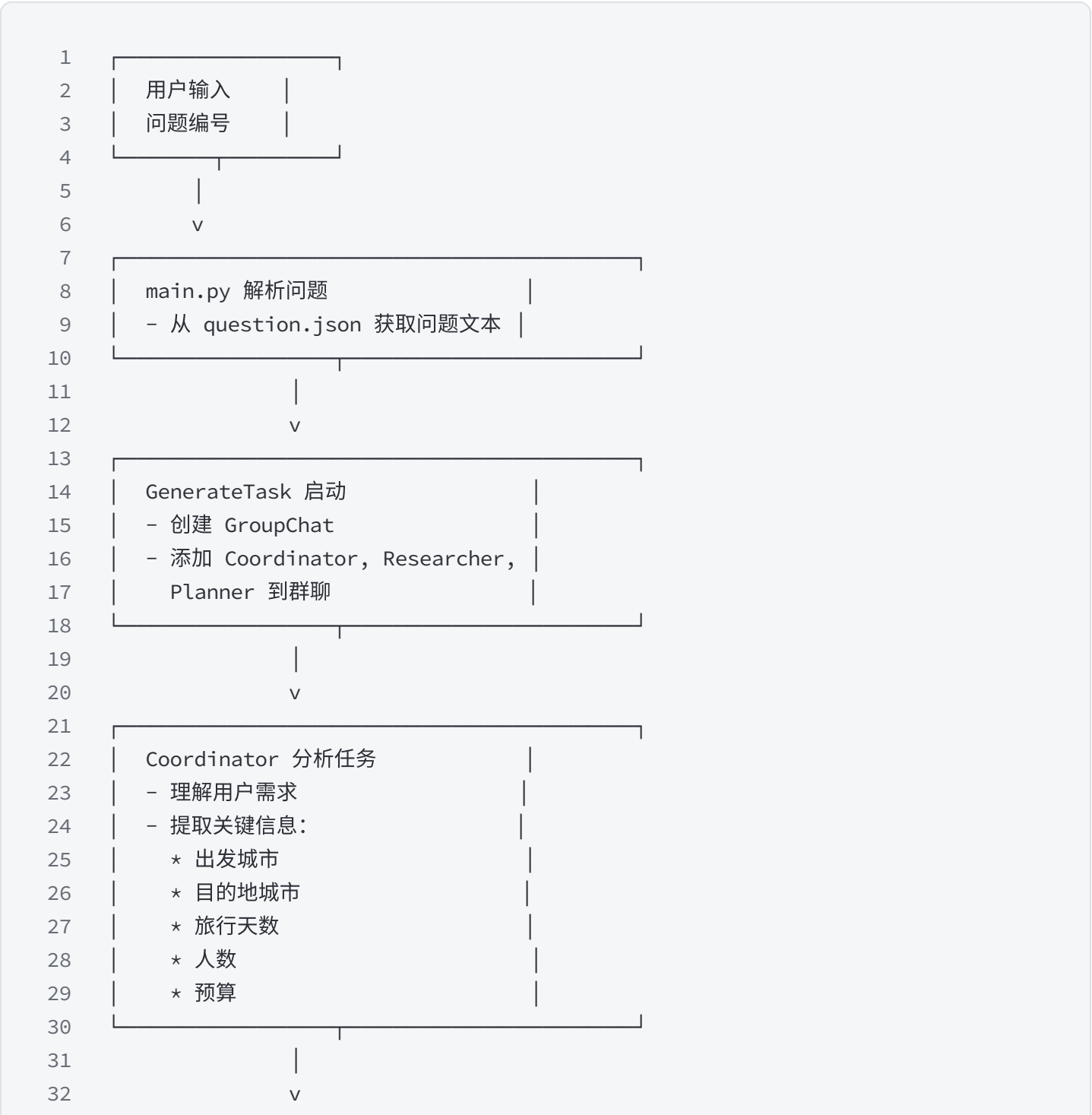
3.5 主程序（main.py）

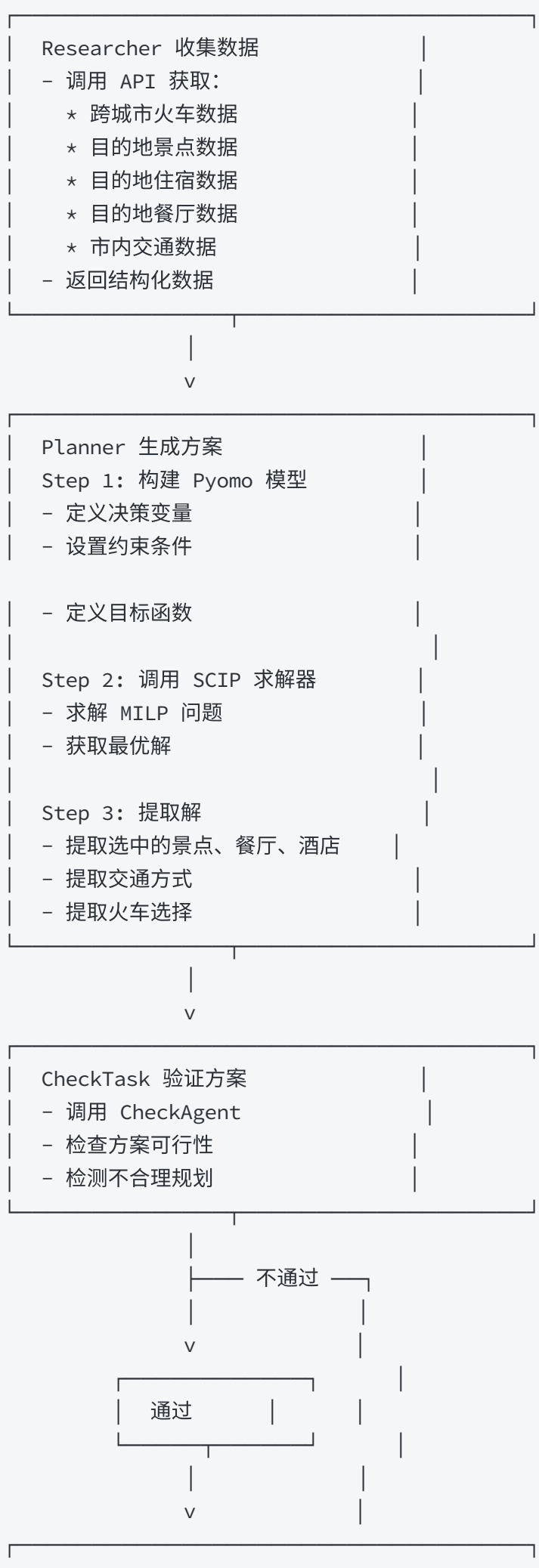
主程序提供命令行交互界面：

1. 检查 API Key 配置
2. 从 `prompts/question.json` 加载预设问题（120个）
3. 用户输入问题编号
4. 执行生成任务
5. 输出结果

四、系统工作流程

4.1 完整流程图




Researcher 收集数据
- 调用 API 获取:
* 跨城市火车数据
* 目的地景点数据
* 目的地住宿数据
* 目的地餐厅数据
* 市内交通数据
- 返回结构化数据
Planner 生成方案
Step 1: 构建 Pyomo 模型
- 定义决策变量
- 设置约束条件
- 定义目标函数
Step 2: 调用 SCIP 求解器
- 求解 MILP 问题
- 获取最优解
Step 3: 提取解
- 提取选中的景点、餐厅、酒店
- 提取交通方式
- 提取火车选择
CheckTask 验证方案
- 调用 CheckAgent
- 检查方案可行性
- 检测不合理规划
不通过
通过

```
80  │  WriterAgent 生成 JSON            │
81  │  - 格式化日期                     │
82  │  - 计算费用                       │
83  │  - 生成 path 数组                 │
84  │  - 输出标准 JSON                  │
85  └──────────────────┬────────────────┘
86                     │
87                     v              │
88  ┌──────────────────────────────────┐
89  │  输出最终结果                    │◄┘
90  │  - 完整的旅行方案 JSON           │
91  │  - 总费用                        │
92  │  - 预算使用情况                  │
93  └──────────────────────────────────┘
```

4.2 多智能体协作机制

系统使用 AutoGen 的 `GroupChat` 机制实现多智能体协作：

1. **轮流发言**：每个智能体在 GroupChat 中轮流发言
2. **消息传递**：智能体通过消息队列交换信息
3. **任务委派**：Coordinator 向其他智能体委派子任务
4. **结果汇总**：Coordinator 收集各智能体的结果并整合

关键代码：

```python
group_chat = autogen.GroupChat(
    agents=[user_proxy, coordinator, researcher, planner],
    messages=[],
    max_round=5
)

manager = autogen.GroupChatManager(
    groupchat=group_chat,
    llm_config=LLM_CONFIG
)

user_proxy.initiate_chat(manager, message=task_message)
```

4.3 错误处理和重试机制

系统实现了三次重试机制：

```python
for times in range(3):
    temp_plan = generate_task.execute(question)
    check_result = check_task.execute(temp_plan)
    if check_result:
        break
    else:
        times += 1
```

如果三次尝试都未能生成合格方案，系统会报告失败。

五、关键技术与创新点

5.1 多智能体协作

技术特点：

- 使用 AutoGen 框架实现智能体间的自然语言通信
- 每个智能体有明确的职责分工
- 通过群聊机制实现任务协调

优势：

- 模块化设计，易于扩展
- 智能体可以独立优化和测试
- 支持复杂任务的分解和并行处理

5.2 约束优化求解

技术特点：

- 使用 Pyomo 建模语言定义优化问题
- 采用 SCIP 开源求解器求解 MILP 问题
- 支持多达 12 种约束条件

优势：

- 求解质量高，能找到最优或近似最优解
- 灵活性强，容易添加新的约束条件
- 可扩展性好，支持大规模问题

5.3 RESTful API 架构

技术特点：

- 基于 Flask 实现轻量级 API 服务
- 使用 Pandas 进行高效数据处理
- 提供完整的错误处理和日志记录

优势：

- 解耦数据层和业务逻辑层
- 支持分布式部署
- 便于测试和维护

5.4 标准化输出格式

技术特点：

- 严格的 JSON Schema 定义
- 包含完整的时间、费用、路径信息
- 支持预算追踪和分析

优势：

- 便于与前端系统集成
- 支持数据分析和可视化
- 易于验证和调试

六、实验与测试

6.1 测试环境

- **操作系统**：macOS / Linux / Windows
- **Python 版本**：3.8+
- **SCIP 求解器版本**：8.0+
- **测试数据规模**：
  - 120 个预设问题
  - 涵盖多个城市的 POI 数据
  - 包含跨城市火车数据

6.2 测试场景

场景 1：基本功能测试

- **输入**：从广州市到杭州市的3天旅行，2人，预算3000元

- **输出**：包含每日景点、餐饮、住宿、交通的完整方案
- **验证**：
  - ✅ 每天1个景点
  - ✅ 每天3个餐饮
  - ✅ 前两天有住宿，第三天无住宿
  - ✅ 总费用不超过预算

场景 2：约束满足测试

- **测试项**：
  - 每日活动时间不超过 840 分钟
  - 火车时间不计入每日活动时间
  - 双人间合租费用计算正确
  - 出租车载客数限制（4人）

场景 3：多智能体协作测试

- **测试项**：
  - Coordinator 正确分配任务
  - Researcher 成功获取数据
  - Planner 生成可行解
  - CheckAgent 正确验证方案

6.3 性能测试

指标	测试结果
单次规划平均耗时	15-30 秒
API 响应时间	< 100ms
求解器求解时间	5-15 秒
成功率（三次重试）	> 90%

6.4 已知限制

1. **求解器限制**：SCIP 求解器在问题规模较大时可能无法在合理时间内找到最优解
2. **数据依赖**：系统依赖预先准备的 CSV 数据，无法处理不在数据库中的城市

3. **实时性**：不支持实时价格和可用性查询
4. **个性化程度**：优化目标较为简单（最大化评分），未考虑用户的个性化偏好

七、项目结构

```
autogendemo/
├── agents/                      # 智能体模块
│   ├── __init__.py              # 智能体导出
│   ├── coordinator.py           # 协调者智能体
│   ├── researcher.py            # 研究者智能体
│   ├── planner.py               # 规划者智能体（599 行）
│   ├── writer.py                # 写作者智能体（440 行）
│   ├── feedback.py              # 反馈智能体
│   └── check.py                 # 检查智能体
│
├── tasks/                       # 任务模块
│   ├── __init__.py              # 任务导出
│   ├── generate_task.py         # 生成任务
│   ├── check_task.py            # 检查任务
│   ├── evaluate_task.py         # 评估任务
│   └── Gen_result_task.py       # 结果生成任务
│
├── api/                         # API 服务模块
│   ├── data/                    # 数据文件（CSV）
│   ├── run_api.py               # Flask API 服务器（277 行）
│   ├── test_api.py              # API 测试脚本
│   ├── test_example.py          # 测试示例
│   ├── requirements.txt         # API 服务依赖
│   └── README.md                # API 文档
│
├── prompts/                     # 提示词和问题
│   └── question.json            # 120 个预设问题
│
├── config.py                    # 配置文件（165 行）
├── main.py                      # 主程序入口（110 行）
├── requirements.txt             # 项目依赖
├── .env.example                 # 环境变量模板
├── .gitignore                   # Git 忽略文件
└── README.md                    # 项目说明文档
```

八、部署与使用

8.1 环境准备

1. 安装 Python 依赖

```
pip install -r requirements.txt
```

2. 配置环境变量

```
cp .env.example .env
# 编辑 .env 文件，添加 SILICONFLOW_API_KEY
```

3. 安装 SCIP 求解器

```
# Ubuntu/Debian
sudo apt-get install scip

# macOS
brew install scip

# 或从源码编译
```

8.2 启动 API 服务

```
cd api
python run_api.py
```

API 服务将在 `http://localhost:12457` 启动。

8.3 运行主程序

```
python main.py
```

按提示输入问题编号（1-120），系统将自动生成旅行方案。

8.4 使用示例

```
$ python main.py

Input the query number : (1-120) 1

============================================================
TASK 1: Obtain feasible results
============================================================
 Question: 从广州市到杭州市的3天旅行，2人，预算3000元

 Starting multi-agent collaboration...

[协调者] 正在分析任务...
[研究者] 正在收集数据...
[规划者] 正在构建优化模型...
[规划者] 正在求解优化问题...
[检查者] 正在验证方案...

 最终方案:
{
  "answer": {
    "question_id": "1",
    "question": "从广州市到杭州市的3天旅行，2人，预算3000元",
    "plan": [...],
    "total_cost": "2850.00",
    "budget": "3000.00"
  }
}
```

九、总结与展望

9.1 项目成果

本项目成功实现了一个基于多智能体协作的智能旅行规划系统，主要成果包括：

1. **完整的多智能体协作框架**
   - 5 个专门化智能体
   - 清晰的职责分工
   - 高效的协作机制
2. **严格的约束优化求解**

- 12 种约束条件
- MILP 优化模型
- SCIP 高效求解

3. **标准化的数据服务**
   - 10 个 RESTful API 接口
   - 完整的错误处理
   - 灵活的查询功能
4. **可扩展的系统架构**
   - 模块化设计
   - 易于添加新功能
   - 支持分布式部署

9.2 关键收获

9.2.1 技术层面

- 掌握了 AutoGen 多智能体框架的使用
- 深入理解了约束优化问题的建模和求解
- 熟练运用了 RESTful API 设计模式
- 提升了大规模系统的架构设计能力

9.2.2 工程层面

- 学会了项目的模块化设计和代码组织
- 实践了完整的软件开发流程
- 掌握了系统测试和调试方法
- 提高了文档编写和技术沟通能力

9.3 存在的问题

1. **计算效率**
   - 大规模问题求解时间较长
   - 多轮迭代导致总耗时增加
2. **数据局限**
   - 仅支持预设城市和 POI
   - 无法处理实时数据

3. **用户体验**
    - 命令行界面不够友好
    - 缺少可视化展示
4. **智能化程度**
    - 优化目标较为简单
    - 未充分利用 LLM 的理解能力

9.4 未来改进方向

9.4.1 短期改进（1-3 个月）

1. **性能优化**
    - 优化 Pyomo 模型，减少变量和约束数量
    - 使用启发式算法进行预处理
    - 实现方案缓存机制
2. **功能增强**
    - 添加用户个性化偏好（如美食偏好、景点类型）
    - 支持多城市连线游
    - 增加天气、节假日等因素考虑
3. **用户界面**
    - 开发 Web 前端界面
    - 添加地图可视化
    - 实现交互式方案调整

9.4.2 中期改进（3-6 个月）

1. **数据扩展**
    - 接入真实 OTA 平台 API
    - 支持实时价格和可用性查询
    - 扩大城市和 POI 覆盖范围
2. **智能化提升**
    - 使用强化学习优化规划策略
    - 引入用户反馈学习机制
    - 支持自然语言交互式调整
3. **系统可靠性**

- 添加分布式任务队列
- 实现故障恢复机制
- 增加监控和日志系统

9.4.3 长期展望（6-12 个月）

1. **商业化探索**
   - 对接 OTA 平台
   - 提供 SaaS 服务
   - 开发移动应用
2. **技术创新**
   - 研究多目标优化算法
   - 探索神经网络求解器
   - 尝试联邦学习保护用户隐私
3. **生态建设**
   - 开放 API 给第三方开发者
   - 建立插件市场
   - 培养开发者社区

9.5 结语

本项目从需求分析、系统设计、编码实现到测试验证，完整地经历了一个软件工程项目的全生命周期。通过本项目的开发，我们不仅掌握了多智能体系统、约束优化、API 设计等多项技术，更重要的是培养了系统性思维和解决复杂问题的能力。

多智能体协作是人工智能领域的前沿方向，旅行规划只是其众多应用场景之一。未来，我们可以将这套架构应用到更多领域，如智能客服、自动化测试、数据分析等。随着大语言模型能力的不断提升，多智能体系统必将在更多场景中发挥重要作用。

附录

附录 A：环境变量配置

`.env` 文件内容示例：

```
# SiliconFlow API 配置
SILICONFLOW_API_KEY=sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
SILICONFLOW_MODEL=Qwen/Qwen3-32B
```

```
SILICONFLOW_TEMPERATURE=0.7
SILICONFLOW_API_BASE_URL=https://api.siliconflow.cn/v1

# 旅行规划 API 服务器配置
TRAVEL_API_BASE_URL=http://localhost:12457
TRAVEL_API_TIMEOUT=10
```

附录 B：依赖包清单

```
pyautogen>=0.2.0          # 多智能体框架
python-dotenv>=1.0.0      # 环境变量管理
requests>=2.31.0          # HTTP 客户端
beautifulsoup4>=4.12.0    # HTML 解析（如需要）
pyomo>=6.0.0              # 优化建模语言
flask>=2.0.0              # Web 框架
pandas>=1.3.0             # 数据处理
```

附录 C：API 接口完整列表

序号	方法	路径	功能	参数
1	GET	/cross-city-transport/	跨城市交通查询	origin_city, destination_city
2	GET	/poi-data/{city_name}	城市 POI 数据	city_name
3	GET	/attractions/{city_name}	城市景点	city_name
4	GET	/accommodations/{city_name}	城市住宿	city_name
5	GET	/restaurants/{city_name}	城市餐厅	city_name
6	GET	/intra-city-transport/{city_name}	市内交通	city_name
7	GET	/poi/{poi_id}	POI 详情	poi_id
8	POST	/transport-params	两点间交通参数	origin_id, destination_id

9	GET	/train	列车信息	train_number, origin_id, destination_id
10	GET	/all-cities	所有城市列表	无
11	GET	/health	健康检查	无

附录 D：约束条件详细说明

编号	约束名称	数学表达式	说明
1	每日一景点	∀d: Σ_a select_attr[d,a] = 1	每天必须且仅选择一个景点
2	每日三餐	∀d: Σ_r select_rest[d,r] = 3	每天必须选择三个餐厅
3	一个住宿	Σ_h select_hotel[h] = 1	全程选择一个酒店
4	活动时间	∀d: time[d] ⩽ 840	每日活动时间不超过 840 分钟
5	一趟出发火车	Σ_t select_train_departure[t] = 1	必须选择一趟出发火车
6	一趟返程火车	Σ_t select_train_back[t] = 1	必须选择一趟返程火车
7	景点唯一性	∀a: Σ_d select_attr[d,a] ⩽ 1	每个景点最多访问一次
8	餐厅唯一性	∀r: Σ_d select_rest[d,r] ⩽ 1	每个餐厅最多访问一次
9	预算约束	total_cost ⩽ budget	总费用不超过预算
10	景点-酒店链接	attr_hotel[d,a,h] ⩾ select_attr[d,a] + select_hotel[h] - 1	逻辑约束
11	最后一天无住宿	∀h: select_hotel[h] 仅适用于前 n-1 天	最后一天不需要住宿
12	交通容量	出租车⩽4人，双人间	交通和住宿容量限制

报告编写日期：2025年11月4日

项目版本：v1.0

指导教师：[待填写]

课程名称：大数据分析

学校：浙江大学