急先锋搜索

1 设计理念

急先锋搜索是一个AI驱动的搜索引擎，基于MoFA开发框架构建，旨在为用户提供快速、精准的热点话题信息整合服务。项目的核心目标是通过多智能体协作，整合官方渠道和社交媒体上的实时信息，帮助用户在紧急情况下快速获取权威信息和有效建议。

1.1 核心特点

1. **多智能体协作**：
   - 系统由多个智能体组成，包括帮助智能体、搜索智能体和思考智能体，各司其职，协同工作。
   - 帮助智能体提供初步回答，搜索智能体从多平台获取实时数据，思考智能体整合信息并生成最终分析。

2. **实时性与准确性**：
   - 通过集成多平台数据源（如必应搜索、微博、今日头条等），确保信息的实时性。
   - 利用大模型的强大分析能力，过滤冗余信息，提供高质量的答案。

3. **用户友好性**：
   - 终端输入作为用户交互入口，简化了操作流程。
   - 系统设计注重用户体验，提供清晰的查询结果和直观的交互方式。

4. **可扩展性**：
   - 基于MoFA框架的模块化设计，便于功能扩展和智能体的替换。
   - 支持多种大模型（如OpenAI、星火大模型等），可根据需求灵活调整。

1.2 应用场景

- **突发事件应急**：快速获取气象灾害、应急警报等相关信息，为用户提供权威建议。
- **热点话题追踪**：整合社交媒体和官方渠道的信息，帮助用户全面了解事件动态。
- **信息整合与分析**：通过多平台数据源的整合，生成高质量的分析报告，适用于新闻、研究等领域。

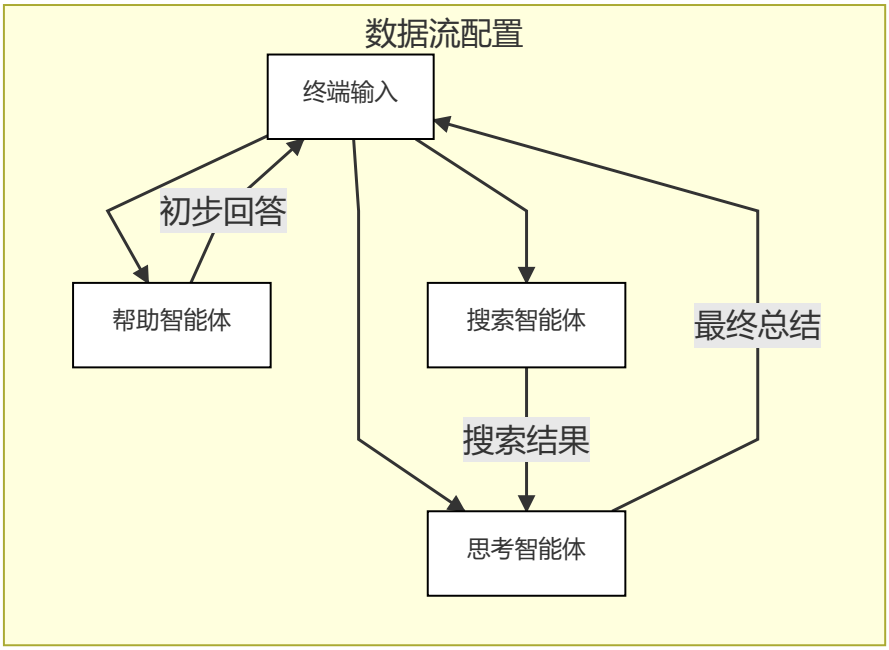
1.3 设计目标

- **高效**：通过多智能体并行处理，提升信息获取与分析的效率。
- **可靠**：确保数据来源的权威性与结果的准确性。
- **易用**：提供简单直观的交互方式，降低用户使用门槛。

2 技术架构图

2.1 MoFA 框架图

2.2 Agent 框架



3 运行指南

3.1 1 Rust 环境

```
# 安装 Rust
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

# 安装 Dora 运行时
cargo install dora-cli

# 验证安装
rustc --version
cargo --version
dora --version
```

3.2 2 Mofa 框架环境

```
# 安装 UV 包管理器
pip install uv
# 在合适的位置克隆仓库
git clone https://github.com/moxin-org/mofa.git
cd mofa/python
# 安装依赖
uv pip install -e .
pip install -e .
```

3.3 注意:

- 本地python环境要纯净，不要多个python版本，否则容易导致Dora-rs运行环境和Mofa框架的冲突。
- 如果你的环境使用的是Anaconda / Miniconda，务必将Mofa安装到`Base`环境下，以保证Dora运行环境和Mofa环境的一致。
- 要求python环境 >= 3.10。
-

3.4 3 安装 CrisisPioneerSearch

```
# 在合适的位置克隆仓库
git clone https://github.com/lovemaoli/CrisisPioneerSearch.git
cd ./CrisisPioneerSearch

# 安装依赖
uv pip install -r requirements.txt
```

3.5 4 配置环境变量

创建 `.env.secret` 文件(crisis_pioneer_search.yml目录同级进行创建):

```
LLM_API_KEY=your_api_key_here
LLM_API_BASE=https://api.openai.com/v1  # 或其他API地址
LLM_MODEL=gpt-3.5-turbo  # 或其他模型名称
```

注: 写者使用的是星火大模型

3.6 5 启动数据流

```
cd  ./crisis_pioneer_search

# 启动 Dora 服务
dora up

# 构建并运行数据流
dora build .\examples\crisis_pioneer_search\crisis_pioneer_search.yml
dora start .\examples\crisis_pioneer_search\crisis_pioneer_search.yml
```

3.7 6 测试交互

```
# 在另一个终端运行输入节点
terminal-input

# 输入测试数据
> hello
```

3.8 7 运行效果

```
root@root hello_world % terminal-input

 Send You Task :  你好
-------------llm_result---------------
……
```

4 架构设计

4.1 Agent 处理流程设计

1. 用户通过 terminal-input 输入应急事件查询
2. 查询分别发送给帮助智能体和搜索智能体
3. 帮助智能体提供初步回答，基于大模型已有知识
4. 搜索智能体在多个渠道检索最新信息
5. 思考智能体整合搜索结果和初步回答，生成最终分析报告
6. 最终分析报告返回给terminal-input显示给用户

4.2 数据流配置解释

4.2.1 关于 `crisis_pioneer_search.yml`：

1. terminal-input：

- 作为用户交互入口
- 接收来自三个智能体的输出：初步回答(first_answer)、相关网站列表(relevent_site)和最终分析结果(agent_response)
- 输出用户查询(data)到其他智能体

2. crisis-help-agent：

- 帮助智能体，提供快速初步回答
- 接收用户查询(terminal-input/data)
- 生成初步分析结果(llm_result)

3. crisis-search-agent：

- 搜索智能体，负责从多个平台获取信息
- 接收用户查询(terminal-input/data)
- 输出两种结果：
  (1) bing_result：从必应搜索获取的数据
  (2) site_result：从其他相关网站(如微博、今日头条)收集的信息

4. crisis-think-agent：

- 思考智能体，整合搜索结果和初步回答
- 接收用户查询(terminal-input/data)和必应搜索结果(crisis-search-agent/bing_result)
- 生成最终综合分析(final_result)
- IS_DATAFLOW_END: true 表示此节点执行完毕后数据流结束，准备接收新的用户输入

4.3 关键代码说明

1. **使用装饰器**
   - 使用 `@run_agent` 装饰器简化代码结构
   - 自动处理循环和异常
2. **简单的输入输出**
   - 接收单个输入参数 `query`
   - 返回单个输出结果 `llm_result`
3. **错误处理**
   - 使用 try-except 捕获异常
   - 记录错误日志
   - 返回错误信息给用户

4.4 如何自定义

1. **修改系统提示词**

```
messages=[
    {"role": "system", "content": "你的自定义系统提示词"},
    {"role": "user", "content": user_input}
]
```

2. **更换LLM提供商**
   - 修改 `.env.secret` 中的 API 配置
   - 根据需要调整模型参数

4.5 注意事项

1. 确保 `.env.secret` 已添加到 `.gitignore`
2. API密钥要妥善保管

4.6 日志文件位置

- logs/crisis-xxx-agent.txt: 主智能体运行日志
- logs/dora-coordinator.txt: 协调器日志
- logs/dora-daemon.txt: 守护进程日志

4.7 项目结构

```
crisis_pioneer_search/
├── agent-hub/
│   ├── crisis-search-agent/
│   │   ├── configs/
│   │   │   └── agent.yml          # 配置文件
│   │   ├── main.py                # 主程序
```

```
│   │   └── __init__.py
│   ├── search-engine-agent/
│   │   ├── configs/
│   │   │   └── agent.yml          # 配置文件
│   │   ├── main.py                # 主程序
│   │   └── __init__.py
│   └── thinking-agent/
│       ├── configs/
│       │   └── agent.yml          # 配置文件
│       ├── main.py                # 主程序
│       └── __init__.py
├── node-hub/
│    └── terminal-input/
│        └── terminal_input/
│           └── main.py            # 主程序
├── examples/
│   ├── .env.secret                # API密钥配置
│   └── crisis_pioneer_search.yml  # 数据流配置
└── README.md                      # 项目文档
```