

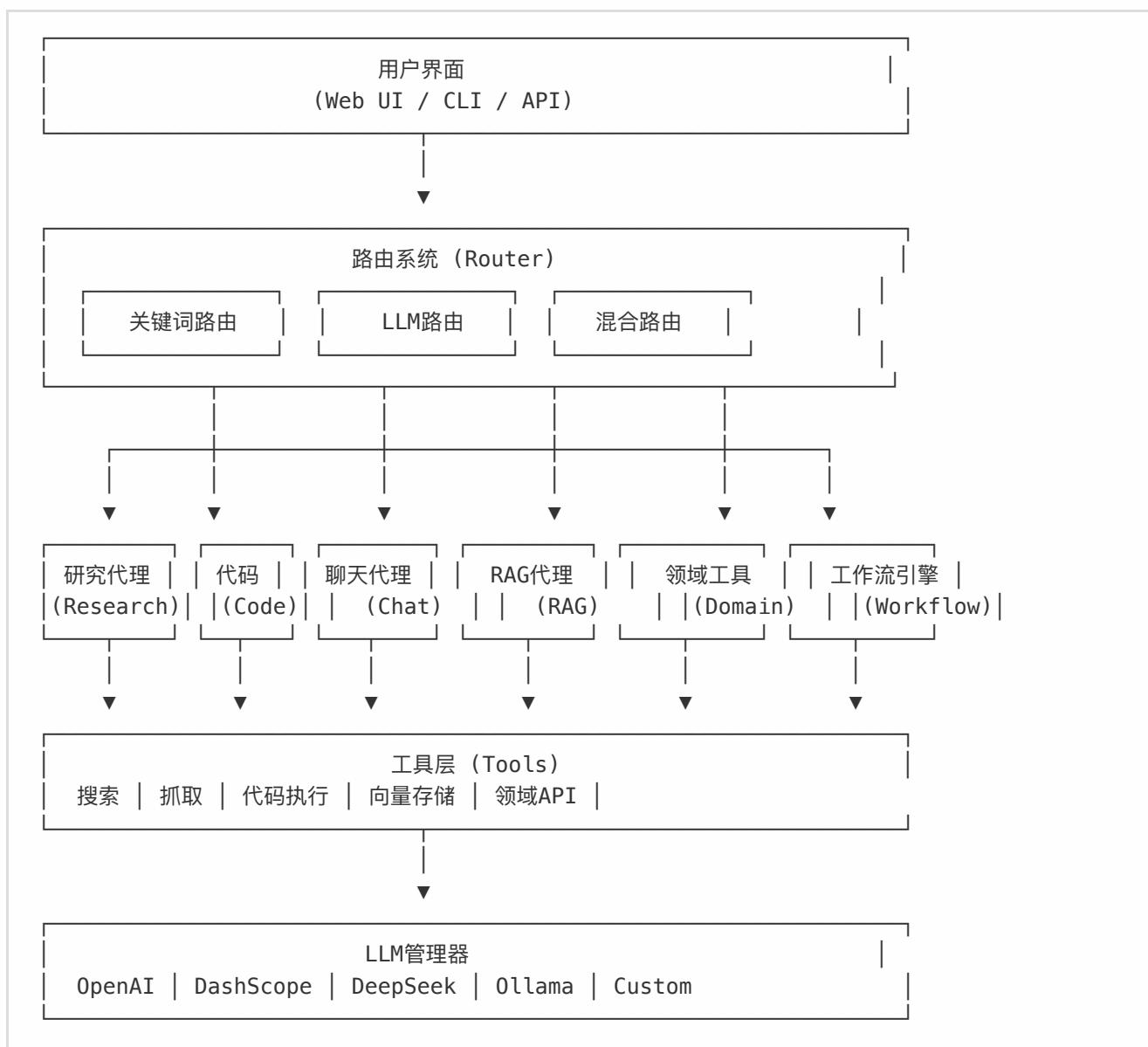
第一部分：团队概览

- 团队名称:
- 团队成员:

第二部分：系统设计与架构

本部分阐述了“系统设计概览”。

高层架构图:



技术栈:

- 后端框架: FastAPI
- 数据库 / 向量存储:
 - SQLite (用于对话历史记录)
 - ChromaDB (用于RAG向量存储)
- 核心大语言模型 / API:
 - OpenAI (GPT 系列)
 - Anthropic (Claude 系列)
 - Google (Gemini 系列)
 - Ollama (用于本地模型如 Llama)
 - 其他兼容OpenAI的API (例如: 阿里云通义千问, DeepSeek)
- 关键库:
 - sentence-transformers (用于文本嵌入)
 - transformers (用于本地重排模型)
 - paddleocr / paddlepaddle (用于OCR)
 - beautifulsoup4 / trafileatura (用于网页抓取)
 - pyowm, yfinance, openrouteservice (用于特定领域工具)

数据流:

一个典型查询的数据流如下: 用户查询通过FastAPI的Web界面进入系统。首先, 查询被一个混合路由器 (HybridRouter) 处理, 该路由器结合了快速的关键词匹配和强大的基于LLM的分类, 以确定用户的意图 (例如, 研究、代码、RAG或特定领域查询如天气)。根据这个决策, 查询被分派给相应的代理(Agent) (例如 ResearchAgent或RAGAgent)。然后, 代理使用一套工具(Tools)来执行任务——这可能涉及调用外部API (如网页搜索或金融数据)、在安全沙箱中执行代码, 或查询ChromaDB向量存储。工具返回的结果由LLM管理器(LLMManager)进行综合处理, 该管理器支持多个提供商并具备回退功能。最终, 格式化的响应被发送回用户, 并且该次交互被记录在SQLite历史数据库中。

第三部分: 当前进展与功能实现

主工作流实现

- 状态: 已实现并已测试
- 实现细节: 项目中包含一个工作流引擎 (`src/workflow/`), 用于编排多步骤任务。架构图表明它可能使用有向无环图 (DAG) 模型。这使得系统可以执行复杂的操作序列, 例如一个结构化的研究任务, 包括规划、搜索、抓取和总结。相关测试位于 `tests/test_workflow.py`。
- 遇到的挑战 (如有): 在异步环境中管理不同任务之间的状态和依赖关系可能很复杂。确保工作流中每个步骤的错误处理和重试机制的健壮性是一个关键挑战。

智能源选择 (天气, 交通, 金融等)

- 状态: 已实现并已测试

- **实现细节:** 系统使用一个混合路由器 (`src/routing/hybrid_router.py`)，它首先尝试使用高效的关键词匹配来分类查询。如果置信度较低，则回退到功能更强大的基于LLM的路由器。该路由器能识别特定领域查询的意图，并将其路由到正确的工具（`WeatherTool`, `FinanceTool`等）。`test_routing.py` 的存在证实了此功能的测试。
- **遇到的挑战 (如有):** 主要挑战是在关键词路由器的速度和LLM路由器的准确性之间取得平衡。微调置信度阈值和扩展关键词列表以覆盖更多用户表达方式，同时避免产生歧义，是一个持续的过程。

本地RAG集成

- **状态:** 已实现并已测试
- **实现细节:** RAG（检索增强生成）系统已完全实现。它使用 `src/tools/document_processor.py` 来提取文本和分块文档，使用 `sentence-transformers` 创建嵌入，并使用 ChromaDB 作为持久化向量存储 (`src/tools/vector_store.py`)。RAGAgent (`src/agents/rag_agent.py`) 负责处理检索、上下文增强和生成过程。
- **遇到的挑战 (如有):** 一个关键挑战是优化分块策略以保持语义上下文，这对检索质量至关重要。另一个挑战是随着文档数量的增长，如何有效管理向量数据库并确保高效的相似性搜索。

重排与过滤

- **状态:** 已实现
- **实现细节:** 系统中已部署一个重排管道 (`src/tools/reranker.py`, `reranker_hybrid.py`)。从向量存储进行初步检索后，会使用更复杂的重排模型（例如，来自`transformers`库的Cross-Encoder）根据与查询的语义相关性对结果进行重新排序。这显著提高了提供给LLM的上下文质量。
- **遇到的挑战 (如有):** 重排会增加计算开销和延迟。挑战在于选择一个能在不过多延长用户等待时间的情况下，显著提升结果质量的模型。文件名中提到的混合方法可能就是为了平衡性能和准确性。

多模态支持 (例如, 图像/音频输入)

- **状态:** 已实现 (针对图像)
- **实现细节:** 系统支持图像输入。它使用 `paddleocr` 对图像进行光学字符识别 (OCR) (`src/tools/ocr_tool.py`)，并与像Gemini这样的多模态LLM集成以进行常规图像理解 (`src/tools/vision_tool.py`)。这允许用户就图像中的文本或图像内容提问。当前代码库中没有证据表明支持音频输入。
- **遇到的挑战 (如有):** 集成多个OCR和视觉模型，并根据用户查询将请求路由到正确的模型是一个挑战。高效地处理各种图像格式和尺寸也需要仔细的实现。