

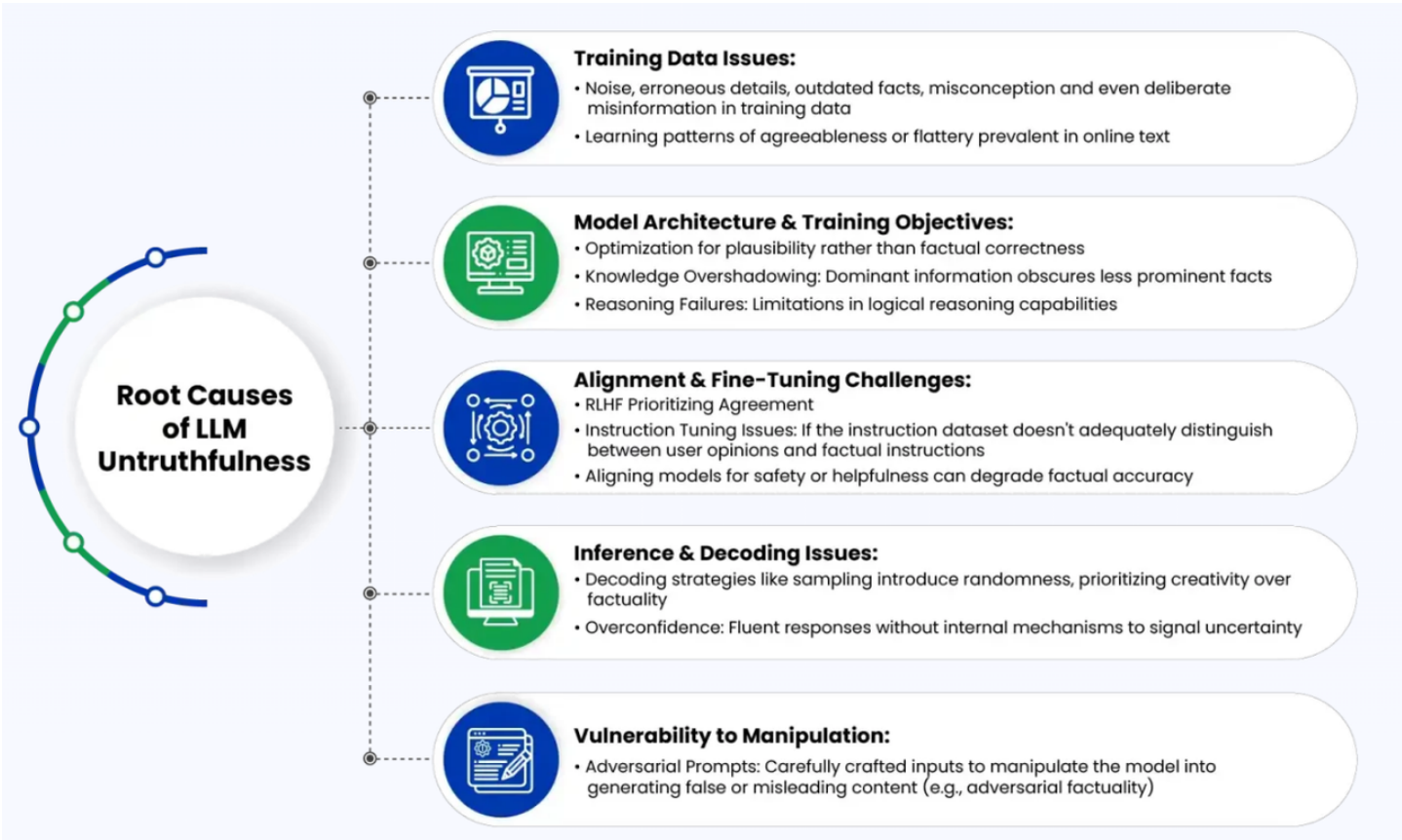
AIC-2025-87935724-场景创新-AdaptiMultiRAG-技术报告

作品概述

背景介绍

大模型时效性与专业能力局限

当前大语言模型面临两个根本性瓶颈：知识时效性滞后与垂直领域专业能力不足。由于模型训练数据的时效性限制，大模型的知识更新通常滞后数月甚至数年，无法回答最新技术发展和前沿研究问题。同时，面对各垂直领域的专业技术问题时，通用大模型往往缺乏足够的领域知识深度，难以提供准确、专业的回答。虽然ChatGPT、Kimi、Gemini等主流AI助手支持网络搜索，但受限于技术和成本，仍无法通过模型微调的方式让用户自主学习特定领域的专业知识。



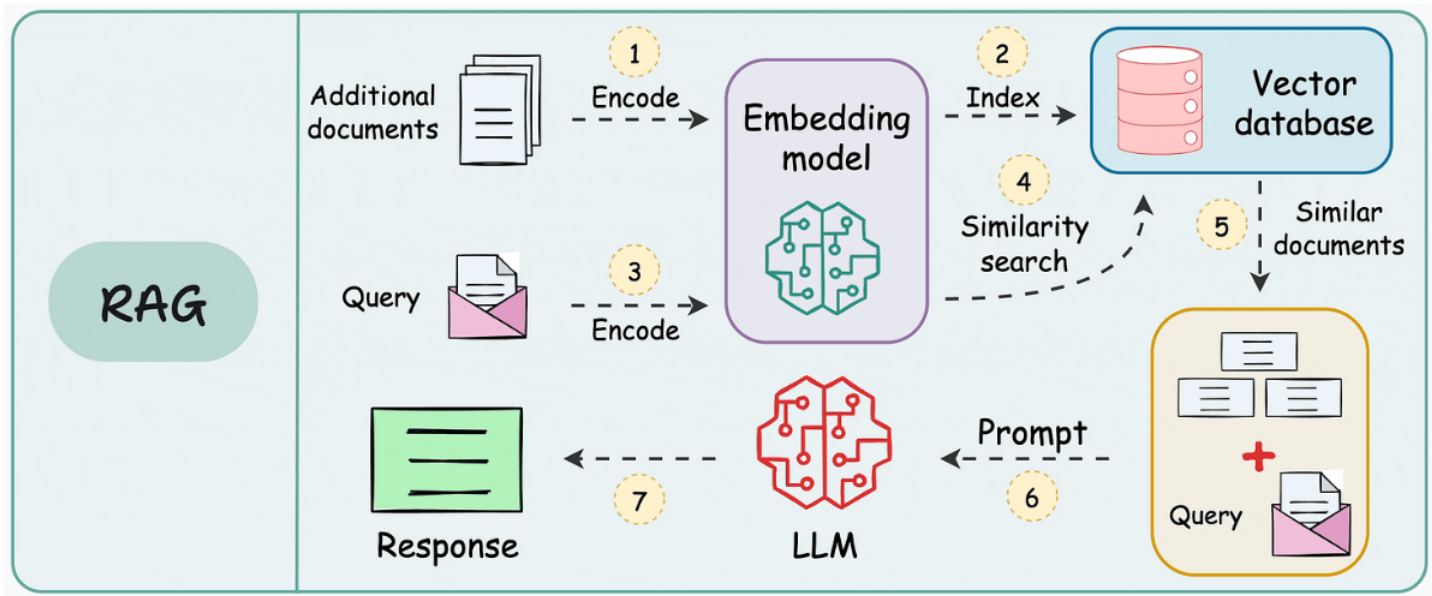
技术工作者信息获取困境

对于资深技术工作者而言，信息获取效率低下已成为制约工作效能的关键瓶颈。在面对复杂的技术文档、API规范、学术论文等专业资料时，技术人员需要花费大量时间进行文献检索、资料筛选和信息整

理。据统计，科研人员平均需花费30-40%的工作时间在文献检索和资料整理上。面对海量的技术文档，人工检索不仅效率低下，而且容易遗漏重要信息，严重影响工作进度和决策质量。

单一RAG技术方案局限性

现有的RAG（检索增强生成）技术虽能在一定程度上缓解大模型的知识局限，但单一检索策略存在明显不足。传统向量检索在面对定义类问题时表现尚可，但在处理实体关系、因果推理、技术关联等复杂问题时效果欠佳，难以保留文档中的结构化信息。同时，现有RAG系统大多依赖人工上传文档，无法自动获取最新资料，系统决策过程缺乏透明度，难以满足科研场景



目标定位

针对背景中提出的大模型时效性局限、技术工作者信息获取困境、单一RAG方案效果不佳三大痛点，我们构建了AdaptiMultiRAG自适应多RAG智能体系统，提供针对性的解决方案：

解决大模型时效性与专业能力不足

- **智能爬虫自动更新**：集成Crawl4AI智能爬虫，支持arXiv、GitHub等平台的实时抓取，自动获取最新技术文档和前沿研究成果，解决知识滞后问题
- **垂直领域知识库构建**：用户只需提供文档链接，系统即可自动构建专业的领域知识库，实现特定领域知识的深度学习与积累
- **多模型协同机制**：结合向量检索与知识图谱检索，充分发挥不同技术优势，提升专业问题回答的准确性和深度

破解技术工作者信息获取困境

- **一键式知识库构建**：面对复杂文档，用户只需输入链接即可快速构建RAG系统，无需繁琐的文档整理和上传过程
- **智能问答替代人工检索**：通过自然语言提问即可精准获取所需信息，将原本需要数小时的文献检索工作缩短至数分钟

- **多模态文档处理**：支持PDF、Word、Markdown等多种格式，结合OCR技术处理扫描文档，全面提升信息获取效率

突破单一RAG技术局限性

- **自适应双模式检索**：根据问题类型智能选择向量检索或知识图谱检索，定义类问题用向量检索确保精度，关系类问题用图检索保留结构，复杂问题采用混合检索发挥协同优势
- **全流程自动化**：从文档爬取、内容解析、知识抽取到检索生成，实现端到端自动化处理，减少人工干预
- **可视化透明决策**：通过Agent执行流程可视化和知识图谱展示，让用户清晰了解AI的决策过程，增强系统可信度

核心功能架构：

- **双模式融合检索**：向量+知识图谱自适应选择
- **智能爬虫集成**：实时获取最新文献资料
- **完整文档处理**：多格式支持+智能切块+OCR识别
- **可视化执行流程**：Agent工作流透明化展示
- **智能记忆管理**：多轮对话上下文连贯

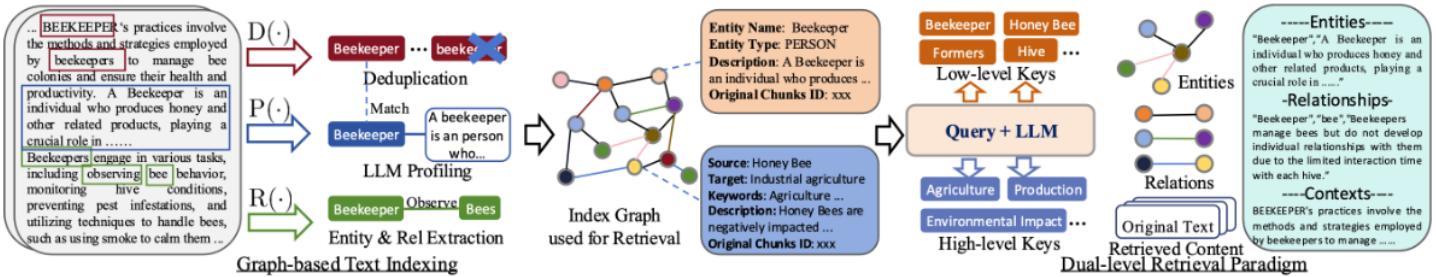


Figure 1: Overall architecture of the proposed LightRAG framework.

Lightrag架构图

需求分析

问题剖析

针对所选领域场景，深入分析作品要解决的具体问题，从多维度 挖掘问题本质，明确问题的关键难点与痛点。

需求调研

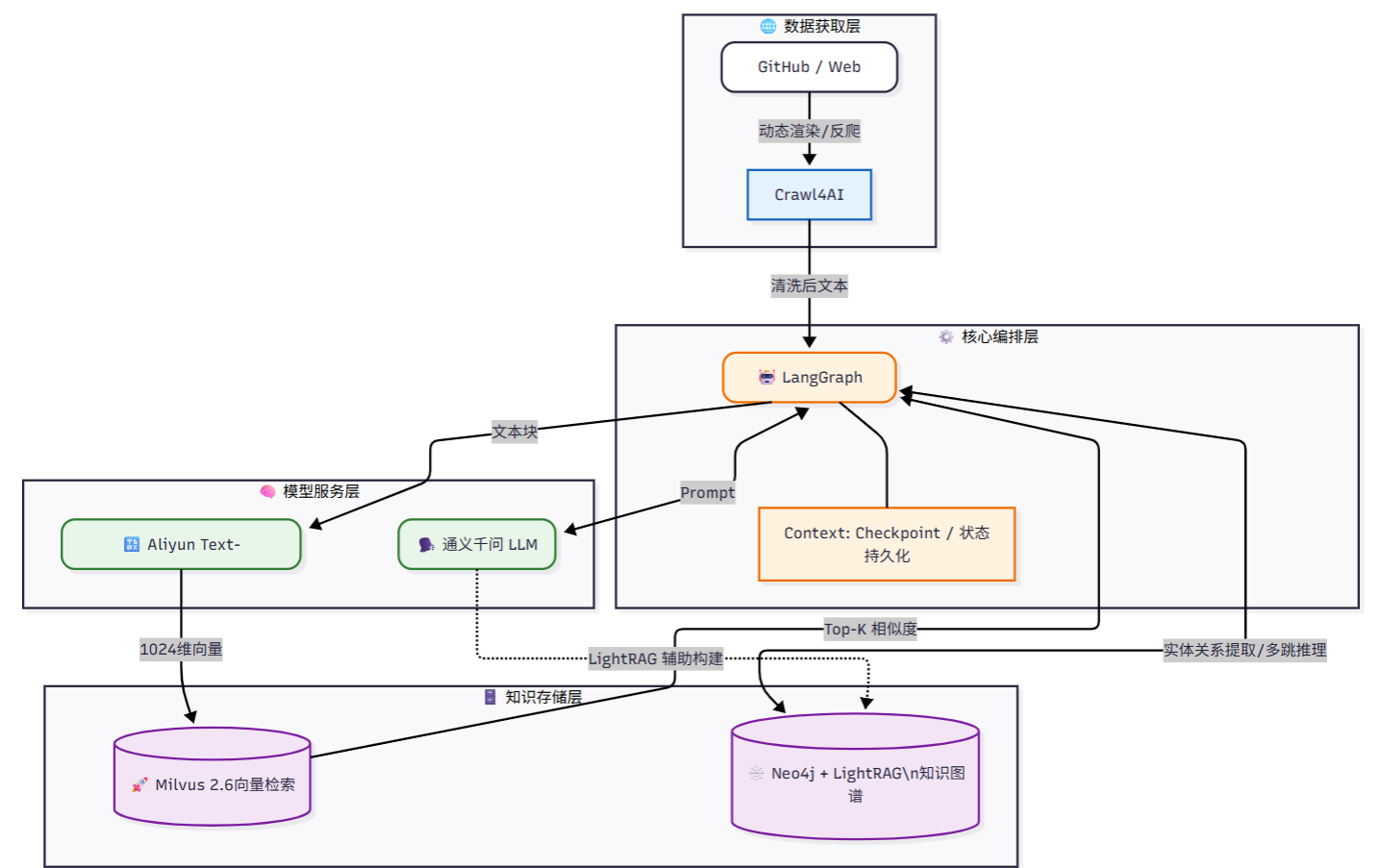
通过调查、访谈、市场分析、文献研究、现场试验等方式，收集 分析目标用户、行业或某个应用场景需求，或者是自己在学习、科研工作中的发现的需求，或者是承担某个纵向（横向）项目中的需求等，整理归纳功能、性能、体验等 方面的具体要求

AI技术工具选择与运用

结合所选场景需求与智能体功能目标，我们选择了以下人工智能技术与工具：

核心AI技术：

- **大语言模型**：阿里云通义千问系列，支持中英文混合处理，32K上下文窗口适合长文档，响应速度快(<1s)
- **智能体框架**：LangGraph 0.6+，提供基于有向图的工作流编排，内置checkpoint机制，支持流式输出和状态持久化
- **向量数据库**：Milvus 2.6，支持亿级向量检索，采用IVF_FLAT索引和COSINE余弦相似度，查询性能<100ms
- **知识图谱框架**：LightRAG + Neo4j组合，自动从文档构建知识图谱无需人工标注，支持实体识别、关系抽取和多跳推理
- **智能爬虫**：Crawl4AI作为AI驱动的智能爬虫，支持动态JavaScript渲染，内置反爬虫机制，优化覆盖arXiv论文、GitHub文档等
- **嵌入模型**：阿里云text-embedding-v4提供1024维向量，针对中英文文本优化



核心技术汇总

核心AI技术架构优势：

1. LangGraph + LangChain 企业级智能体框架

- **高度灵活可控**：与Coze等低代码平台不同，LangGraph提供代码级别的完整控制能力，支持复杂业务逻辑的精细化实现，满足企业级应用的定制化需求
- **工作流编排能力**：基于有向图的智能体工作流设计，支持条件分支、循环、并行处理等复杂流程，实现8节点自适应检索流程的精确控制
- **状态持久化机制**：内置checkpoint机制支持对话状态持久化，确保长对话的连续性和系统稳定性
- **企业级特性**：支持高并发、错误处理、监控日志等企业级功能，具备生产环境部署能力
- **生态集成优势**：与LangChain生态无缝集成，可直接调用丰富的工具链和预置组件

2. Crawl4AI AI驱动智能爬虫

- **智能内容提取**：不同于传统爬虫的固定规则，Crawl4AI利用AI理解网页结构，自动识别和提取核心内容，适应不同网站布局变化
- **动态页面支持**：内置JavaScript渲染引擎，完美处理arXiv、GitHub等现代化动态网站，获取传统爬虫无法触及的内容
- **反爬虫突破**：智能模拟人类浏览行为，绕过常见的反爬虫机制，确保数据获取的稳定性和持续性
- **结构化数据输出**：AI自动将非结构化网页内容转换为结构化数据，直接存入知识库，无需后续人工处理
- **平台适配优化**：针对科研平台特点进行专门优化，支持论文元数据提取、API文档结构化处理等

3. LightRAG + Neo4j 智能知识图谱

- **自动图谱构建**：LightRAG自动从文档中识别实体、抽取关系，无需人工标注即可构建高质量知识图谱，大幅降低知识图谱构建成本
- **多跳推理能力**：Neo4j图数据库支持复杂的多跳关系查询，能够发现文档中隐含的关联关系，实现深度知识发现
- **语义理解增强**：结合向量检索和图检索的优势，既保持语义相似性匹配，又保留实体关系的结构化信息
- **动态更新支持**：支持知识图谱的增量更新和版本管理，新文档可自动融入现有知识网络
- **可视化交互**：ECharts力导向图实时展示知识网络，支持交互式探索，提升用户体验

技术协同优势：

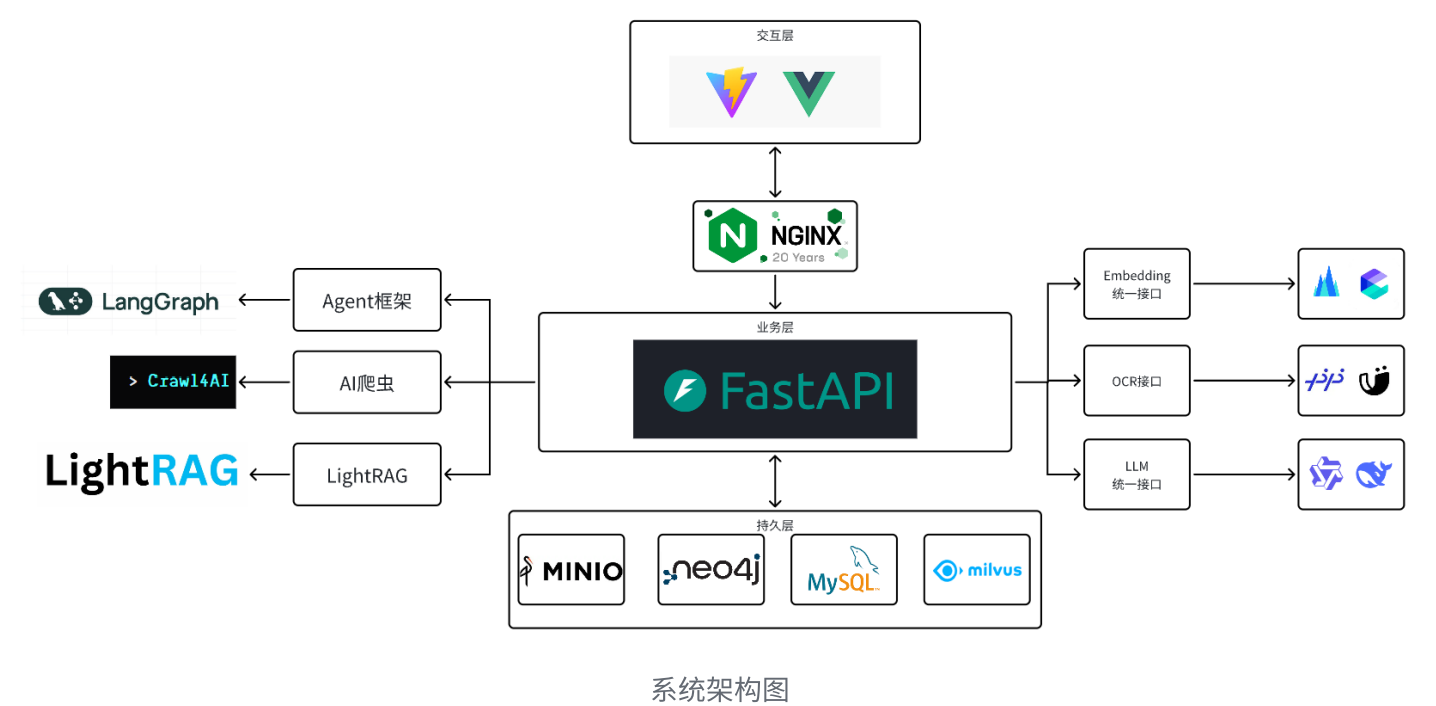
- **LangGraph统一调度**：通过LangGraph工作流统一协调爬虫、检索、图谱构建等各环节，实现端到端自动化

- **数据闭环处理**：Crawl4AI获取原始数据 → LightRAG构建知识图谱 → LangGraph调度智能检索 → 结果反馈优化爬虫策略
- **企业级部署能力**：整套技术栈支持容器化部署、水平扩展、监控告警等企业级特性
- **技术创新突破**：三项核心技术形成独特的技术壁垒，在自适应检索、智能爬取、知识图谱构建方面具有显著

核心创新价值：

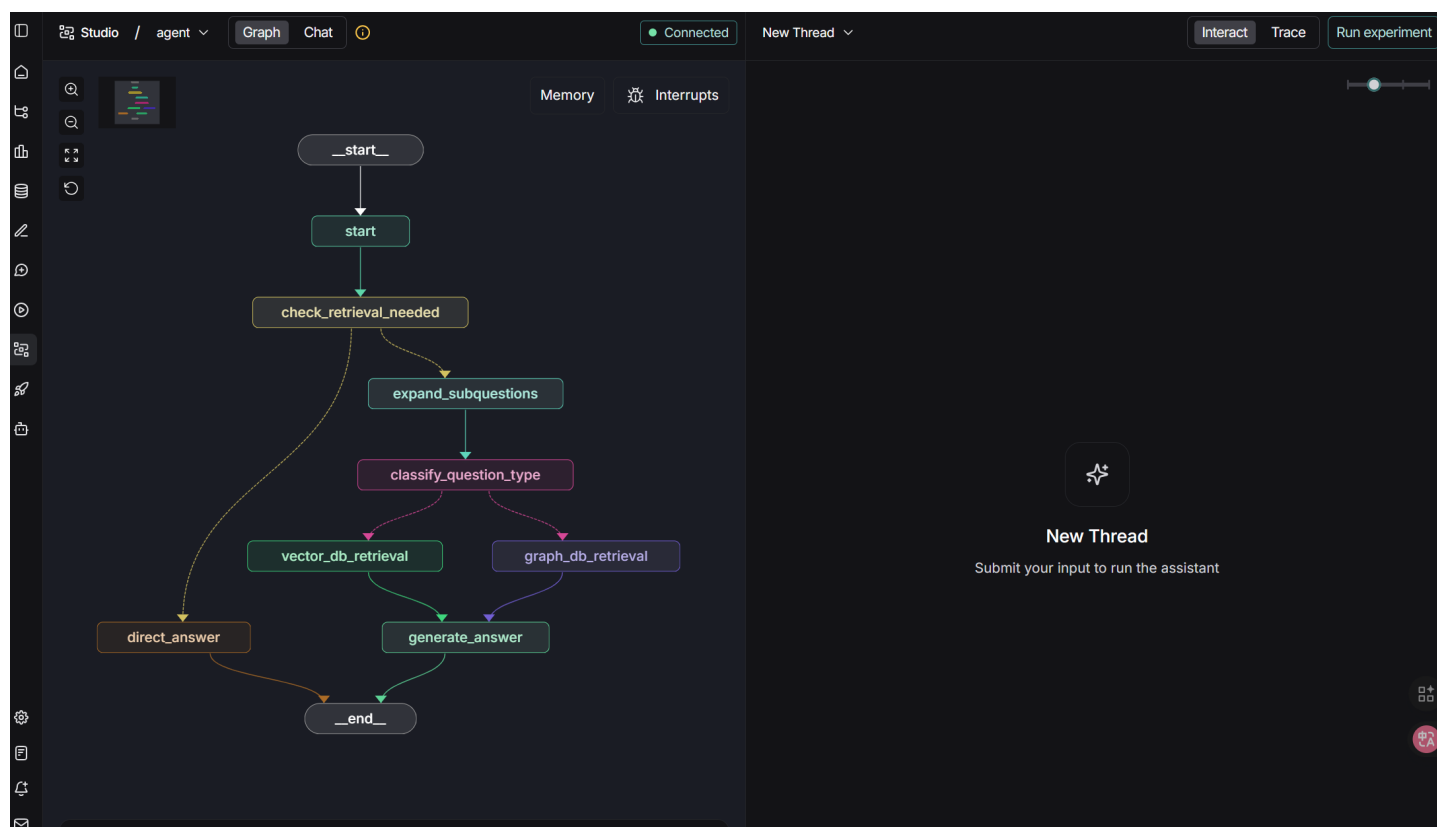
通过LangGraph的企业级控制能力、Crawl4AI的智能获取能力、LightRAG+Neo4j的深度理解能力，构建了区别于传统RAG系统的全新技术架构，实现了从静态检索到动态学习、从单一模式到自适应智能的跨越式升级。

项目实施



关键技术实现

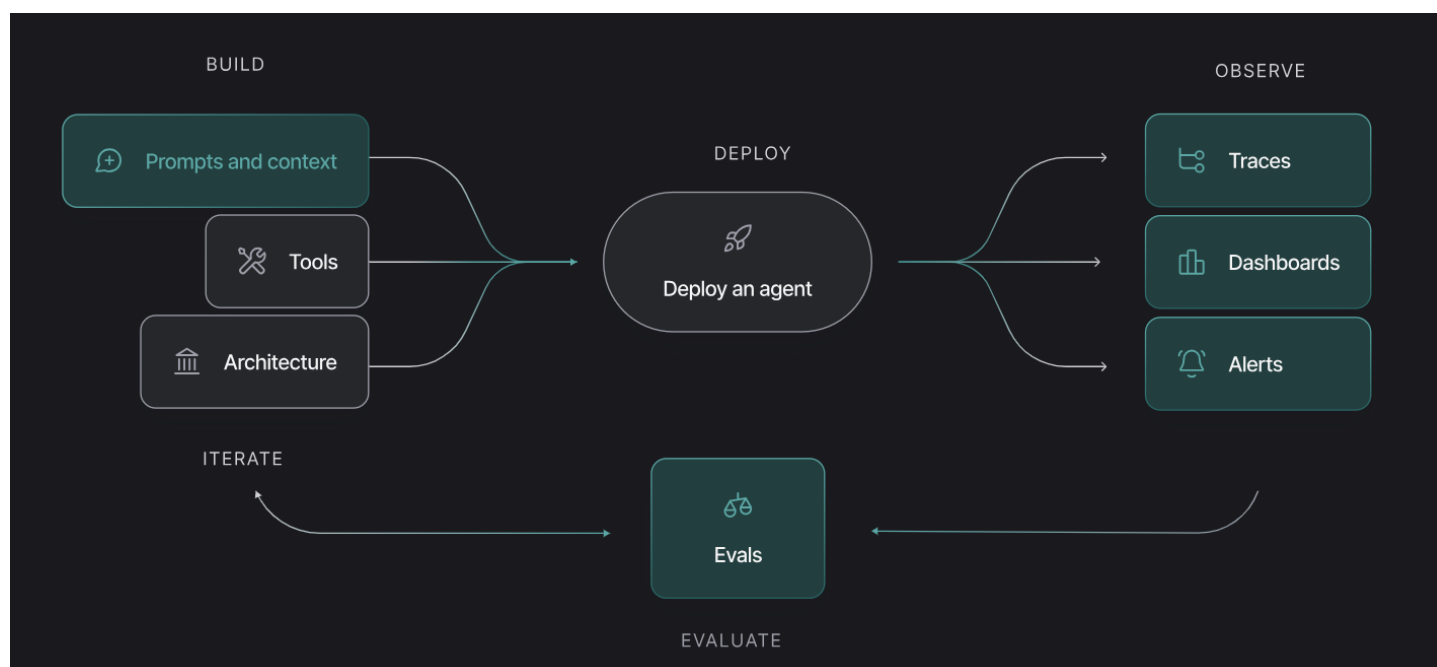
- 向量检索实现**：文档经过格式解析、OCR处理、智能切块后，通过阿里云embedding服务转化为1024维向量存储到Milvus。检索时问题向量化后进行余弦相似度匹配，返回Top-K结果。系统采用动态collection创建机制，每个知识库拥有独立的向量空间，确保数据隔离。
- 知识图谱检索实现**：使用LightRAG从文档自动构建知识图谱，包括实体识别、关系抽取和图存储到Neo4j。检索时通过Cypher查询语言支持单跳查询、多跳推理和路径查找。LightRAG支持local、global、hybrid三种检索模式，适应不同查询需求。
- Agent自适应路由实现**：LangGraph工作流包含问题路由节点、问题分类节点、向量检索/图检索节点和答案生成节点。通过LLM判断问题类型（定义类、关系类、混合类），动态选择最优检索策略，实现智能路由决策。



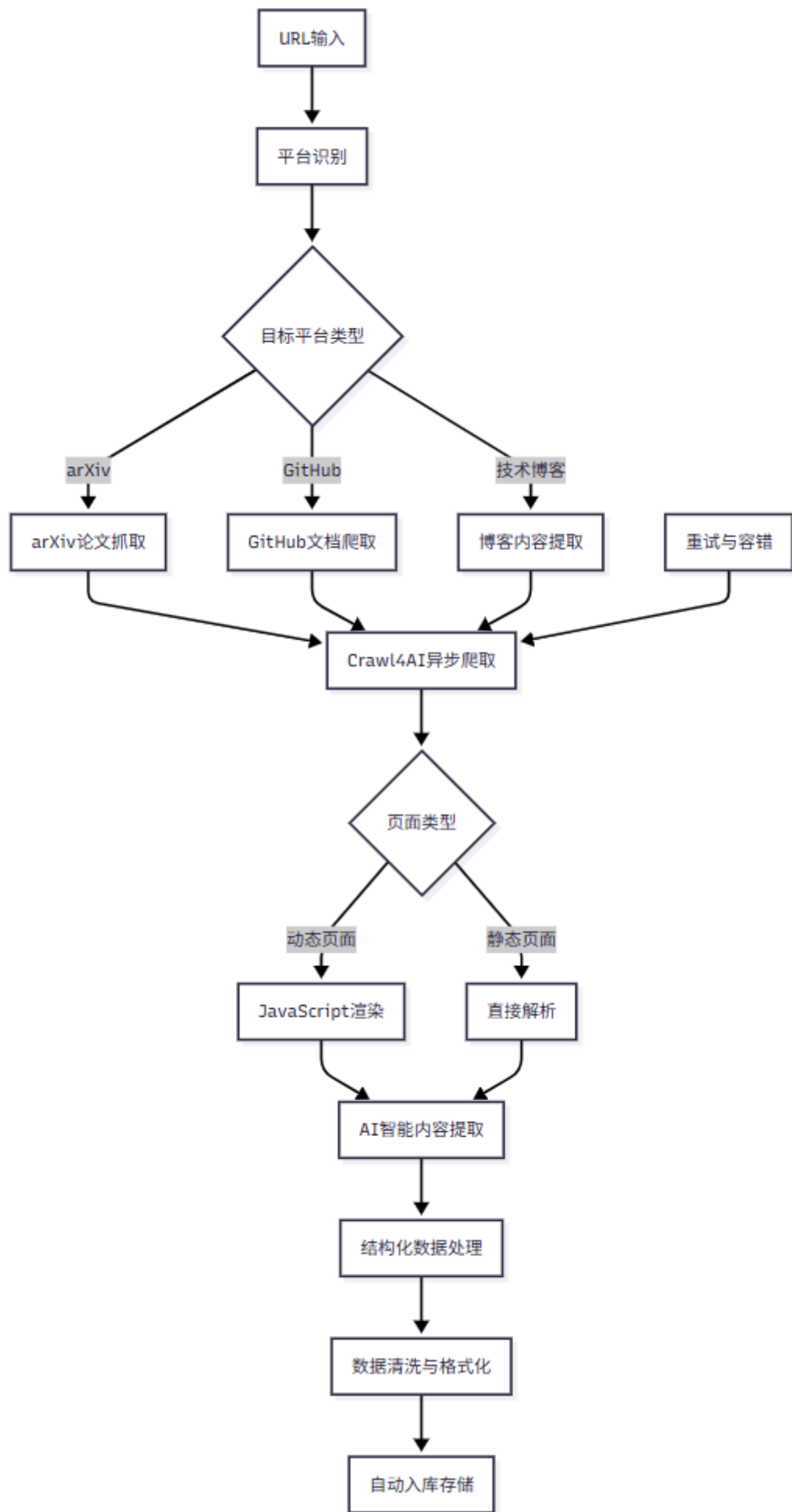
LangSmith可视化检测：集成LangSmith平台对Agent执行流程进行深度监控和可视化分析，提供企业级LLM应用的可观测性解决方案。

LangSmith核心能力：

- **实时监控：**实时跟踪LangGraph工作流中每个节点的执行状态、耗时和资源消耗
- **详细追踪：**记录LLM调用链、prompt和response详情、token使用情况等关键信息
- **性能分析：**提供响应时间、成功率、成本等KPI指标的趋势分析和异常检测
- **调试工具：**支持单步调试、错误回溯、执行路径分析，快速定位问题根因
- **版本对比：**支持不同版本Agent的性能对比和回归测试，确保系统稳定性



智能爬虫实现： Crawl4AI集成支持多个科研平台，包括arXiv论文抓取、GitHub文档爬取和技术博客提取。采用异步爬取、结构化提取和自动入库流程，支持JavaScript渲染和反爬虫机制。



crawl4ai流程

流式交互实现：后端使用FastAPI的StreamingResponse支持SSE流式输出，LangGraph的astream方法逐步产生节点事件，前端EventSource接收流式数据实现打字机效果。StreamChatManager类统一管理流式状态和事件分发。

作品展示

开源地址：

- GitHub仓库：<https://github.com/zxj-2023/AdaptiMultiRAG.git>

- 采用MIT开源协议，包含完整的前后端代码、配置文件、文档和测试用例

部署访问：系统支持本地部署和云端部署，提供Docker容器化部署方案，便于快速搭建和使用。

公网地址：

整个系统全部由我们自行开发，主要包含以下模块：

系统登录

前端系统：基于Vue 3.5 + Vite 7.x + Element Plus构建现代化界面，采用类似Claude AI的纸质感设计风格，支持响应式布局和移动端适配。主要功能包括登录注册、知识库管理、文档上传、智能对话、可视化展示等。

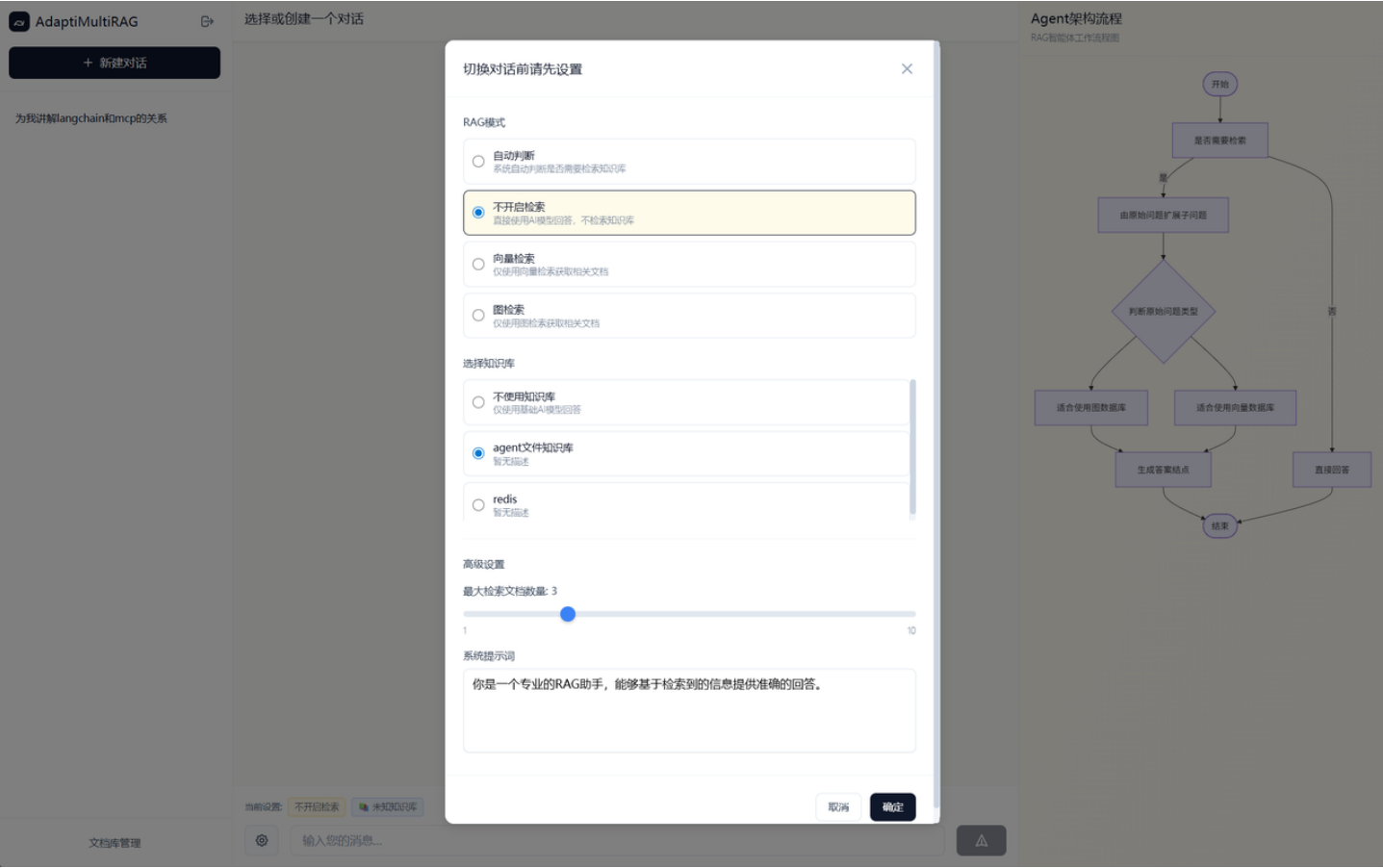


后端系统：基于FastAPI 0.115+构建异步RESTful API，集成LangGraph工作流引擎、Milvus向量检索、Neo4j图数据库、MySQL业务数据存储等。实现用户认证、文档处理、检索路由、流式响应等核心功能。

完整功能模块：

- 用户认证系统（JWT双token机制）
- 多知识库管理（Collection隔离）
- 文档处理流水线（PDF/Word/Markdown/OCR）
- 自适应检索引擎（向量+图谱双模式）
- 智能爬虫集成（Web/GitHub等）
- 流式对话交互（langmem记忆管理）
- Agent流程可视化（Mermaid动态展示）
- 知识图谱可视化（ECharts力导向图）

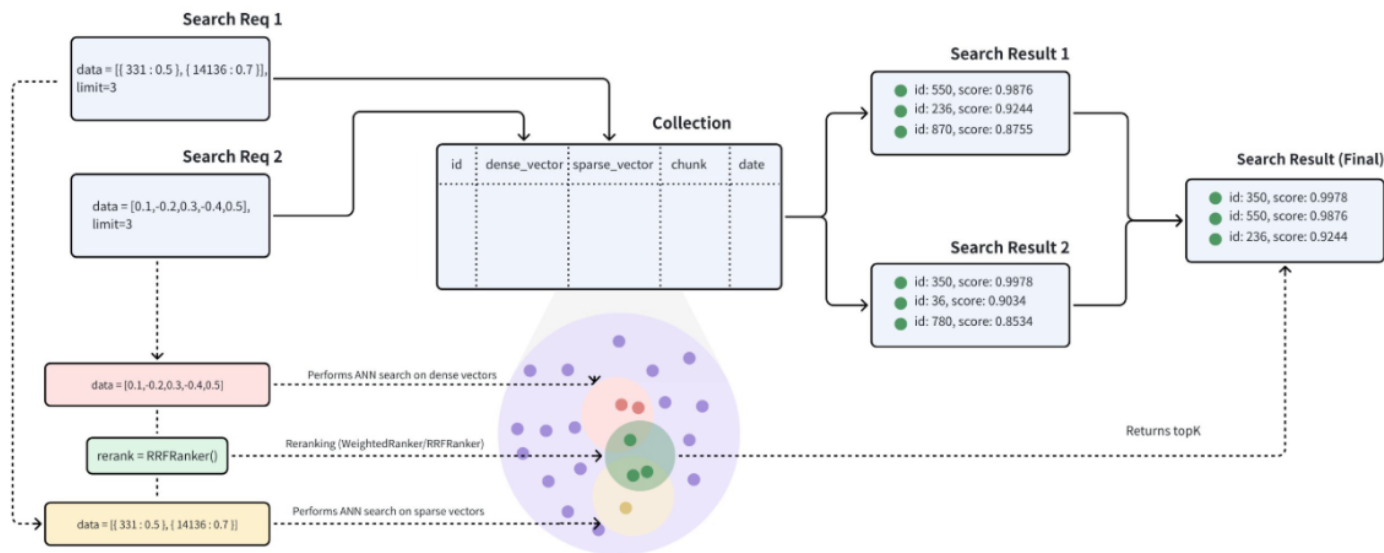
知识库选择



对话前，请选择检索模式，知识库，检索文档数与系统提示词的设置，支持以下几种检索模式

- 1.向量检索模式：** 基于Milvus向量数据库，采取向量的混合检索，结合使用**密集向量**查询产品的文本描述和使用**稀疏向量**的关键词匹配，实现对相关文档的检索，并搭配**Rerank模型**进行重排序。
- 2.知识图谱检索模式：** LightRAG自动识别实体、抽取关系构建**知识图谱**，Neo4j图数据库存储实体节点和关系边，通过Cypher查询语言支持**多跳推理**。

3.自动识别检索模式：LangGraph搭建Agent，智能路由节点动态选择最优策略，根据用户问题自动选择检索模式，结果融合算法综合两种检索的优势，LLM基于融合结果生成最终答案



向量检索工作流程

知识库问答

AdaptiMultiRAG

+ 新建对话

为我讲解langchain和mcp的关系

为我讲解langchain和mcp的关系

> 开始

> 检索需求判断

< 子问题扩展

节点名称为expand_subquestions, 扩展子问题为1. langchain是什么?
2. mcp是什么?
3. langchain和mcp之间有什么联系或集成方式?
4. 在实际应用中, langchain如何与mcp协同工作?

< 问题类型分类

节点名称为classify_question_type, LLM判断检索模式为graph_only, 理由为用户的问题明确询问langchain和mcp的关系是什么?, 这是一个典型的关系查询, 需要找出两个具体实体 (langchain和mcp) 之间的关联。图检索专门用于处理实体间关系的查询, 能够通过图数据库中的节点和边来找到这两个技术之间的连接关系, 比如是否是集成关系、依赖关系、竞争关系等。向量检索更适合语义相似性或概念性问题, 而这个问题需要的是精确的结构化关系信息。

> 图数据库检索

LangChain 与 MCP 的关系

LangChain 与 Model Context Protocol (MCP) 之间的关系是一种集成与适配的关系: LangChain 通过专门的适配层库支持 MCP 协议, 从而使其智能体 (agents) 能够调用由 MCP 服务端提供的标准化工具。

具体而言, MCP 是一个开放协议, 旨在标准化 AI 应用 (如大语言模型驱动的智能体) 如何连接外部系统、访问数据源和使用工具。它类似于 AI 应用的 USB-C 接口, 为不同工具和上下文提供统一的接入方式 [DC] unknown_source。

为了在 LangChain 中使用 MCP 工具, 开发者需要安装 'langchain-mcp-adapters' 库。该库提供了 'MultiServerMCPClient' 等组件, 允许 LangChain 智能体动态加载并调用运行在本地或远程 MCP 服务器上的工具 (例如数学计算、天气查询等) [DC] unknown_source。这些工具可以基于不同的传输机制 (如 stdio 或 Streamable HTTP) 进行通信 [DC] unknown_source。

此外, 用户也可以使用 'mcp' 库自行构建 MCP 服务器, 并通过 LangChain 集成到其智能体工作流中, 这使得 LangChain 能够无缝接入符合 MCP 标准的任意工具生态系统, 提升智能体的能力和灵活性 [DC] unknown_source。

当前设置: 自动判断 agent文件知识库

输入您的消息...

Agent架构流程

RAG智能体工作流程图

开始

是否需要检索

是

由原始问题扩展子问题

判断原始问题类型

适合使用图数据库

适合使用向量数据库

生成答案节点

直接回答

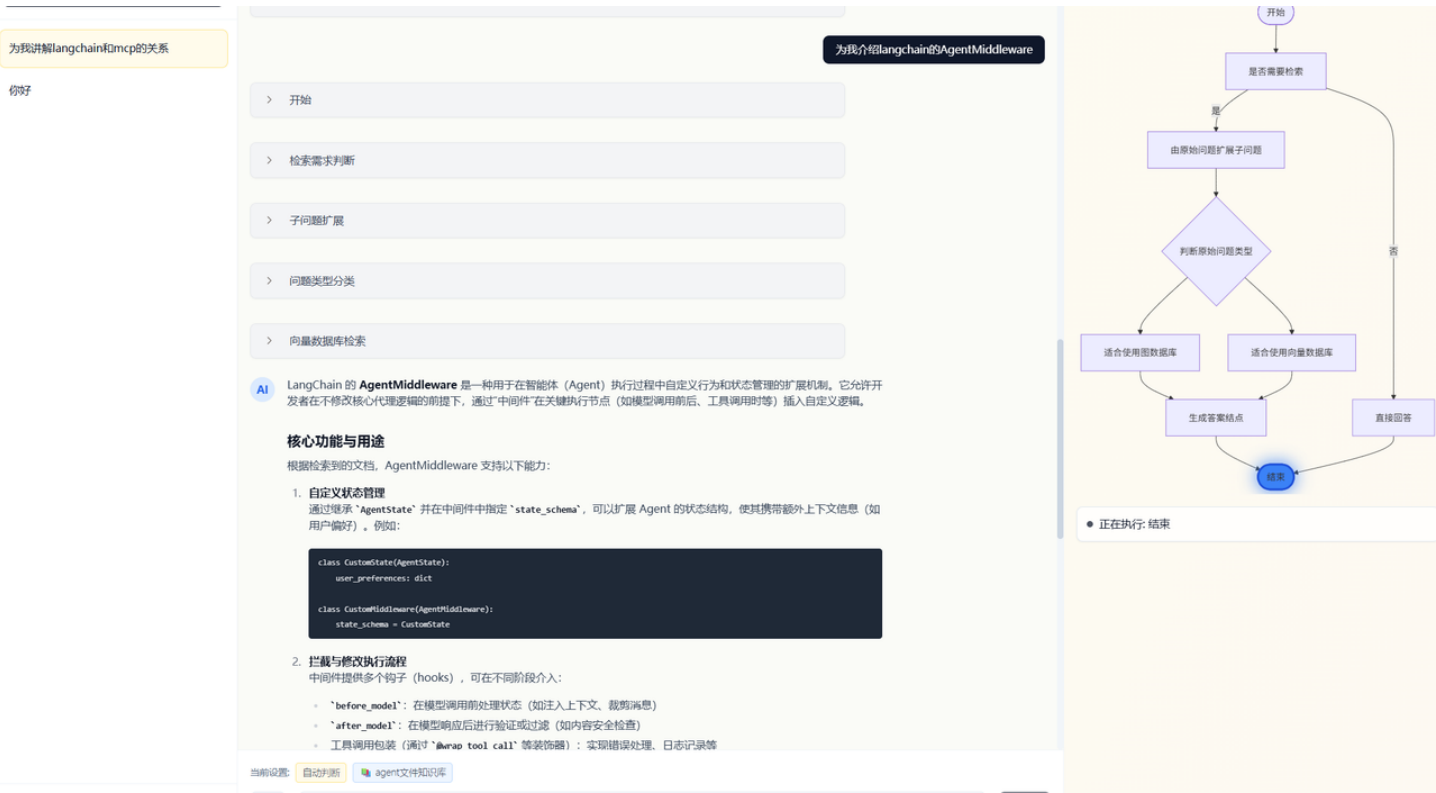
否

结束

正在执行: 结束

在本次演示中，我选取了agent文件知识库与自动判断检索模式。


- 1. 页面右侧实时展示 **agent 的整体架构与判断流程图**；在问答执行过程中，界面会高亮当前正在运行的节点（节点运行到哪一步一目了然），并以时间线形式呈现执行顺序。
- 2. 每个节点都支持**下拉菜单**查看其内部信息，便于用户随时审计 agent 的决策依据与中间结果，了解 agent 的内部信息
- 3. 以“langchain 与 mcp 的关系”这一问题为例，系统的问题分类节点会先调用轻量级 LLM 进行问题类型判定；由于该问题属于**关系型查询**（关注实体间的联系与依赖），agent 智能选择知识图谱检索作为优先策略。图检索模块（LightRAG + Neo4j）执行多跳路径查找与实体关系匹配，同时返回路径上下文与源文档片段供答案生成节点参考。整个过程的每一步输出都可通过界面下拉查看，保证检索与生成的可解释性与可追溯性。



本次演示中，我选择了一个具有代表性的技术问题："请介绍langchain的AgentMiddleware"。AgentMiddleware是langchain 1.0版本中新推出的中间件机制，这是一个典型的**定义类问题**，需要对特定技术概念进行准确、全面的解释说明。

在AdaptiMultiRAG系统的处理流程中，当用户输入这个问题后，LangGraph工作流首先启动问题分类节点，通过轻量级LLM（qwen-turbo）快速判断问题类型。系统识别出这是一个介绍性的定义类问题，根据预设的自适应检索策略，智能选择向量数据库作为主要检索源。向量检索在处理定义类问题时具有天然优势，能够通过语义相似度匹配精准定位到文档中与AgentMiddleware相关的核心描述段落，包括其设计理念、技术架构、使用方法和典型应用场景等内容。生成的回答不仅准确描述了AgentMiddleware的核心功能和技术特点，还结合了知识库中的实际代码示例和最佳实践说明，远超通用大模型在相同问题上的表现。

为我介绍langchain的AgentMiddleware

 Qwen3-Max

截至我知识的最新更新（2024年6月），LangChain 并没有一个官方叫做 `AgentMiddleware` 的组件。在 LangChain 的官方文档、GitHub 仓库或主流社区讨论中，并未定义名为 `AgentMiddleware` 的标准类或模块。

不过，我们可以从两个角度来理解你提到的“AgentMiddleware”：

1. 可能是对 LangChain 中“中间件”概念的泛化理解

在 LangChain 的 Agent（智能体）系统中，虽然没有明确叫 `AgentMiddleware` 的类，但你可以通过以下机制实现类似“中间件”的功能：

✅ 自定义工具（Tools）

LangChain 的 Agent 通过工具（Tool）与外部交互。你可以包装工具，在调用前后加入日志、权限检查、缓存、监控等逻辑，这类似于中间件行为。

对比上图展示的通义千问官方的qwen-max的回答效果可以发现，由于通用大模型的训练数据截止时间限制，它对langchain 1.0这类最新技术特性的了解非常有限，往往只能给出模糊的概述性回答，甚至可能产生"幻觉"提供错误信息。而AdaptiMultiRAG系统通过集成最新的官方文档和技术博客，结合精准的向量检索和智能的答案生成机制，能够提供准确度和专业度都显著更高的回答。

知识库添加

文档库管理

←

agent文件知识库

暂无描述

知识库链接

添加文档

+ 新建文档库

agent文件知识库

暂无描述

3 个文档 • 2025/11/15

redis

暂无描述

4 个文档 • 2025/11/14

langchain

网站链接

<https://docs.langchain.com/oss/python/langchain/overview>

2025/11/15

mcp

网站链接

<https://modelcontextprotocol.io/docs/getting-started/intro>

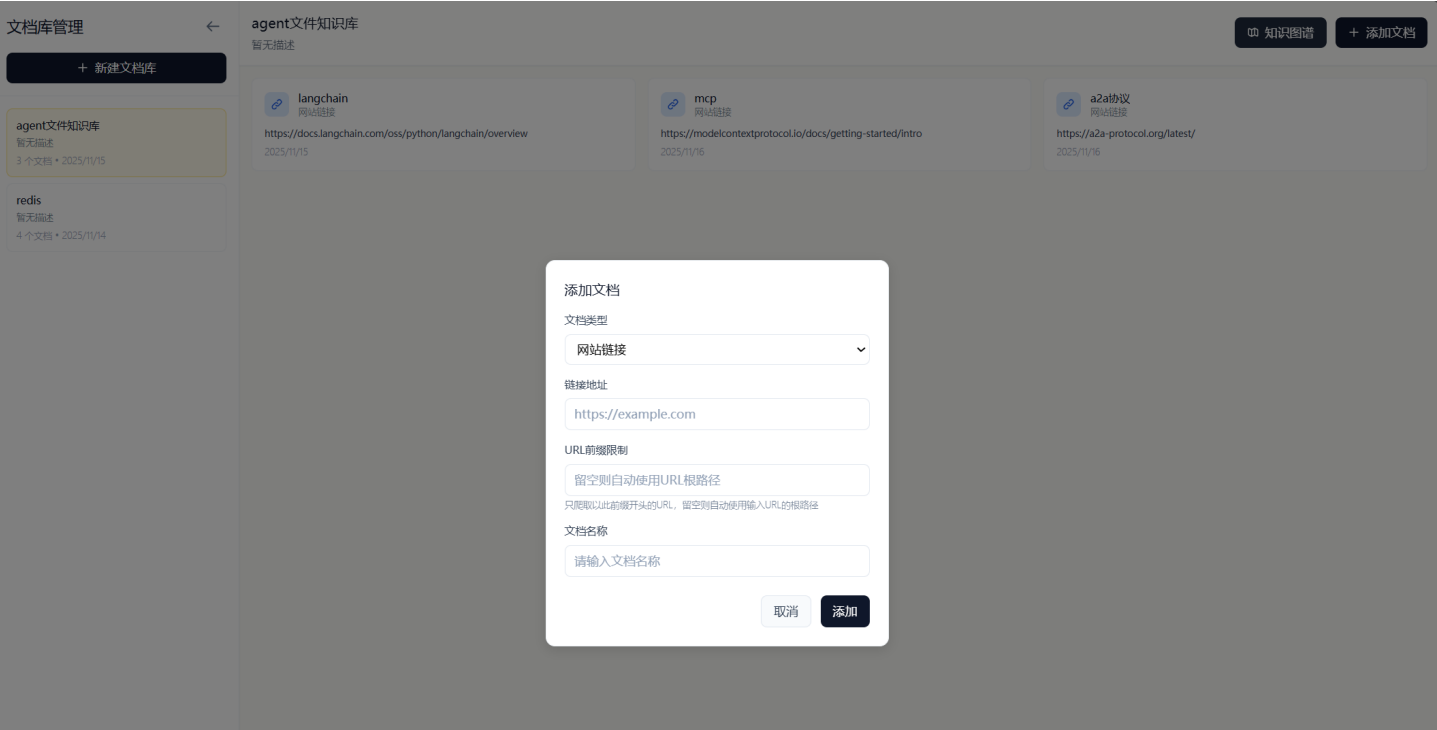
2025/11/16

a2a协议

网站链接


<https://a2a-protocol.org/latest/>

2025/11/16



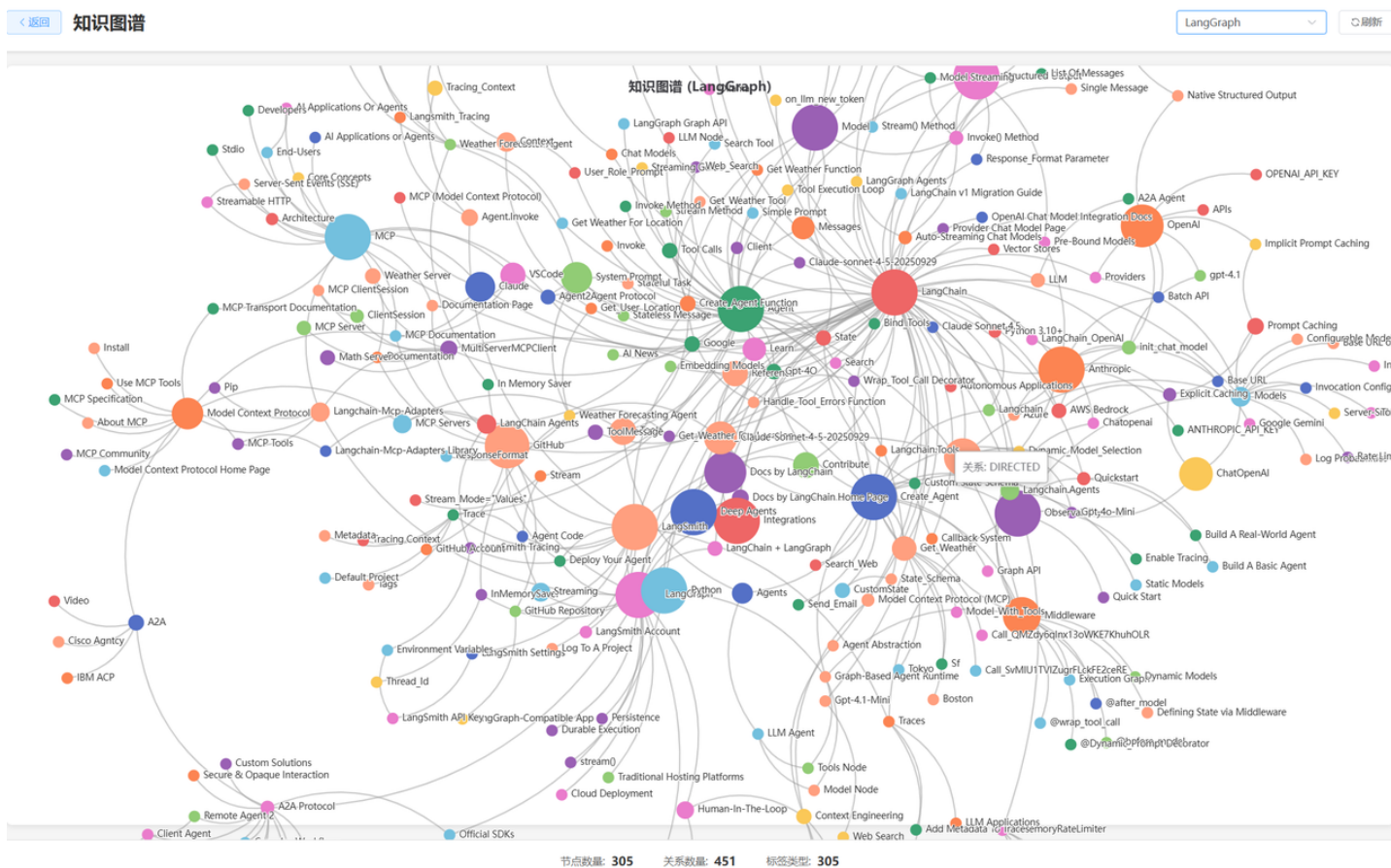
当前展示的是系统的**知识库添加页面**,这是AdaptiMultiRAG区别于传统RAG系统的核心创新功能之一。在该页面中,用户无需经历繁琐的文档下载、格式转换、手动上传等传统流程,仅提供**目标文档的URL链接**,系统即可自动完成从文档获取到知识库构建的**全流程自动化处理**。

系统集成的**Crawl4AI智能爬虫**在接收到用户输入的文档链接后,会首先通过AI算法自动识别目标网站的类型和结构特征。



这里我们为测评人员搭建了两个现有的知识库demo，供测试人员选择：分别为**agent文档知识库**和**运维知识库**

知识图谱可视化



AdaptiMultiRAG系统集成了强大的**知识图谱可视化功能**,通过**ECharts**力导向图技术将抽象的知识关系转化为直观的可视化网络,极大提升了用户对领域知识结构的理解和掌握效率。节点之间的连线表示实体间的语义关系,线条粗细代表关系强度,鼠标悬停在连线上时会显示具体的关系类型(如"依赖于"、"继承自"、"用于"、"实现了"等),让用户精确理解知识之间的内在联系。

系统在界面右上角提供了智能化的**节点筛选面板**,这是知识图谱可视化的一大创新特性。用户可以通过下拉菜单选择感兴趣的实体类型或特定节点,系统会自动高亮选中节点及其直接相关的实体和关系,同时将其他无关节点置为半透明状态,实现焦点式的知识探索。

集成langsmith智能监控agent

The screenshot displays the LangGraph interface, which provides a detailed trace of an agent's execution. The interface is organized into three main sections:

- Left Sidebar (Trace):** Lists the sequence of steps in the execution, including model calls (e.g., qwen3-max), parsing steps (PydanticToolsParser), and retrieval steps (VectorStoreRetriever). Each step shows its duration and token count.
- Central Panel (Input/Output):** Displays the input messages and the final output. The input is a human message: "为我介绍langchain的AgentMiddleware". The output includes the final answer, original question, retrieval mode (auto), session ID, system prompt, user ID, answer sources, graph DB results, and processed questions.
- Right Sidebar (Statistics):** Provides key performance indicators and status information:
 - START TIME:** 11/21/2025, 04:32:00 PM
 - END TIME:** 11/21/2025, 04:32:30 PM
 - TIME TO FIRST TOKEN:** 1.00s
 - STATUS:** Success
 - TOTAL TOKENS:** 3,319 tokens
 - LATENCY:** 29.99s
 - TYPE:** Chain

我们的系统还集成langsmith实现对agent的智能监控，不仅可以后台监控agent的处理流程，还能智能检测agent所花费的tokens与处理的正确率

AdaptiMultiRAG系统深度集成了**LangSmith智能监控平台**,实现了对Agent执行过程的全方位可观测性管理。这一创新特性将传统的黑盒AI系统转变为完全透明、可追溯、可优化的企业级应用,为系统的持续改进和生产环境部署提供了坚实的技术保障。

在后台监控层面,LangSmith**实时记录Agent执行过程中的每一个细节**。当用户发起一次问答请求时,系统会自动创建一个完整的执行轨迹(Trace),详细记录从问题输入、问题路由、问题分类、检索执行、结果融合到答案生成的完整工作流。

在**Token消耗监控**方面,LangSmith提供了精确到每次LLM调用的成本统计功能。系统会自动记录每个节点调用大语言模型时的输入Token数量、输出Token数量、使用的模型类型(如qwen-plus、qwen-turbo)以及对应的计费金额。

在**准确率智能检测**方面,LangSmith集成了多维度的质量评估体系。系统支持自定义评估器(Evaluator),可以基于规则、模型或人工反馈对Agent生成的答案进行自动化评分。

项目目录树状结构图

```
代码块

1  AdaptiMultiRAG/
2  |—— rag-backend/ (后端系统)
3  |   |—— main.py (FastAPI入口)
4  |   |—— langgraph.json (LangGraph配置)
5  |   |—— pyproject.toml (uv依赖管理)
6  |   └── backend/
7  |       |—— .env (环境变量配置)
8  |       |—— init_db.py (数据库初始化)
9  |       |—— agent/ (智能体核心层)
10 |       |   |—— graph/ (LangGraph workflow)
11 |       |   |—— contexts/ (上下文管理)
12 |       |   |—— models/ (智能体模型)
13 |       |   |—— prompts/ (提示词管理)
14 |       |   |—— states/ (状态定义)
15 |       |   └── tools/ (工具函数)
16 |       |—— api/ (API路由层)
17 |       |   |—— auth.py (认证接口)
18 |       |   |—— chat.py (聊天接口)
19 |       |   |—— crawl.py (爬虫接口)
20 |       |   |—— rag.py (RAG核心接口)
21 |       |   |—— knowledge_library.py (知识库管理)
22 |       |   └── visual_graph.py (图谱可视化)
23 |       |—— config/ (配置管理)
24 |       |—— model/ (数据模型)
25 |       |—— param/ (参数配置)
26 |       |—— rag/ (RAG功能层)
27 |       |   |—— storage/ (存储层)
28 |       |       |—— milvus_storage.py
29 |       |       └── lightrag_storage.py
30 |       |   |—— chunks/ (文档切块)
31 |       |   └── outputs/ (输出目录)
32 |       |—— service/ (业务服务层)
33 |       |—— utils/ (工具函数)
34 |       └── tests/ (测试文件)
35 |
36 |—— rag-frontend/ (前端系统)
37 |   |—— package.json (npm配置)
38 |   |—— vite.config.js (Vite配置)
39 |   |—— tailwind.config.js (Tailwind配置)
40 |   └── index.html (入口HTML)
```

```
41 | | └─ src/ (源码目录)
42 | |   └─ App.vue (根组件)
43 | |   └─ main.js (应用入口)
44 | |   └─ style.css (全局样式)
45 | |   └─ api/ (API接口封装)
46 | |   └─ assets/ (静态资源)
47 | |   └─ components/ (可复用组件)
48 | |   └─ router/ (路由配置)
49 | |   └─ stores/ (Pinia状态管理)
50 | |     └─ auth.js, chat.js等
51 | |   └─ utils/ (工具函数)
52 | |     └─ views/ (页面组件)
53 | |       └─ Login.vue (登录注册)
54 | |       └─ Chat.vue (智能对话)
55 | |       └─ DocumentLibrary.vue (文档管理)
56 | |       └─ KnowledgeGraph.vue (知识图谱)
57 | |       └─ History.vue (对话历史)
58 | |       └─ Demo.vue (演示页面)
59 | └─ public/ (公共资源)
```