文献调研分析 Agent 系统方案设计

(A)使用 SerpApi 方案

本文档详细设计了一个支持多 Agent 协同的文献调研分析系统方案。该系统围绕黑板架构构建，各功能通过 Agent 模块化实现，可根据科研需求自动检索文献、提取信息并进行多层次分析，同时为关键文献构建 RAG（Retrieval-Augmented Generation）知识库供进一步智能应用使用。下文涵盖系统架构、数据流与模块分工、Agent 接口、黑板/前端交互机制、RAG 知识库构建流程、工具链选型建议，以及扩展性与错误处理策略等各方面，旨在作为进入开发阶段的技术说明。

系统架构概览

系统采用**多 Agent 黑板架构**。黑板系统作为共享知识库，存储各 Agent 产生的中间结果和最终信息，各 Agent 监听黑板更新并根据自身职责进一步处理。整个流程由一个**控制壳（Control Shell）**协调，确保各 Agent 按序或并发工作并避免冲突。主要组件包括：

- **用户接口（UI)**：研究者通过前端提出科研问题，浏览系统输出，并可标记关键文献或补充输入。
- **中央控制器**：接收用户问题，在黑板上初始化问题描述与任务队列，激活各功能 Agent。
- **黑板数据库**：共享存储，所有 Agent 从中读取任务输入，并将各自结果写入其中。可采用内存数据库或文档数据库保存结构化信息，支持订阅/发布机制实现 Agent 间解耦通信。
- **功能 Agents**：围绕任务需求划分的多个模块化 Agent，例如关键词生成 Agent、文献检索 Agent、元数据提取 Agent、深度分析 Agent、RAG 构建 Agent 等。每个 Agent 监视黑板上相关触发信息并执行特定功能，将结果发布回黑板。

- **持久存储**：用于保存下载的文献全文（PDF）和向量化后的知识库，以及日志等。

该架构的核心是**黑板模式**：各 Agent 仿佛多位专家在公共黑板上协作。初始时用户问题被写上黑板，关键词生成专家看到问题后写出关键词列表；检索专家发现关键词后执行文献搜索，将结果列表写回黑板；解析专家读取结果列表下载论文并提取信息，依次类推，逐步丰富黑板内容直至问题得到全面解决。

数据流与模块分工

以下描述从用户提出科研问题到获得文献分析结果的数据流各步骤，以及负责的 Agent 模块：

1. **科研问题输入**：用户在前端输入研究课题或问题描述。中央控制器将其记录在黑板，生成唯一查询 ID，并创建初始任务项（例如 *生成检索关键词*）。
2. **关键词生成 Agent**：监听到新问题任务后，使用 NLP 技术分析问题内容，自动生成一组相关**检索关键词**。方法可包括：
    - 基于预训练语言模型或词向量找出主题相关术语和同义词；
    - 或利用问答模型提取问题中的重要名词、实体和主题词。

    生成的关键词列表（含主要关键词及扩展词）写入黑板。随后触发下一步检索任务。
3. **文献检索 Agent**：读取黑板中的关键词列表，通过集成外部学术搜索 API 执行**文献检索**。支持的检索方式：
    - 使用 **SerpApi** 的 Google Scholar API，以关键词查询学术文献。SerpApi 返回的 JSON 结果包含文章标题、链接、作者、引用数等信息。利用这些结构化结果，可直接获取每篇文献的题名、作者列表、出处（期刊/会议）、发表年份以及**被引用次数**等。

- 平台支持其他检索源：如调用 Semantic Scholar 或 CrossRef API 获取元数据（标题、摘要、作者等），以增强信息完整性（可选）。
- Sci-Hub 集成：对于有 DOI 的结果，可通过 Sci-Hub 获取直链或 PDF。如果 SerpApi 返回结果包含直链 PDF（如所示部分结果已提供 PDF 链接），则可直接下载；否则利用 DOI 构造 Sci-Hub 查询下载全文。

  注：Sci-Hub 无官方 API，可借助开源工具（如 **scihub.py**）通过 DOI/URL 自动获取论文 PDF。

4. **结果列表存储**：检索 Agent 将获取到的**候选文献列表**写入黑板，包括每篇的结构化条目：标题、作者、来源、年份、引用次数、链接（及 PDF 链接若有）等。黑板同时更新检索状态（成功/失败数，若失败则记录错误原因）。
5. **元数据提取 Agent**：该 Agent 遍历黑板中的检索结果条目，对每篇文献执行**元信息提取**：
   - **摘要获取**：优先使用可用资源获取摘要：例如通过调用语义学平台 API（Semantic Scholar 提供摘要和关键词）或 CrossRef 查询摘要；若不可得且有权限，考虑解析 PDF 文本抽取摘要段落。也可采用 PDF 解析工具（如 GROBID 或 PyMuPDF）定位 "Abstract" 段落提取内容。
   - **详细信息**：确保每条文献条目补充完整信息字段：题目、作者列表、作者单位（如可获取）、期刊或会议名、出版年月、引用次数（已有）、关键词（如果 API 返回或可由摘要自动提取）。此外，记录数字对象唯一标识（DOI）用于定位。

     该 Agent 将完整结构化的文献信息对象更新写入黑板，与初始简要列表区分标记。这样黑板就累积了检索到的**文献元数据集合**。
6. **分析任务生成**：中央控制器根据用户需求或预设，生成一系列**分析任务**：

- 若用户未指定，默认至少执行**基础提取**（即完成上述元数据提取和摘要获取）。
- 根据用户在界面勾选的“分析深度”选项，可创建以下任务：**方法与结论抽取**、**内容摘要**、**主题聚类分析**、**引用链分析** 等。

  每项分析任务作为新的任务对象附加到黑板相应条目或全局任务列表中。

7. **深度分析 Agents**：针对不同分析任务，有专门 Agent 执行：
   - **方法与结论抽取 Agent**：读取每篇文献的全文或摘要，利用模式匹配或 NLP 模型定位论文中的方法章节和结论章节。比如通过正则/关键字搜索章节标题（如“Method”，“方法”，“Conclusion”，“结论”等）来截取文本；或者应用段落分类模型将全文段落归类，提取其中的方法描述和主要结论陈述。这些信息提取后附加到对应文献条目下的结构化字段中（如 methods, conclusions 字段）。
   - **文本摘要 Agent**：针对每篇文献，生成**简明摘要**（区别于作者原摘要）。可使用预训练文本摘要模型（如 BART、T5）或 GPT 类大模型，对全文进行概括，提取核心论点和发现。考虑篇幅较长时可先分段总结再综合。生成的简明摘要存入黑板文献条目下，比如字段 summary_ai。
   - **主题聚类 Agent**：对**全部候选文献集**进行主题分析。具体方法：对每篇文献的标题或摘要计算向量表示（利用选定的嵌入模型，如 SciBERT 或 BGE），然后采用无监督聚类（如 K-Means 或 HDBSCAN）在向量空间将文献分组。生成若干主题簇，每簇给出代表性关键词或描述（可基于该簇中心文献的高频词）。该 Agent 将**聚类结果**（每个主题对应文献列表及主题标签）写入黑板全局区，以供前端展示主题板块。

- **引用链分析 Agent**：构建文献之间的**引用网络**。基于每篇文献的引用情况，分析它们的相互关系与影响力链条。例如：
  - 查询每篇文献的参考文献列表（可从论文 PDF 末尾解析 References 部分，或借助语义学数据库获取引用关系）；
  - 交叉比对候选文献集中是否有互相引用关系，或某些共同被引的重要文献。据此构建一个**引用图**，节点为论文，边表示引用。可以进一步计算每篇的被引次数（已在元数据）以及网络中介度等，以发现**关键文献**（如高被引的核心论文）。

    分析结果以**引用分析报告**形式写入黑板，包括关键文献列表、引用网络简述等。

8. **结果集成与黑板更新**：随着各分析 Agent 完成任务，黑板上文献信息被逐步丰富和标注。中央控制器跟踪任务状态，当所有选定分析完成后，将黑板中的最终结构化结果集标记为**完成**状态。黑板此时汇集的数据包括：
   - 候选文献列表及每篇的详细信息（元数据、摘要、方法、结论、AI 摘要等）。
   - 全局分析结果（主题簇、引用网络）。
   - 日志和错误信息（如某些文献下载失败或解析失败的记录）。
9. **前端同步与展示**：黑板系统与前端通过轮询或推送机制同步数据。前端界面根据黑板内容实时更新：
   - 以列表或表格形式显示候选文献条目，可按引用次数或相关度排序。
   - 展示每篇文献的摘要等详情，供用户浏览。

- 展示整体分析模块：主题聚类结果（分组展示）、引用网络概要（可视化图或列表）。

  用户可审阅这些结果，并有权对文献标注关键性或者添加遗漏的文献。

Agent 接口设计

各 Agent 模块需要定义清晰的接口，以实现与黑板的交互和与其他 Agent 的解耦。接口包括：

- **触发条件**：Agent 需声明其监听的黑板事件或数据模式。例如，关键词生成 Agent 监听“新问题任务”事件；检索 Agent 监听“关键词列表已生成”事件；解析 Agent 监听“新文献列表项”事件；RAG 构建 Agent 监听“关键文献标记”事件，等等。
- **输入数据格式**：规定 Agent 从黑板读取的数据结构。例如，检索 Agent 从黑板读取关键词列表（数组或逗号分隔字符串）；解析 Agent 读取文献基本信息（包含链接或 DOI）；摘要 Agent 读取全文文本或段落列表等。
- **输出数据格式**：Agent 将结果写回黑板的结构定义。比如，元数据提取 Agent 输出：{文献 ID: {title: ..., authors: [...], abstract: ..., journal: ..., year: ..., citations: ..., keywords: [...]}}。方法抽取 Agent 输出：在对应文献条目下新增 methods 和 conclusions 字段（文本或要点列表）。各 Agent 输出需附带任务状态标记（完成/失败）及时间戳。
- **错误信息**：若任务执行失败，Agent 应写入错误日志条目至黑板，包括关联的文献或任务 ID、错误类型（异常、超时等）、错误描述，供系统记录和 UI 提示。

通过统一接口，保证各 Agent 实现可以替换或扩展而不影响整体协作。例如，可日后增加“作者网络分析 Agent”只需监听黑板上文献作者信息并生成关系图，不干扰其他部分。

黑板系统与前端交互

黑板系统在此方案中既是数据存储也是交互中枢。设计要点：

- **数据存储与访问**：可采用内存型数据库（如 Redis）或文档数据库（如 MongoDB）实现黑板。需支持根据任务/文献 ID 高效读取和更新指定条目。黑板数据可以按照查询会话隔离（不同用户查询不同命名空间）。
- **发布/订阅机制**：实现 Agent 对黑板事件的监听。例如基于 Redis 的 Pub/Sub，或在应用层实现事件总线。这样，当黑板某数据路径更新时，会通知订阅该路径的 Agent 执行。
- **并发与锁**：黑板须考虑并发控制，避免多个 Agent 同时写入冲突。同一文献条目可通过细粒度锁或原子操作更新，如使用 MongoDB 原子更新文档字段或 Redis 事务 pipeline。

前端界面通过调用后端接口访问黑板数据，实现**同步展示**：

- **结果呈现**：前端定期请求或通过 WebSocket 推送黑板中的更新内容。数据以结构化形式（JSON）传输，前端解析后渲染各模块：
    - 文献列表展示：标题、作者、期刊、年份、引用次数等直接可见，并提供展开按钮查看摘要、方法、结论等详细字段。
    - 分析模块展示：主题聚类以分组列表或彩色标签呈现，每组下列出所属文献标题。引用分析可选用网络图（节点连边）或列表形式，突出高引用论文及互引关系。
- **用户标注交互**：用户在前端可将某些文献标记为**“关键文献”**（例如通过复选框或星标）。前端调用后端接口，将该标记写入黑板对应文献条目属性（如 is_key: true）。这会触发后续 RAG 构建 Agent 操作。
- **用户补充**：如果用户发现检索结果中遗漏了重要论文，界面提供**手动补充**功能：用户可输入论文 DOI 或直接上传 PDF。前端将此信息发送给后台一个专门接口，后台解析后

新增文献条目到黑板，并标记为用户补充。之后普通解析 Agent 即可对其提取信息，纳入分析。

前端和黑板的交互需保证**实时性**和**数据一致性**。推荐在后台维护文献列表和分析结果的**快照**，每次黑板更新后推送增量更新至前端，使用户能即时看到进度。例如当检索开始时前端显示“检索中...”，检索 Agent 写入若干结果后前端立即展示相应文献条目等等，提升交互体验。

分析深度模式支持

系统支持按需配置不同**分析深度**，以适应用户不同阶段需求。主要模式有：

- **基础模式**：只执行**元数据/摘要提取**。结果包括文献信息的基本字段和摘要，适合获取快速综述。
- **深入模式**：在基础上增加**方法与结论提炼**。让用户了解每篇论文的研究方法和主要结论，有助于技术比对和结果比较。
- **综合总结模式**：进一步对每篇论文生成 **AI 摘要**（更精炼的描述）以及**主题聚类分析**整体领域热点。适合用户快速掌握整体研究脉络。
- **全面分析模式**：包含上述所有，并增加**引用链分析**或其他高级分析（例如作者合作网络、年度趋势分析等）。提供最丰富的信息，利于全面的文献调研报告。

这些模式可以通过在前端提供选项，由用户选择所需分析层次。后台中央控制器据此决定触发哪些 Agent 执行。默认可选择“基础”到“全面”四档；也允许用户自定义（例如只要聚类、不需要引用分析）。这种设计保证系统灵活性，同时节省不必要的计算开销：如用户只关心摘要就无需运行聚类等耗时任务。

各模式的结果均存储在黑板，前端依据模式呈现不同板块。用户随时可调整需求触发补充分析，例如先基础后发现需要更深入，则可再次触发剩余分析 Agent，对已有结果进行补充（无需重复检索步骤）。

关键文献的 RAG 知识库构建

当用户将某些文献标记为**“关键文献”**后，系统将对这些文献构建** RAG（检索增强生成）知识库**，供其他智能 Agent 或 QA 模块使用。具体流程：

1. **全文获取**：RAG 构建 Agent 监听到黑板上文献条目 is_key=true 的标记后，首先确保获取该文献**全文文本**。如果先前未下载 PDF，则通过 Sci-Hub 或其他链接下载。使用 PDF 解析工具提取文字内容（可能需要 OCR 处理扫描版，但大部分论文 PDF 可提取文本）。
2. **文档分段**：将全文按**章节或语义段落**划分为若干段落（例如每≈200-500 字一段，避免段过长影响向量表示效果）。可结合论文的结构（如基于章节标题、段落自然段）进行拆分，并为每段记录其来源文献 ID 及段落位置。
3. **文本向量化**：使用选定的**嵌入模型**将每个段落编码为稠密向量表示。这一步选择合适的模型非常关键，下节将详细评估。向量化时注意：
    - 中文或多语言内容支持：如选用多语言嵌入模型或分别对中英文使用对应模型。
    - 归一化处理：很多检索向量需要单位化，以使用内积/余弦相似度。
    - 批量计算优化：若关键文献较多或篇幅很长，可批量送入模型编码，加速处理。
4. **向量数据库存储**：选择合适的**向量数据库**建立知识库，将每段的向量及其元数据存储为一条向量记录。元数据至少包括：文献 ID、段落 ID、可能还存储原文本或摘要以便后续呈现。存储时可指定索引类型（如 HNSW）和度量（余弦相似度）。
5. **索引构建**：插入数据后，向量库对新向量集合建立索引结构以支持高效相似检索。不同数据库可能自动完成或需要调用构建索引操作。
6. **知识库标记完成**：黑板上将该文献的状态更新为已索引到 RAG 库，并在全局记录中添加**知识库引用**。系统其他 Agent（如解答 Agent）通过调用向量数据库 API 即可按需

检索相关段落，实现**检索增强**。在 RAG 模式下，当有新的自然语言问题咨询系统时，可以：

- 将问题也通过相同嵌入模型编码为查询向量；
- 在向量数据库中检索 top-k 最相关段落；
- 将检索到的内容连同原问题提交给大语言模型来生成回答。

  这样 LLM 会参考知识库中的权威内容，有依据地回答科研问题。

RAG 知识库构建主要服务于**后续的智能问答或总结 Agent**。例如用户可能在完成文献调研后，向系统提问“这些关键文献中有哪些主要共识和分歧？”。QA Agent 就可以检索知识库相关段落，然后结合 LLM 生成答案并引用出处。这种架构提高了回答的专业性和可信度。

嵌入模型与向量数据库选型

RAG 模块的效果取决于**文本嵌入模型**和**向量数据库**的选择。以下对几种候选方案进行评估，并结合科研文献场景提出推荐。

文本嵌入模型评估

1. **BGE (BAAI General Embedding)**：由北京智源研究院推出的系列 embedding 模型，专为**密集检索**优化。提供中英文模型（如 *bge-large-zh-v1.5* 和 *bge-large-en-v1.5*），对中文支持良好。特点：
   - **检索性能**：在中英多语言检索基准上达到 SOTA 效果。据官方报告，在多语言检索基准 MIRACL 和 MKQA 上表现优异。C-MTEB(中文集)平均精度达 64 以上（bge-large-zh)。
   - **多语言**：FlagEmbedding 模型支持 100 多种语言并具备跨语言检索能力。适合处理中文文献同时兼顾英文资料的场景。

- **模型规模**：有 large/base/small 不同规模。Large 版输出 1024 维向量，精度高但推理资源占用大；Base 版 768 维，性能略低一点但更轻量。可按硬件选择。
- **长上下文**：支持输入长达 512 甚至 8192 tokens（部分模型），适于论文长文本。

  推荐：若系统重点面向中文或双语科研文献检索，BGE 是**首选**。可使用 base 或 large 模型视算力决定。开源 Apache2.0 许可，易于集成 HuggingFace 或 FlagEmbedding 库使用。

2. **SciBERT**：Allen AI 发布的针对科学文献语料预训练的 BERT 模型。特点：
   - **领域优化**：在科学英语文本任务上明显优于通用 BERT。擅长科学术语理解、关键词抽取、分类等。对**英文**论文的语义表示质量较高。
   - **局限**：SciBERT 自带词汇表主要是英文科学词，**中文支持较弱**（有社区衍生的 SciBERT-CN 针对中文计算机文献，但相对 BGE 中文效果未知）。没有针对句嵌入专门优化，需要 fine-tune 用于检索任务才能发挥作用。
   - **模型规模**：Base-size BERT(≈110M 参数)，向量维度 768。适中资源需求。

     建议：如果文献主要为英文且希望利用科学文本预训练优势，可考虑 SciBERT 微调为句向量后使用。但对于包含大量中文或需要最高检索性能的场景，SciBERT 可能不及新近的 BGE 等模型。可作为补充选项。
3. **MiniLM 多语言版**：MiniLM 是 Microsoft 推出的小型 Transformer，常用于**句向量**（如 SentenceTransformers 提供的 *all-MiniLM-L6-v2* 仅 6 层、384 维）。多语言版本（e.g. *paraphrase-multilingual-MiniLM*）能处理中文和英文。
   - **优点**：模型小、推理快，内存占用低，适合资源受限环境和大批量向量化需求。部署难度低（可以 CPU 快速跑）。

- **性能**：由于参数少，捕捉语义细节能力逊于大型模型。对于精细的学术语义匹配，准确率较中大型模型低一个档次。不过在通用语义检索上仍表现尚可，胜在速度。**建议**：MiniLM 适合**性能要求不高且重效率**的场景，比如系统初期验证或设备算力较弱时采用。在精度关键的 RAG 场景下，MiniLM 可作为降级方案，但主要推荐还是使用 BGE 这类更强模型以确保检索相关性。

4. **其他模型**：如 **MPNet, RoBERTa** 多语言版、**SimCSE** 等在嵌入任务也有应用。若开源社区有专为中文/学术优化的对比学习句向量模型（如 **Chinese-PatentBERT** 等），也可评估。不过上述提及模型已涵盖主流选择。

综合考虑，**推荐方案**是在硬件允许情况下，采用 **BGE 模型**系列构建嵌入。BGE-large-zh 与 BGE-large-en 分别用于中文和英文文献嵌入（或使用跨语言模型统一向量空间）。BGE 提供现成 SDK，可与向量数据库如 Milvus 集成。其 Apache 开源许可证和良好社区支持也利于长期维护。SciBERT 可作为英文语料补充，用于验证 BGE 对科学术语的覆盖。MiniLM 等可作为备用选项以提升速度或在边缘场景运行。

向量数据库评估

文献知识库需持久存储和快速相似检索嵌入向量，以下比较几款主流向量数据库在科研调研系统中的适配性：

1. **Milvus**：开源分布式向量数据库，由 Zilliz 主导开发。
   - **性能与规模**：针对十亿级向量搜索设计，可扩展分布式部署。支持多种索引（IVF, HNSW, ANNOY 等）和 GPU 加速，检索精度高且性能强劲，在大量数据下表现出色。
   - **功能**：支持按属性过滤、分区检索、批量查询等高级功能。Milvus 2.x 引入多向量检索机制，可对实体的多个嵌入执行混合搜索。

- **部署难度**：完整分布式依赖 etcd、RockDB 等，有一定部署复杂度。但也提供轻量 docker 容器。社区活跃、文档完备，有中文社区支持（Milvus 源自国内，交流方便）。
- **中文支持**：对中文没有特殊限制，本质存储向量。但 Milvus 不自带分词，需要上层处理。如需关键词模糊检索，可结合 Elastic 等。

  推荐使用情境：当系统需要**大规模文献库**（几十万以上段落）或者计划未来扩展数据量，Milvus 是稳妥之选，以**性能最强、功能最丰富**见长。开发者可利用其 Python SDK (PyMilvus) 快速集成，尤其适合已有 Milvus 生态（如 Zilliz 云服务）的团队。

2. **Qdrant**：新晋开源向量库，Rust 实现。
   - **资源占用**：非常轻量，内存与 CPU 效率高。适合在有限资源服务器或边缘设备上运行。支持持久化存储，内置 HNSW 索引。
   - **功能**：提供 **Payload** 机制，可存储文档的元数据并按条件过滤查询。支持简单的全文检索过滤和前缀匹配，但不支持复杂 BM25 排序（注：Qdrant 专注向量语义，不内置传统倒排检索功能）。
   - **集成**：REST API 和客户端多语言 SDK 易用，亦兼容 LangChain 等工具。支持集合级别多租户隔离。开源协议 Apache，社区也较活跃。
   - **中文支持**：可存储中文文本为 payload，无特殊问题。只是在文字过滤时需注意中文分词需自行处理成前缀或 WholeWord 匹配。

     推荐使用情境：**中小规模**知识库或注重**部署简便**的场景。Qdrant 可非常快速上手并运行，维护成本低。如果预计文献库在百万级以内，Qdrant 完全可以胜任且表现优异。其**低开销**也利于开发测试和迭代。

3. **Weaviate**：以 Go 语言实现的开源向量 DB，提供丰富功能。
    - **混合检索**：内置纯向量检索、BM25 关键词检索以及二者结合的 **Hybrid 检索**能力。对于需要同时支持语义和关键词查询的应用，Weaviate 相当便利。
    - **内置模块**：可选内置 text embedding 插件（FastText 等）和跨模态支持。GraphQL 查询接口强大，可灵活制定条件和返回内容。亦提供 RESTful 和客户端。
    - **部署**：有官方 SaaS，也可自托管。自托管需要 Docker 容器运行，使用起来尚算简单，但内存占用稍高。支持分片扩展和备份。
    - **中文支持**：Weaviate 的 BM25 检索需要分词器，默认英文，对于中文需要接入分词插件或通过倒排索引模式细调。

      推荐使用情境：当需要**复杂查询和混合搜索**时，Weaviate 是不错选择。例如用户既想按研究方向关键词筛选，又要语义相关的内容。Weaviate 支持**组合过滤**和**聚合分析**。如果团队熟悉 GraphQL，它的查询语言将非常高效。缺点是资源消耗较 Qdrant 略大，且中文分词需配置，但对于大型系统可以接受。
4. **FAISS**：Facebook 开发的向量相似库（C++/Python）。
    - **特点**：作为库而非服务，它嵌入在应用内部使用，无网络开销。支持多种 ANN 算法，高度可定制。适合小规模数据做内嵌检索，延迟极低。
    - **不足**：**不支持多客户端并发**和持续增删。一般用于静态数据离线构建索引，然后查询。对本方案黑板架构下多 Agent 访问不太友好，且缺乏在线扩展能力。

      建议：FAISS 可用于**原型阶段**快速实验或存储少量向量，但作为完整系统后端不够便利。可以在测试 embedding 效果时用 FAISS 做 benchmark，然后再迁移至上面的向量数据库。

5. **其他**：如 **Pinecone**（闭源商用，优势是免维护但需付费，不详述）、**ElasticSearch 向量插件**（能将语义与关键词统一索引，但性能不及专业向量库），**Vespa**（Yahoo 开源，强大但复杂）等。综合考虑开源、自主可控和社区支持，以上 Milvus/Qdrant/Weaviate 已是首选。

推荐方案：结合科研文献调研的规模和需求，**首选**考虑 **Milvus 或 Qdrant**。如果预期文献数据量逐渐增大且需要企业级扩展，Milvus 以**性能和完整特性**取胜；而在初始阶段或中等规模下，Qdrant 以**简洁高效**更为合适。具体取舍可以是：

- **方案 A**：采用 **Milvus** 自托管，配合 PyMilvus SDK 和 BGE 嵌入，无缝构建超大规模向量库。同时由于 Milvus 原生支持中文社区，遇到问题可获得本地支持。
- **方案 B**：采用 **Qdrant**，轻量部署在后端服务中，通过 REST API 读写。Qdrant 对资源要求低，方便在现有 Web 服务中集成，其**低延迟**有助于提供实时交互体验。
- **方案 C**：若强调混合搜索需求，考虑 **Weaviate**，让用户可用中文关键词或语义任意组合筛选文档。这需要一定 GraphQL 学习成本，但能带来灵活交互查询功能。

无论哪种数据库，嵌入模型的向量需 **Normalize** 以使用余弦相似度比较，这些数据库均支持计算 inner product 或欧氏距离来等价实现。通过实际测试各库在本场景的查询延迟和准确率，可进一步验证选择。例如 Qdrant 和 Milvus 都宣称高性能，不妨针对万级向量做 Benchmark，比较平均搜索耗时和资源占用，以最终确定。

工具链与 API 集成建议

实现上述功能，需要集成多种第三方工具和 API。以下列出关键环节的选型和集成方案：

- **自然语言处理**：关键词生成、摘要提取等，可使用 **Hugging Face Transformers** 库加载所需模型（如 T5、BERT 等）。对于中文支持，可选用哈工大 LTP 或 BERT-wwm 模型来协助分词和实体提取。

- **SerpApi Scholar API**：使用 SerpApi 提供的 Google Scholar 接口检索文献。需注册 API Key，调用时构造查询参数（引擎=google_scholar，q=关键词等），得到返回 JSON 解析出 articles 列表。注意调用频率和配额（考虑缓存已检索过的 query 结果以减少消耗）。若不使用 SerpApi，也可采用 **Scholarly** Python 库(非官方抓取 Scholar)作为替代，但可靠性略差。
- **Sci-Hub 获取 PDF**：集成 Sci-Hub 可参考开源实现。如利用 scihub.py：通过 sh.download(doi_url) 即可自动解析获取 PDF。需要处理 Sci-Hub 可能出现的验证码或失败重试机制。也可设置多个 Sci-Hub 镜像备用（如 .org.cn 等）。
- **PDF 解析**：优先使用结构化解析工具 **GROBID**（可容器化部署）提取 PDF 的标题、摘要、作者等结构化字段，这在元数据提取 Agent 可以用到。若 GROBID 部署复杂，可退而用 **PyMuPDF** 直接按页面提取文本，然后自行定位摘要段。参考 Reddit 经验，GROBID 对批量论文解析效果较好。解析后文本需简单清洗（去除换页符等）。
- **NLP 信息抽取**：方法和结论部分抽取，可针对论文全文训练一个**段落分类模型**（Transformer+分类头)来识别段落属于哪部分，从而定位“方法论”段落开头和“结论”段落。这需要一定标注数据。如果没有，可采取基于规则的粗略方法 (搜索关键字)先实现。
- **文本摘要**：可选用开源**文本摘要模型**，如 Facebook BART 大型模型（需 GPU）、或者 T5-small 等。如需中文支持，百度的 **ERNIE-doc** 或微软 ProphetNet 中文等也可考虑。另一思路是调用在线 API 如 OpenAI GPT-4/GPT-3.5 生成摘要，但这涉及付费与隐私问题，除非用户许可及预算足够。开发阶段建议先用开源模型跑本地。
- **Embedding 计算**：使用 HuggingFace Transformers 或 SentenceTransformers 接口加载如 BGE 模型进行编码。可封装一个 **Embedding Service**，支持批量向量化

API，供 RAG Agent 或其他模块调用，以充分利用 GPU 资源（如一次对 N 段文本编码）。

- **向量数据库**：使用官方 SDK 集成：
  - Milvus: PyMilvus 已支持直接调用 HuggingFace 模型生成 embedding 并插入，但更灵活的是先用模型算好 embedding 再用 SDK 写入。Milvus 需要先创建 collection（指定维度、索引类型等），再多次 insert 向量，最后 load collection 以供搜索。
  - Qdrant: 提供 Python 客户端(qdrant-client)和 RESTful 接口。可以直接使用 REST API /collections/{col}/points 上传向量及 payload，或用 client 高层方法。需要选择 HNSW 参数（默认 ef/search 等）。初始部署可用 Docker 启动 Qdrant 服务。
  - Weaviate: 提供 Python 客户端，先定义 Schema（类、属性，其中向量作为特征），可以使用 Batch 导入向量。其 Hybrid 检索可用 GraphQL query 指定 bm25 等。
- **前端开发**：可选用 Web 框架开发交互界面，如 React/Vue 结合 Ant Design 等组件库展示列表和图表。引用网络可以用 D3.js 或 ECharts 实现交互图谱。前端通过 REST API 获取数据，也可使用 WebSocket 订阅后台事件推送。需规划 API 路由，如：
  - POST /query 提交新科研问题，返回 query_id；
  - GET /query/{id}/results 获取黑板上该查询的最新结果数据；
  - POST /query/{id}/mark_key 标记关键文献 ID 列表；

    - POST /query/{id}/add_paper 用户补充文献（通过 DOI 或文件）。后台相应路由处理，操控黑板并触发 Agents。
- **日志与监控**：引入日志框架（Python 的 logging 或 ELK 栈）记录各模块活动和错误。对关键步骤埋点监控性能（比如检索耗时、向量化耗时、查询响应时间），为优化提供依据。

整个工具链应当在**容器化**环境下搭建，方便部署与扩展。各 Agent 可以是独立服务（如用 Celery 队列调度任务），也可以是多线程模块。考虑到 Python 生态对 AI 友好，Agent 主体可用 Python 实现。向量 DB、GROBID 可通过 Docker Compose 编排。确保在开发和部署文档中详细记录各依赖版本和配置方法，使团队易于搭建一致的环境。

可扩展性设计

为保证系统具备良好的扩展性，设计时采取了**模块化松耦合**和**可配置**的策略：

- **新增 Agent**：采用黑板架构使加入新 Agent 变得容易。例如未来想增加“学术趋势分析 Agent”（统计文献年份分布、热门机构等），只需订阅黑板上文献信息，执行分析后写入结果，不需改动现有模块逻辑。黑板作为中介屏蔽了模块之间的直接依赖。
- **模型替换**：嵌入模型、摘要模型等均通过配置加载，支持替换升级。如有更先进的中文 embedding 出现，可以直接在配置中指向新模型路径，输出维度变化时在向量库中新增字段即可兼容。向量数据库也可根据需要更换，实现层应封装检索操作细节，使业务逻辑不依赖具体 DB 实现。
- **横向扩展**：对于高并发或大数据需求，各 Agent 服务可独立扩展实例数。例如检索 Agent 可多进程/多机部署，提高并行检索吞吐；向量库支持集群扩容更多节点。黑板可用集中式缓存（Redis Cluster）或数据库（如 Mongo 副本集）扩展容量和性能。整

个系统可按需要演化为微服务架构，每个 Agent 作为微服务，通过消息队列通信以实现大规模分布式处理。

- **配置驱动**：大量参数（如检索关键词数量、每步最大并行数、各分析任务开关、embedding 模型名、DB 连接等）都放在配置文件或环境变量中，使系统行为可调整而无需改代码。这样不同项目或不同阶段（开发/生产）可以使用不同配置满足需求。

错误处理与容错机制

在文献调研任务中，不可避免会遇到各种异常情况。方案中设计了系统性的**错误处理**和**用户反馈**机制：

- **检索失败重试**：如果 SerpApi 调用失败（网络或配额问题），系统应捕获异常，记录到黑板日志，并可尝试启用备用策略：如使用开源爬虫库抓取谷歌学术网页，或者改用关键词在必应学术等引擎检索。重试可设定最大次数，避免无限等待。
- **下载失败**：Sci-Hub 下载论文可能失败（文章不存在或网络问题）。Agent 应在黑板该文献条目添加标记如 pdf_status: "failed" 并写明原因。如果重要性高（例如用户标记关键却下载不了），可以提示用户手动提供 PDF。系统可在前端显示“[下载失败，点击重试或手动上传]”的选项。
- **解析错误**：PDF 解析或信息提取可能遇到格式异常。Agent 捕获后写入 parse_error 字段（如无法定位摘要）。对于摘要缺失的，可以考虑后续用 AI 模型直接生成摘要作为补救。
- **超时处理**：为防止某些 Agent 长时间卡顿（例如外部 API 无响应），对各主要步骤设置**超时**。比如检索 Agent 单次 API 调用限时 10 秒，超时则返回错误。Agent 应该能被中断或取消，防止阻塞后续流程。控制器可监视任务持续时间，超时则标记任务失败并通知相关 Agent 停止。

- **不一致检查**：引用分析 Agent 需校验文献 ID 匹配，避免由于解析错误引入的误连接。如果发现某些引用指向不在候选列表的文献，可忽略或尝试查询补全。
- **用户通知**：所有错误情况通过前端 UI 以非阻断的方式提醒用户。例如在对应文献项显示红色警告符号，hover 显示错误详情（如“摘要提取失败，内容可能不全”）。对于整体流程的问题（如检索无结果），在结果区域给出明显提示“未检索到相关文献，请尝试更换关键词”。引导用户调整输入而非陷入无反馈状态。
- **日志追踪**：系统维护内部日志文件/数据库，包括每次查询任务各步骤的执行日志和错误栈。管理员可通过日志定位问题来源（比如某 PDF 解析函数反复异常）。同时记录关键统计数据如检索成功率、平均用时等，长期监控以改善系统稳定性。

通过上述机制，系统能够在面对异常时**平稳降级**：即使部分论文无法获取全文，至少保证元数据和其他文献的分析正常完成，不让一次错误中断整个任务。对于严重故障（如黑板数据库连接失败），系统会将错误传递到前端让用户知悉，同时自动触发重启或管理员报警，尽快恢复服务。

(B)爬虫方案

模块划分

- **用户界面模块（前端）**：提供 Web 界面供用户输入科研问题，并实时展示分析结果列表。支持对文献信息的标签标注、标记关键文献以及评论修改等交互操作。
- **代理控制模块 (后端 Agent 服务)**：接收前端请求，负责 orchestrate 整个调研流程。包含关键词生成、任务调度、频率控制等子功能，将请求分发给浏览器自动化和信息提取模块执行。
- **浏览器自动化模块**：通过 Playwright 或 Selenium 控制 Chrome 浏览器（可在校园网环境下运行），自动访问 Google Scholar 等学术搜索引擎执行关键词搜索。模拟用户点击搜索结果中的原文链接（如 Springer、IEEE Xplore、Elsevier 等），获取目标文献页面的 HTML 内容。模块内嵌访问频率控制和异常处理机制，防止触发反爬虫。
- **信息提取模块**：解析学术文献网页内容，提取文献元数据和关键信息。包括标题、作者列表、摘要、期刊/会议名称、出版年份、引用次数等。针对常见出版平台采用模板匹配提取（利用已有的 HTML 结构经验），对于未知结构可采用可学习解析器（如基于 HTML 标记的机器学习模型）提取核心内容。优先从页面中的标准元数据标签提取信息（如<meta name="citation_title">、<meta name="citation_author">等），否则退而分析页面结构（如标题通常在<h1>标签，摘要段落标记有 Abstract 等）。
- **分析与标注模块**：对提取的文献信息进行初步分析和整理。可按与用户查询的相关度或引用影响力对结果排序（例如基于 Google Scholar 提供的引用次数）。支持将用户标记结果（标签、关键文献标记、评论)与对应文献条目关联存储。此模块管理文献条目的状态，例如哪些被标记为“关键”。

- **黑板同步模块**：充当 Agent 与“黑板”系统之间的接口，将分析得到的文献信息条目写入黑板知识库，并监听用户在黑板界面上的操作。黑板系统作为中心数据存储，保存所有文献信息及其标签/标记状态，供多人查看和协同编辑。该模块通过 API 将新的文献条目、用户评论等同步更新到黑板数据库中，实现前后端数据一致。

工具链选型

- **浏览器自动化**：推荐采用 **Playwright**（支持更高级别并发控制和更丰富的防机器人特性）或 **Selenium** 结合 **Undetected ChromeDriver** 来控制 Chrome。Undetected ChromeDriver 可规避反爬检测（例如绕过 navigator.webdriver 标志）。浏览器运行在**无头模式**以提高效率，并可配置使用校园网代理或 VPN，以利用校园 IP 访问文献资源的权限。
- **后端技术栈**：使用 Python 的 **Flask/FastAPI** 构建 Web 服务接口，或使用 Node.js 的 **Express/Koa** 框架。Python 生态下可方便利用学术工具（如用于关键词提取的 NLTK 等）和成熟的 HTML 解析库（BeautifulSoup、lxml）。参考实现中，有学者选择 Python Flask 作为后端。
- **前端技术栈**：采用 **React/Vue/Angular** 构建单页 Web 应用，以实现友好的交互体验和实时更新。之前的类似系统中采用了 Angular 作为前端框架。前端通过 WebSocket 或 Server-Sent Events 与后端保持长连接，以流式接收分析结果的推送，或通过定时轮询 REST API 获取最新结果列表。
- **数据存储**：选用关系型数据库（如 MySQL/PostgreSQL）设计文献条目表，存储提取的元信息及标注；或文档型数据库（如 MongoDB）以 JSON 形式灵活存储文献详情和用户批注。数据库充当“黑板”知识库，支持按用户或项目查询文献集合。为了方便与前端交互，可构建一层简单的 ORM 数据模型映射文献条目。

- **解析与内容提取**：利用 **BeautifulSoup4** 或 **lxml** 解析网页 DOM，提取所需字段。针对各大出版商的平台，可维护一组解析模板（例如 IEEE Xplore 的摘要位于 div.abstract-text, Springer 文章标题在 h1.Title 等）。同时，可使用**正则表达式**或 XPath 辅助定位文本。对于结构差异较大的站点，引入**机器学习解析**思路：例如训练一个模型根据 HTML 标记和上下文词汇来识别标题、作者、摘要段落。
- **任务调度与并发**：在后端采用多线程或异步协程处理爬取任务。Playwright 本身支持异步操作，可并行打开多个标签页抓取多个文献页面。若使用 Python 同步代码，考虑使用**多线程/多进程**（通过 ThreadPool 或 Celery 分发任务）以提高 IO 并发。文献指出，多线程爬取 Google Scholar 多个结果页可显著加速响应。
- **代理与网络**：集成**代理 IP 池**支持。可使用第三方代理服务（如 Oxylabs、NetNut 等）提供的住宅 IP，以减小被封锁风险。工具链需支持动态切换 HTTP/HTTPS 代理（Playwright/Selenium 启动浏览器时设置代理服务器）。

信息流处理

图：文献调研 Agent 的信息流流程示意图。本方案通过浏览器自动化抓取 Google Scholar 搜索结果并解析文献信息，与参考文献收集系统的设计类似。

1. **查询输入**：用户在前端界面输入科研问题或关键词，提交查询请求至后端。系统可对自然语言问题进行简单的关键词提取或同义词扩展，生成适合学术搜索的查询串（例如提取主题中的名词关键词）。
2. **学术搜索**：后端 Agent 接收到查询后，调用浏览器自动化模块控制 Chrome 打开 Google Scholar 官网，并在搜索框中输入查询串模拟提交。系统设置适当的地域/语言参数，以确保检索结果最相关。

3. **结果列表获取**：Google Scholar 返回搜索结果页面，浏览器自动化模块解析结果列表 HTML，从中提取每条结果的基本信息和链接。包括论文标题、作者、发表源、年份、摘要片段、引用次数等。这些信息有助于后续筛选和排序。对于多页结果，系统可翻页抓取一定数量页面（可配置，如前 3 页）。
4. **原文页面抓取**：对于每条结果，浏览器模块模拟点击其标题链接，跳转到论文在出版商官网的详情页。采用异步并发或多线程提高效率——例如同时打开多个页面加载内容。在校园网环境下，若学校已订阅文献，则可直接获取页面正文和摘要；否则仅获取公开部分（题目、摘要等）。
5. **文献信息提取**：信息提取模块对每个页面进行解析，提取核心元信息字段。优先检查 HTML 的<meta>标签：多数学术平台遵循 Google Scholar 收录规范，提供标准的元数据标签，如 Highwire Press 的 citation_*系列。例如：
    - <meta name="citation_title" content="..."> 提供论文标题
    - <meta name="citation_author" content="..."> 提供作者
    - <meta name="citation_publication_date" content="..."> 提供出版日期等

    若存在这些标签，可直接读取其 content 属性。若缺少标准 meta 标签，则采用模板匹配：根据域名识别站点类型，使用预定义解析规则提取标题（如页面中的<h1>标签文本）、作者列表(可能位于带有class="authors"的内)、摘要正文(例如位于以“Abstract”开头的段落）。在解析过程中，过滤掉页面导航、参考文献列表等无关内容，只保留文献自身的信息。
6. **信息汇总与分析**：将提取的文献信息汇总成结构化数据对象(如 Python 字典或 JSON)。每条文献记录至少包含：标题、作者、来源（期刊/会议）、年份、摘要、引用次数、原文链接等。后端 Agent 可根据需要对结果进行分析排序：例如按引用次数降序，或根

据与用户问题关键词匹配度排序（可以简单统计摘要中关键词出现频率，或引入余弦相似度计算）。如果扩展高级功能，可引入预训练模型（如 BERT）计算论文与查询的语义相关度分数，以辅助排序。**注意**：大多数情况下不下载 PDF 全文，直接基于网页内容即可完成上述信息提取和相关度分析，从而降低开销并避免版权问题。

7. **结果实时展示**：当每篇文献信息提取完成后，后端立即通过黑板模块将该记录写入中央知识库，并推送增量更新到前端界面。用户界面接收到新结果后，即刻渲染展示，包括文献题目、作者、摘要摘要等关键信息。这样的流式处理使用户无需等待所有文献处理完毕，就能逐步看到分析结果。指出后端可以将处理结果以 JSON 格式提供给前端，客户端据此动态呈现列表。
8. **用户交互反馈**：用户在浏览文献信息列表时，可对每条目进行操作：添加**标签**（例如标注主题分类、研究方法等关键词）、将某文献标记为**关键文献**（关键程度高的重点参考文献），或撰写**评论/笔记**（例如记录与自己课题的关联）。这些操作通过前端组件发送请求给后端黑板同步模块。
9. **黑板系统同步**：黑板模块接收到用户的标注/评论指令后，更新中央知识库中相应文献条目的字段。例如，在数据表中将该文献的 is_key 标志置为 true，附加用户标签列表，或新增一条评论记录关联该文献。更新后的数据通过黑板 API 广播或供前端查询，从而在界面上即时反映用户的修改（例如高亮被标记为关键的文献，显示用户评论等）。黑板系统也支持多人协作，确保不同用户的评价标签都持久保存。同理，如果 Agent 后续对文献分析结果有修正（比如重新爬取得到更多信息），也通过黑板接口更新统一数据源。

上述流程形成一个闭环：用户提出问题，Agent 自动搜集整理文献信息，用户再对结果进行微调标注，最终在黑板知识库中沉淀结构化的调研信息。整个过程对用户而言高度自动化、交互友好。

访问频率控制与重试机制

为保障系统稳定运行并避免因过频访问被目标网站封禁，需要在各环节实施严格的频率控制和容错策略：

- **请求节奏控制**：对 Google Scholar 搜索请求设置频率上限，例如每分钟不超过一定次数。连续检索多页结果时，加入随机延时（如 1-3 秒随机停顿）以模拟人工翻页间隔，降低被识别为机器人流量的风险。对于访问学术平台文献页面的请求，也应串行化或小批并行，避免瞬时大量并发。可以实现一个中央计数器，跟踪在一定时间窗口内各域名的请求次数，超出阈值则强制等待。
- **随机代理与 IP 轮换**：集成代理 IP 池，在高并发或批量运行时定期更换出口 IP。优选住宅 IP 或校园网内不同出口，以分散流量来源。系统可在每次启动浏览器时随机选用代理服务器，或者每爬取若干篇文献后切换 IP。一些付费代理提供自动轮换功能和内置反爬虫对策，可考虑结合使用。通过代理轮换，避免所有请求都来自单一 IP，从而降低封禁概率。
- **模拟真实浏览行为**：在浏览器自动化操作中加入人性化行为，以绕过简易反自动化检测。例如设置 **User-Agent** 为常见浏览器标识，启用浏览器的 JS 渲染和加载图片（或至少部分加载）以模拟真实用户访问。可在页面加载后滚动浏览或短暂停留，再提取内容，仿真阅读行为。通过诸如此类细节，让爬虫流量更接近人类访问特征。
- **指数退避与重试**：当遇到请求失败或被拒绝（如 HTTP 503/429，或 Google Scholar 返回验证码页面）时，自动触发重试机制。采用指数退避算法逐步延长重试间隔，并更

换代理或切换 User-Agent 后再试。对于 Google Scholar 的验证码，系统可选择**通知管理员或用户进行人工干预**（在前端提示输入验证码），或集成第三方打码服务自动识别。当检测到 IP 被永久封禁时（持续返回拒绝响应），及时弃用该 IP 或暂停相关任务一段时间。

- **并发与队列**：实现一个**爬取任务队列**调度系统。对于多个用户同时发起的搜索请求，将任务排队，合理分配浏览器实例资源，防止同时过多实例竞争。可限定每次仅启动 N 个浏览器并行，其他任务等待，确保对单一站点的瞬时访问量在安全范围内。
- **监控与告警**：部署实时监控组件捕捉异常访问行为。例如记录每分钟请求量、各站点 HTTP 错误率。如发现某段时间错误激增（可能表明触发反爬机制），立即发出告警并自动降低爬取速率。监控 Google Scholar 返回的特殊标志（如出现 sorry 页面或 CAPTCHA）来判断是否被临时屏蔽。管理员界面应提供当前代理 IP 状态、请求统计等信息，便于手动调整策略。

通过以上机制，系统在保证效率的同时，最大限度降低了被封禁风险，做到“匀速且温和”地抓取目标数据。当运行环境变化（如学校 IP 被 Google 临时限制）时，也能凭借代理切换和退避策略平稳恢复，保障调研流程的持续性和稳定性。

数据结构设计

为支持文献信息的存储、展示和标注，设计合理的数据结构至关重要。以下是核心实体及结构说明：

- **文献条目（Literature Entry）**：代表一篇学术文献及其元数据。建议使用 JSON 对象或关系型表记录，包含字段：
    - id：唯一标识（主键），可使用自增 ID 或文献 DOI/URL 的哈希。
    - title：标题文本。

- authors：作者名单（字符串列表或逗号分隔字符串）。
- abstract：摘要正文（若可获取）。
- publication：出版源（期刊名称、会议名称或预印本平台等）。
- year：出版年份。
- venue_info：期刊卷期号、会议地点等补充信息（如有）。
- citation_count：引用次数（来自 Google Scholar 等）。
- url：原文链接 URL（指向出版社页面或 PDF 地址）。
- pdf_link：PDF 下载链接（如 Google Scholar 提供的直链，或自行解析获得的 PDF 地址，如 arXiv）。
- keywords：系统自动提取的关键词列表（可选，用于快速筛选）。
- query_id：所属搜索查询的 ID，关联到哪次用户提问（便于一个问题对应多个文献）。
- tags：用户添加的标签列表（字符串数组），初始为空。
- is_key：布尔值，标记是否被用户标为关键文献。
- comments：用户评论列表，每条包含评论内容、作者、时间戳等。

- **查询记录（Query）**：保存用户每次提出的问题及其上下文：
  - query_id：查询标识。
  - question：用户原始科研问题文本。
  - created_at：发起时间。
  - status：处理状态（进行中、已完成、失败等）。
  - 关联：该查询产生的文献条目列表（在关系型 DB 中可通过外键或关联表实现，一对多关联到文献条目）。

- **用户**（如果系统需支持多用户协同）：包括用户 ID、姓名/账户等，以及其权限（编辑、评论权限）。文献条目的评论应关联用户 ID。

在实现上，可将上述结构映射为关系数据库的多张表，如：Literature 表、Query 表、Literature_Tag（文献标签多对多关系）、Literature_Comment 表等。对于简单场景也可直接用 MongoDB 文档存储，一个文献条目文档嵌套其评论和标签。但关系型设计在多用户协作、复杂查询过滤（如按标签筛选文献）时更高效。

数据结构需支持**版本管理**和**纠错**：当 Agent 后续重新抓取或用户手动修改了某条目的元信息（例如更正错误的作者名），可以通过在文献表中增加 last_updated 时间戳或维护一个修改记录表，保存变更历史。

同时，为提高查询性能，可以对常用检索字段（如 title, authors, tags）建立索引，便于前端按关键词搜索过滤已有条目。黑板知识库作为中心数据源，应保证数据一致性和持久化，为此需设计唯一约束（如同一文献避免重复收录）以及定期备份机制。

前后端接口规范

为了实现前端与后端 Agent/黑板系统的解耦交互，需要定义清晰的 HTTP API 或 WebSocket 接口协议。下面提供一些接口规范建议：

- **检索接口**（POST /api/search）：接受用户提交的科研问题或关键词。请求体包含查询字符串以及可选参数（如每页结果数 page_size，获取页数 max_pages 等）。后端接收后立即返回任务创建成功的响应（包含 query_id），并异步开始处理。不建议同步等待所有结果再返回，以提升响应速度。
    - *Response*: { "query_id": 123, "status": "started" }（指示任务已开始，可用该 ID 追踪）。

- **结果获取接口**（GET /api/results/{query_id}）：用于前端轮询查询指定问题的最新文献列表。后端根据 query_id 在数据库查询已抓取的文献条目并返回 JSON 数组。支持传参 since_id 或 offset，以获取自上次以来的新增项，实现增量加载。也可提供 status 字段指示任务是否完成。
    - *Response*: { "query_id": 123, "status": "running", "results": [{文献条目 1}, {文献条目 2}, ...] }
- **实时推送**：为了更顺畅的实时体验，可选用 WebSocket 或 Server-Sent Events 在后端任务运行过程中推送消息。每当新文献解析完成，后端通过 WS 向客户端发送消息，例如：{ "event": "new_result", "data": {文献条目 JSON} }。前端接收后将新数据插入列表。还可以有 progress 事件通知（如已完成第 N 页爬取）。
- **标签标注接口**（POST /api/literature/{id}/tags）：用于添加或更新某文献的标签。请求体包含要添加的标签列表（或单个标签）。后端将新标签合并到该文献条目的 tags 字段，并写入黑板数据库。响应包含更新后的标签集合。删除标签也可复用此接口（通过提交当前完整标签列表或专门的 DELETE 接口）。
- **标记关键文献接口**（POST /api/literature/{id}/mark）：用于将某文献标记/取消标记为关键。请求体可为空或仅指示操作类型，或用同一接口的重复调用切换状态。后端据此将 is_key 字段置 True/False。响应返回操作结果和当前标记状态。
- **评论接口**（POST /api/literature/{id}/comment）：提交用户对该文献的评语或笔记。请求体包含评论内容（以及作者身份，由会话识别或一并提交）。后端在数据库中新建评论记录，关联该文献 ID 和用户 ID。响应返回包含评论 ID、timestamp 等确认信息。可能还需要 GET /api/literature/{id}/comments 用于获取某条目的评论列表，用于前端显示讨论。

- **黑板数据同步接口**：如果黑板系统是独立服务，应提供内部 API 给 Agent 写入数据。例如 POST /api/blackboard/entries 提交新的文献条目 JSON，PUT /api/blackboard/entries/{id}更新条目（添加标签、标记关键等），以及获取条目和查询的接口。不过在本方案中，黑板存储紧密集成在后端服务内，通过数据库操作直接实现，因此无需额外微服务接口。

接口返回格式统一采用 JSON，使用明确的状态码：例如任务提交成功返回 202 Accepted，查询结果返回 200，错误情况返回 4xx/5xx 及错误消息。前后端需要约定数据字段的含义和单位（如时间格式、文本编码 UTF-8 等），确保一致。

前端组件应对用户操作提供本地反馈（如标签添加成功高亮）并乐观更新，同时通过后台响应校准结果。所有接口应进行基础的参数校验和鉴权（如确保用户只能标注自己的项目文献），以保证系统安全。

自动化策略

实现文献调研 Agent 的自动化过程，需要平衡效率、准确性与系统资源，制定如下策略：

- **并行与异步**：充分利用并行能力加速调研过程。例如，Google Scholar 检索结果获取与文献页面解析可以解耦并行：一边翻页抓取结果列表，一边在后台启动已拿到链接的文献页面解析。利用异步 I/O 或线程池，使得浏览器在等待网络响应时，CPU 可并行处理已有的数据。文献研究表明，通过多线程将多页爬取任务并行可将总耗时从 507 秒降至 41 秒。本方案可在提取第 1 页结果后立即着手抓取这些结果的详情，同时浏览器继续加载第 2 页结果，以流水线方式减少空闲时间。
- **浏览器实例管理**：为提高稳定性，可采用**会话池**策略。预先启动少量浏览器实例，重复利用它们执行多个搜索任务，避免频繁创建销毁。每实例隔离不同任务的 cookie 和缓

存（Playwright 支持上下文隔离），防止数据混淆。若某实例发生崩溃或内存过高，监控程序应及时重启该实例并将未完成任务转移到空闲实例继续。

- **模板持续改进**：信息提取模块最初依赖人工定义的规则模板。应制定策略随着使用逐步完善模板库。当遇到新出版商站点无法正确解析时，记录该 URL 及页面片段，离线分析其结构并更新解析规则。对于重点期刊平台（如 ACM、ScienceDirect 等），提前调研其 HTML 模式，编写专用解析逻辑插件。在系统运行中加入**站点识别**步骤：根据 URL 或页面特征自动选用对应解析器类，提高准确率。同时长期看，可积累训练数据（标记各字段在 HTML 中的 XPath 位置等），尝试训练 ML 模型来预测字段位置，实现半自动生成解析规则，减轻人工维护成本。
- **错误处理与容错**：自动化过程中可能出现各种异常，例如网络超时、元素未找到、JavaScript 执行错误等。应在关键步骤加异常捕获：如在页面加载等待一定时长后仍未出现所需元素，则记录该文献链接和错误信息，继续处理下一个，避免阻塞全局。对提取失败或内容缺失的文献，可做标记（比如 entry.status = 'failed'），稍后通过重新调度或改用备用途径(例如尝试解析 Google Scholar 提供的引用信息或其他学术数据库)补充信息。整个流程设计成**幂等**的：可以安全地对某查询重新运行一次抓取，系统会识别已存在的文献条目避免重复插入，而是更新完善信息（例如第一次缺摘要，第二次获取到了就更新该字段）。
- **任务超时与回收**：为防止个别搜索卡顿拖累系统，总体为每个查询任务设置**超时时间**。例如单次检索任务限定在两分钟内完成主要流程。若超时未完成，主动终止剩余爬取，标记任务不完全，并将此情况通过前端提示用户（例如“部分结果未获取，请稍后重试”）。同时释放占用的浏览器和线程资源。对于挂起的浏览器实例，设定空闲超时，闲置超过一定时间自动关闭，节省内存。

- **CI/CD 与部署**：开发阶段采用模块化架构，各模块单独调试。完成后通过 Docker 容器部署整个 Agent 系统，以便在服务器或云端运行，利于依赖环境配置统一。使用 Docker Compose 编排浏览器服务（可使用带有 Chrome 的镜像）、后端服务、数据库服务，使各部分解耦部署。持续集成过程中加入自动化测试，例如模拟输入查询验证是否能正确拿到特定已知文献的元数据，确保爬取解析功能稳定。
- **资源优化**：自动化抓取策略上尽量减少不必要的开销。例如多数情况下不下载 PDF，所以启动浏览器时可禁止 PDF 自动加载或使用 Page 请求拦截跳过大文件。对于图片、广告脚本也考虑禁用，以加快页面加载速度（Playwright 支持路由拦截资源请求来实现）。同时设置浏览器的无头模式和较低分辨率，减少渲染资源耗用。
- **可扩展性**：系统架构应考虑后续扩展，如支持其他学术搜索源（Semantic Scholar、CrossRef API 等）。因此在 Agent 模块设计上，把 Google Scholar 爬取逻辑独立封装，实现类似接口，使将来新增搜索来源时，仅需添加新的 Agent 子类而不影响其他部分。同样，黑板系统接口可以拓展用于其它分析 Agent 写入，让整个科研辅助平台具备插件式扩展能力。

安全与稳定性

安全性和稳定性是系统能长期可靠运行的保证，需要从以下方面考虑：

- **合法合规**：遵循目标网站的服务条款和法律法规。由于 Google Scholar 无官方 API 且禁止大规模抓取，系统的抓取频率和数据用途需把握分寸（仅用于科研辅助，不做商业批量采集）。对于校内订阅的文献资源，尊重版权限制，不向未授权用户分发全文。
- **访问控制**：对前端接口实施鉴权和权限管理。仅允许授权用户使用文献调研 Agent 功能，防止被恶意滥用导致异常流量。黑板系统中的数据应区分不同团队或课题，仅授权

成员可查看编辑。可采用基于 token 的认证机制，前端每次请求附带用户身份 token，后端验证权限后才执行操作。

- **数据安全**：黑板知识库中的文献信息和用户标注需要定期备份，重要数据（如用户评论）应持久化存储。防范数据库故障或数据丢失，可采用主从备份或定期导出存档。在前端，避免将敏感信息暴露在客户端（如隐藏内部 ID，用匿名 ID 代替）。
- **异常监控**：部署完善的日志和监控系统。浏览器自动化过程中的每一步操作（发送请求、解析字段等）都记录日志，当出错时日志中包含 URL 和原因，便于排查。引入监控工具监视系统 CPU、内存、浏览器进程数量等，防止资源泄露导致崩溃。比如连续多少次出现抓取失败或超时时，触发报警或自动重启相关模块以恢复。
- **稳定性优化**：浏览器作为较重的依赖，有内存泄漏和崩溃风险，需要妥善管理。可定期重启浏览器进程（例如每处理几十个页面后），释放累积的内存。对于已知不稳定的站点，抓取前先检查其响应大小、结构，以调整解析策略避免程序异常（如某些页面的特殊字符编码处理）。
- **黑板架构优势**：利用黑板模式，有利于系统稳定。各 Agent 模块通过黑板解耦，爬取、解析与前端展示松散连接：即使某个 Agent 子流程失败，黑板中已有的数据仍可部分展示，不影响用户查看已获取结果。同时其他 Agent 或人工仍可对黑板数据进行补充修改。这种松耦合架构提升了容错能力。
- **输入验证**：对用户输入的问题字符串进行检查，避免引入安全隐患。如防止 SQL 注入（在构造数据库查询时使用参数化）、避免将用户输入直接用于调用系统命令。对于浏览器自动化，把用户查询直接用于 Google Scholar 搜索通常是安全的，但仍需警惕特殊字符可能造成解析混乱，需要做好转义或过滤。

- **前端健壮性**：前端代码考虑各种异常状态，例如长时间无返回、部分结果字段缺失等情况，给出友好的提示或占位符，而不至于崩溃。用户评论内容在展示时进行转义防止 XSS 攻击。对于协同场景，加锁或冲突检测防止多人同时编辑同一条目导致状态不一致。

综上所述，本方案围绕浏览器自动化抓取学术资源、结构化提取文献信息，并通过黑板系统实现人机协同分析。模块划分明确，数据流动顺畅，辅以严谨的频率控制和安全策略，能有效满足文献调研 Agent 的需求。在开发阶段应根据上述方案逐步实现和测试各功能模块，确保系统稳健后投入使用，为科研人员提供高效可靠的文献调研助手。