

Documind z- 本地知识库软件完整开发方案

1. 项目概述

1.1 项目定位

构建一个私有化、全功能的桌面知识库管理系统，集成RAG（检索增强生成）能力，实现从多源数据导入→智能处理→向量存储→AI对话的完整闭环。

1.2 核心指标

- 离线优先：核心功能无网络依赖（除外部LLM清洗）
 - 性能目标：单文档向量化 < 5秒（10页文档），检索延迟 < 500ms
 - 资源控制：内存占用 < 4GB，模型显存占用 < 2GB
 - 扩展性：支持插件式文档解析器和清洗规则
-

2. 系统架构设计

2.1 整体架构（Electron + 微内核）

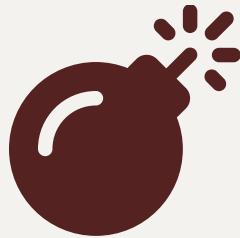


Syntax error in text
mermaid version 11.4.1

2.2 进程模型

- 主进程（Main）：
 - 负责文件I/O、数据库、模型推理等阻塞操作
 - 使用`worker_threads`处理CPU密集型任务（向量化）
 - 提供IPC API供渲染进程调用
- 渲染进程（Renderer）：
 - 纯UI层，无Node.js直接依赖
 - 通过预加载脚本（preload）暴露安全API
 - 状态管理使用Pinia

2.3 数据流转



Syntax error in text
mermaid version 11.4.1

3. 核心模块设计

3.1 文档处理引擎（DocProcessor）

3.1.1 解析器矩阵

| 文件类型 | 解析方案 | 依赖库/工具 | 处理策略 |
|----------|------------|---|------------|
| Markdown | 原生解析 | <code>markdown-it + front-matter</code> | 保留结构元数据 |
| PDF | OCR + 文本提取 | <code>pdf-parse + Tesseract.js</code> | 按页分段 |
| DOCX | XML解析 | <code>mammoth</code> | 保留标题层级 |
| XLSX | 表格解析 | <code>xlsx</code> | 按行/列生成文本描述 |
| PPTX | XML解析 | <code>pptx-parser</code> | 按幻灯片分段 |
| TXT | 编码检测 | <code>jschardet + iconv-lite</code> | 智能分段 |
| 图片 | OCR | <code>Tesseract.js</code> | 图片描述生成 |
| 粘贴文本 | 正则提取 | 原生 | 直接分段 |

3.1.2 数据清洗模块（可配置）

```

interface CleaningChain {
  steps: CleaningStep[];
}

interface CleaningStep {
  type: 'dedupe' | 'correct' | 'summarize' | 'format';
  llmProvider?: 'openai' | 'claude' | 'local';
  prompt?: string; // 自定义清洗Prompt
  batchSize?: number;
}

```

- 去重：基于SimHash的近似去重
- 纠错：调用外部LLM API（可配置成本阈值）
- 摘要：生成文档级/段落级摘要
- 格式化：统一编码、换行符、空白字符

3.1.3 智能分段策略

```
interface SegmentRule {
    minLength: number; // 最小字符数 (默认200)
    maxLength: number; // 最大字符数 (默认800)
    overlap: number; // 重叠字符数 (默认10%)
    delimiter: RegExp; // 分段正则 (默认按句子)
    preserveStructure: boolean; // 是否保留标题/列表结构
}
```

- 递归分段：对超长文本递归拆分
- 语义完整：优先在段落边界、标题处切分
- 上下文保留：滑动窗口保留前后文引用

3.2 向量化服务 (Embedding Service)

3.2.1 模型管理

- 模型：`Qwen3-0.6B-Embedding-Q4_K_M.gguf`（量化版，控制内存）
- 推理引擎：`llama.cpp`（N绑定 `node-llama-cpp`）
- 设备适配：
 - 优先使用GPU（CUDA/MPS）
 - 回退到CPU（配置线程数）
- 缓存策略：LRU缓存最近10,000个向量

3.2.2 批处理与流式处理

```
interface VectorizeOptions {
    batchSize: number;           // 每批数量 (默认64)
    concurrency: number;         // 并发数 (根据CPU核心数)
    retryTimes: number;          // 失败重试
    progressCallback?: (progress: number) => void;
}
```

- 进度追踪：实时显示向量化进度条
- 断点续传：记录失败索引，支持重试
- 内存保护：超过内存阈值自动限流

3.3 数据库设计（双库协同）

3.3.1 SQLite（元数据管理）

```
-- 知识库表
CREATE TABLE knowledge_base (
    id TEXT PRIMARY KEY,
    name TEXT UNIQUE,
    description TEXT,
    embedding_model TEXT,
    created_at TIMESTAMP
);

-- 文档表
CREATE TABLE documents (
    id TEXT PRIMARY KEY,
    kb_id TEXT,
    filename TEXT,
    filepath TEXT,
    filesize INTEGER,
    checksum TEXT,
```

```
status TEXT, -- 'pending', 'processing', 'completed', 'failed'
metadata JSON,
created_at TIMESTAMP,
FOREIGN KEY (kb_id) REFERENCES knowledge_base(id)
);
```

-- 片段表

```
CREATE TABLE segments (
    id TEXT PRIMARY KEY,
    doc_id TEXT,
    content TEXT,
    chunk_index INTEGER,
    word_count INTEGER,
    embedding_id TEXT,
    FOREIGN KEY (doc_id) REFERENCES documents(id)
);
```

-- 会话表

```
CREATE TABLE conversations (
    id TEXT PRIMARY KEY,
    kb_id TEXT,
    title TEXT,
    messages JSON,
    created_at TIMESTAMP
);
```

3.3.2 ChromaDB (向量存储)

```
collection = chroma_client.create_collection(
    name="kb_segments",
    embedding_function=embedding_function,
    metadata={"hnsw:space": "cosine"} # 余弦相似度
)

# 存储结构
```

```
{  
    "ids": ["seg_001", "seg_002"],  
    "embeddings": [[[...], [...]],  
    "metadata": [  
        {"doc_id": "doc_001", "chunk_index": 0, "word_count": 350},  
        {"doc_id": "doc_001", "chunk_index": 1, "word_count": 380}  
    ],  
    "documents": ["片段内容1", "片段内容2"]  
}
```

3.3.3 混合检索策略

```
interface SearchOptions {  
    query: string;  
    topK: number;           // 返回数量  
    threshold: number;     // 相似度阈值  
    filter?: {  
        doc_id?: string[];  
        date_range?: [Date, Date];  
    };  
    hybrid: boolean;        // 是否启用混合检索  
}
```

- 向量检索：ChromaDB原生ANN搜索
- 关键词检索：BM25算法（使用SQLite FTS5）
- 重排序：RRF融合排序（Reciprocal Rank Fusion）

3.4 AI对话引擎（RAG Pipeline）

3.4.1 对话状态管理

```
interface ConversationState {  
    sessionId: string;  
    history: Message[];  
    contextSegments: Segment[]; // 引用的知识片段  
    totalTokens: number;  
    settings: {  
        topK: number;  
        temperature: number;  
        maxTokens: number;  
        stream: boolean;  
    }  
}
```

3.4.2 Prompt工程模板

```
# 知识库问答系统Prompt
```

角色定义

你是一个基于本地知识库的智能助手。请根据提供的**事实片段**回答问题，严格遵循以下规则。

知识片段（按相关性排序）

```
{% for seg in segments %}  
[来源: {{ seg.doc_name }}]{{ seg.chunk_index }}]  
{{ seg.content }}  
{% endfor %}
```

对话历史

```
{% for msg in history %}  
{{ msg.role }}: {{ msg.content }}  
{% endfor %}
```

用户问题

```
User: {{ query }}
```

回答要求

1. 仅使用提供的知识片段内容，不超范围发挥
2. 引用来源时请标注 ^[来源：文件名#段落号]^
3. 若知识不足，明确说明"根据现有资料无法回答"
4. 回答需简洁、准确、结构化
5. 使用{{ language }}回答

最终回答

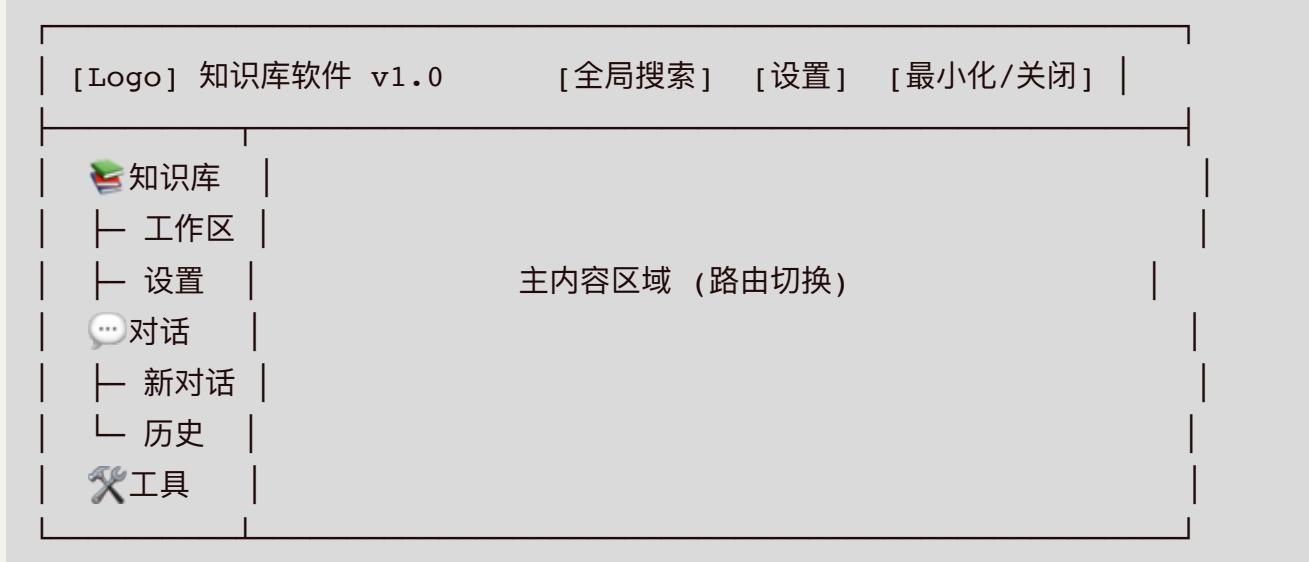
Assistant:

3.4.3 流式响应处理

- SSE推送：Server-Sent Events实现流式输出
 - 引用标注：边生成边解析 [来源:] 标记，高亮显示
 - token计数：实时显示消耗token数
 - 中途取消：支持用户主动终止生成
-

4. UI/UX 设计

4.1 主界面布局



4.2 核心页面

4.2.1 知识库管理页

- 工作区列表：显示所有知识库，含文档数量、最后更新时间
- 导入面板：
 - 拖拽上传区域
 - 格式支持提示
 - 粘贴文本弹窗
- 处理队列：
 - 显示待处理、处理中、失败任务
 - 支持暂停/恢复/重试
 - 实时进度条
- 统计图表：
 - 文档类型分布饼图
 - 每日新增趋势折线图
 - 存储占用分析

4.2.2 文档管理页

- 文档列表：
 - 支持筛选（类型、日期、状态）
 - 快速预览（前500字）
 - 操作：重新向量化、删除、下载
- 分段预览：
 - 点击文档查看分段详情
 - 显示每段的相似片段（去重参考）
 - 编辑分段（高级模式）
- 元数据编辑器：
 - 自定义标签
 - 手动添加摘要
 - 设置文档权重

4.2.3 对话界面

- 对话列表（左侧）：
 - 自动生成的标题
 - 最后消息时间
 - 删除/归档
- 聊天区域（右侧）：
 - 消息气泡（支持Markdown渲染）
 - 引用高亮：点击[来源]展开原文
 - 相似问题推荐：基于历史记录生成
 - 重新生成：更换模型参数重新回答
- 输入工具栏：
 - @提及知识库（限定检索范围）

- #标签（快速过滤）
- /命令（快捷操作）
- 检索详情面板（可折叠）：
 - 显示检索到的原始片段
 - 相似度分数可视化
 - 耗时分析

4.3 交互细节

- 快捷键：Ctrl+K快速搜索、Ctrl+N新建对话、Ctrl+Shift+I导入
 - 自动保存：文档编辑、配置修改实时保存
 - 主题切换：浅色/深色/系统跟随
 - 通知系统：操作成功/失败/警告提示
 - 进度反馈：所有耗时操作显示进度条
-

5. 工程实现

5.1 项目结构 (Electron-Vite)

```
electron-vite-kb/
├── packages/
│   └── main/          # 主进程
│       ├── src/
│       │   ├── services/      # 核心服务
│       │   │   ├── DocProcessor.ts
│       │   │   ├── EmbeddingService.ts
│       │   │   ├── DatabaseService.ts
│       │   │   └── LLMPProxy.ts
│       │   ├── workers/      # 工作线程
│       │   └── vectorize.worker.ts
```

```
|   |   |   \_ index.ts
|   |   \_ vite.config.ts
|   \_ preload/          # 预加载脚本
|       \_ src/
|           \_ index.ts      # 暴露安全的IPC API
\_ renderer/          # 渲染进程
    \_ src/
        \_ router/        # 路由
        \_ stores/        # Pinia状态管理
        \_ views/         # 页面
        \_ components/    # 组件
            \_ App.vue
        \_ vite.config.ts
\_ models/             # 模型文件
    \_ qwen3-0.6b-embedding-q4.gguf
\_ resources/          # 静态资源
\_ build/              # 构建配置
\_ package.json
```

5.2 技术栈选型

- **Electron-Vite**: 快速启动、HMR、TypeScript原生支持
- **Vue 3.5**: 组合式API、性能优化
- **TailwindCSS 3.4**: 原子化CSS，快速开发
- **ChromaDB 0.5**: 轻量级向量数据库
- **Better-SQLite3**: 高性能同步SQLite驱动
- **Node-llama-cpp**: GGUF模型推理
- **Tesseract.js**: 纯JS OCR
- **Pinia 2.2**: 状态管理
- **VueUse**: Composition Utils
- **Mermaid + Markmap**: 知识图谱可视化

5.3 配置系统

```
interface AppConfig {
    // 模型配置
    embedding: {
        modelPath: string;
        gpuLayers: number;
        contextLength: number;
    };

    // 数据库配置
    database: {
        persistPath: string;
        backupEnabled: boolean;
    };

    // 外部LLM配置
    llmCleaner: {
        provider: 'none' | 'openai' | 'claude';
        apiKey: string;
        baseURL: string;
        model: string;
        maxCostPerDoc: number; // 单文档清洗成本上限
    };

    // 界面配置
    ui: {
        theme: 'light' | 'dark' | 'system';
        language: 'zh-CN' | 'en-US';
        fontSize: number;
    };

    // 性能配置
    performance: {
        maxFileSize: number; // MB
        batchSize: number;
    };
}
```

```
    maxConcurrency: number;  
};  
}
```

- 存储位置: `~/.config/knowledge-base/config.json`
- 热重载: 配置修改后自动应用 (无需重启)

5.4 自动化测试与交付

- 测试金字塔: `Vitest` 单元测试覆盖核心工具函数, `Vue Test Utils` 驱动组件测试, `Playwright` 执行端到端流程校验
- 数据伪造: 使用 `msw/faker` 构造解析、检索、对话的模拟数据, 保障离线测试
- 性能回归: 构建独立样本库, CI 中运行向量化、检索基准, 对比耗时与资源占用阈值
- 静态分析: ESLint + TypeScript 严格模式、`vue-tsc --noEmit`、`depcheck` 确保依赖安全
- CI/CD 流程: GitHub Actions 三阶段流水线 (`lint/test` → 构建 → 产物签名与发布草稿), 配合 `electron-builder --publish never` 生成待发布包

6. 性能优化

6.1 向量化优化

- 增量更新: 仅处理新增/修改文档, 跳过未变更文件 (基于 `checksum`)
- 并发控制: 根据 CPU 核心数动态调整 `concurrency`
- 内存映射: 使用 `mmap` 加载模型, 减少内存占用
- 缓存预热: 启动时预热高频查询

6.2 检索优化

- 多路召回：向量检索（top-50）+关键词检索（top-50），再重排序
- 查询缓存：缓存高频查询结果（5分钟TTL）
- 元数据索引：为ChromaDB字段建立SQLite索引加速过滤
- 向量压缩：使用乘积量化（PQ）减少存储

6.3 渲染优化

- 虚拟滚动：文档列表、对话历史使用`vue-virtual-scroller`
- 懒加载：OCR、预览内容按需加载
- **Web Worker**：文本解析、正则处理放Worker线程
- 图像优化：图片预览使用`sharp`生成缩略图

6.4 异常恢复与自监控

- 健康检查：主进程定时轮询向量化服务与数据库连接，异常自动重启Worker
 - 故障回滚：向量化/解析失败保留原始中间数据，允许从断点任务重新调度
 - 资源守卫：超过CPU/内存阈值时暂停新任务并通知用户，可在设置面板手动恢复
 - 报警联动：可选Webhook推送（如Slack/飞书）发送严重错误与备份失败告警
-

7. 部署与分发

7.1 构建配置

- 跨平台：`macOS`（`x64/arm64`）、`Windows`（`x64`）、`Linux`（`AppImage`）
- 代码签名：开发自签名证书，正式环境Apple/Microsoft认证
- 自动更新：集成`electron-updater`

- 体积优化：
 - 使用 `electron-builder` 的 `files` 白名单
 - 模型文件外部下载（首次启动）
 - 启用 `ASAR` 打包

7.2 首次启动流程

1. 检测模型文件是否存在
2. 若不存在，弹窗选择下载或手动指定路径
3. 初始化数据库、ChromaDB
4. 创建默认知识库
5. 显示欢迎向导（功能介绍、快捷键）

7.3 数据迁移与备份

- 导出：支持导出为 `.kbpkg`（含元数据+向量+原始文件）
 - 导入：验证 `checksum`，增量导入
 - 自动备份：每日凌晨打包 SQLite+ChromaDB 到备份目录
 - 版本兼容：提供迁移脚本，支持跨版本升级
-

8. 扩展与生态

8.1 插件系统（二期）

```
interface Plugin {  
    name: string;  
    hooks: {  
        'file:parse'?: (file: File) => Promise<string>;  
        'segment:preprocess'?: (segments: Segment[]) => Segment[];  
        'search:postprocess'?: (results: Result[]) => Result[];  
        'chat:beforeSend'?: (messages: Message[]) => Message[];  
    };  
}
```

- 插件市场：内置插件管理器
- 沙盒隔离：使用 `vm2` 运行插件代码

8.2 知识图谱（三期）

- 实体提取：调用LLM提取人名、地名、概念
- 关系建模：构建实体关系图
- 可视化：使用 `vis-network` 或 `cytoscape.js`

8.3 团队协作（远期）

- 多用户支持：基于SQLite的用户权限
- 冲突解决：文档版本控制（CRDT）
- 同步机制：WebDAV/云盘同步

9. 开发里程碑

Phase 1 (MVP, 4周)

- ✓ 项目骨架搭建 (Electron-Vite)
- ✓ 基础文档解析 (MD/PDF/TXT)
- ✓ 向量化服务集成
- ✓ 对话界面 (基础版)
- ✓ 知识库增删改查

Phase 2 (功能完善, 3周)

- ✓ 全格式支持 (Office/图片/OCR)
- ✓ 数据清洗功能
- ✓ 分段预览与编辑
- ✓ 配置系统
- ✓ 性能优化

Phase 3 (体验打磨, 2周)

- ✓ UI美化 (动效、主题)
- ✓ 高级搜索 (过滤器、排序)
- ✓ 数据统计与可视化
- ✓ 导入导出功能
- ✓ 自动化测试 (端到端)

Phase 4 (发布准备, 1周)

- ✓ 构建与签名
 - ✓ 编写文档
 - ✓ 用户测试反馈修复
 - ✓ 发布GitHub Release
-

10. 风险与对策

| 风险 | 概率 | 影响 | 对策 |
|---------------|----|----|-----------------|
| 模型加载内存溢出 | 中 | 高 | 量化模型、动态卸载、内存监控 |
| Electron包体积过大 | 高 | 中 | 按需加载、模型外部化、ASAR |
| 文档解析兼容性 | 中 | 中 | 多解析器回退、支持格式白名单 |
| ChromaDB数据损坏 | 低 | 高 | 定期备份、事务日志、启动校验 |
| 外部LLM API失效 | 低 | 中 | 本地备选方案、离线模式提示 |

11. 监控与调试

11.1 日志系统

```
enum LogLevel {
    DEBUG = 0,
    INFO = 1,
    WARN = 2,
    ERROR = 3
}

interface LogEntry {
    timestamp: number;
    level: LogLevel;
    module: string;
    message: string;
    data?: any;
}
```

- 存储：滚动写入`~/.config/knowledge-base/logs/app.log`
- 查看：设置页面提供日志查看器（支持筛选、导出）

11.2 性能监控

- 资源面板：CPU、内存、磁盘实时图表
- 任务追踪：显示当前后台任务（带取消）
- 瓶颈分析：记录各阶段耗时（解析/向量化/检索）

11.3 调试工具

- 开发者工具：Ctrl+Shift+I打开Electron DevTools
 - API测试：内置Swagger UI测试IPC接口
 - 数据检查：直接查看SQLite/ChromaDB原始数据
-

12. 文档与维护

12.1 用户文档

- 快速开始：GIF动图演示核心流程
- 格式支持表：各文件类型的最佳实践
- FAQ：常见问题解决方案
- 快捷键清单：可打印的PDF

12.2 开发者文档

- 架构说明：Mermaid架构图
- IPC API文档：TypeScript类型 + 示例
- 数据库ER图：使用dbdiagram.io
- 贡献指南：Git Flow规范

12.3 长期维护

- 版本策略：语义化版本（SemVer）
 - 兼容性：提供LTS版本（每年更新）
 - 社区支持：GitHub Discussions + Issue模板
 - 安全更新：依赖自动审计（Dependabot）
-

13. 安全与合规

13.1 数据安全

- 本地加密：用户可启用知识库目录与SQLite文件的AES-256加密，密钥存储于macOS Keychain/Windows Credential Manager
- 访问控制：应用启动可配置主密码，唤起LLM对话或导出敏感数据时需二次确认
- 传输安全：调用外部LLM或云备份时强制HTTPS，支持自定义CA与代理

13.2 隐私与合规

- 可审计性：所有调用外部服务的请求均记录数据指纹，便于审计数据出境情况
- 数据驻留：默认本地存储，无需联网；若启用云同步需显式同意并提示风险
- 法规参考：遵循GDPR数据最小化原则，提供一键删除知识库与日志的能力

13.3 安全流程

- 依赖巡检：每周自动更新安全报告（npm audit、Snyk），高危依赖7日内修复
 - 渗透测试：发布前执行Electron安全清单检查，验证IPC白名单、Content Security Policy
 - 应急预案：建立安全事件响应表，定义发现→隔离→修复→通报四步流程
-

14. 总结

此方案实现了一个功能完备、架构清晰、可扩展性强的本地知识库软件。关键创新点：

1. 双库协同：SQLite负责元数据与事务，ChromaDB专注向量检索，发挥各自优势
2. 可配置清洗链：将数据预处理抽象为可编排的链式任务，支持零代码扩展
3. 智能分段：基于语义完整性的分段策略，平衡检索精度与召回率
4. 混合检索：向量+关键词双路召回，解决向量检索的精确匹配短板
5. 资源可控：从模型量化到并发控制，全方位优化内存/CPU占用

该方案充分考虑了开发效率、用户体验、长期维护的平衡，可作为实际开发的详细蓝图。