Collector + (Read-AI) 技术报告

1. 执行摘要 (Executive Summary)

项目名称: Collector + (代号: Read-AI)

核心问题: 解决由 "Read Later"（稍后阅读）囤积症导致的 "Read Never"（从不阅读）现象。用户经常保存大量长文或视频，但因认知负担过重而无法消化。

解决方案: 构建一个 基于 LLM 的原生 Web 应用，通过 **"直觉对赌" (Intuition Bets)** 的游戏化机制，瞬间将长篇内容转化为一系列二元选择题（是/否）。用户在 30 秒内通过预测性答题完成核心知识点的吸收，实现“收藏即阅读”。

当前状态: MVP (Minimum Viable Product) 已上线。项目采用高效的 **线性单体架构 (Linear Monolithic Architecture)**，未使用复杂的 Agent 框架，确保了极低的延迟和极高的稳定性。

2. 系统架构 (System Architecture)

本系统采用 **无状态单体后端 + 分离式前端** 的轻量化架构，旨在最大化响应速度并降低运维复杂度。

2.1 架构风格: 线性与确定性

与市面上流行的多 Agent 系统（Multi-Agent Systems）不同，Collector + 目前**不依赖** LangChain、AutoGen 或 CrewAI 等复杂编排框架。我们认为对于内容解析和结构化任务，精心设计的 **线性处理管道 (Linear Processing Pipeline)** 配合 **高级提示工程 (Advanced Prompt Engineering)** 是最高效的解决方案。

2.2 核心组件

- **前端 (Frontend)**:
  - **框架**: React 19 + Vite (构建工具)
  - **样式**: Tailwind CSS (原子化 CSS) + Lucide-React (图标库)
  - **设计哲学**: 极简主义、响应式设计，强调“卡片翻转”的微交互体验。
- **后端 (Backend)**:
  - **框架**: Python FastAPI (异步高性能)
  - **AI 核心**: MiniMax 大模型 (通过 OpenAI SDK 调用)
  - **数据层**: SQLite (轻量级持久化) + 内存缓存
- **内容处理层**:
  - **Trafilatura**: 通用网页内容提取
  - **BeautifulSoup4**: 针对微信公众号/B站等特定平台的 DOM 解析与清洗

3. 核心技术实现 (Core Technical Implementation)

3.1 线性处理管道 (The Linear Pipeline)

对应文件: `backend/main.py`

系统处理流程是严格线性的，确保了结果的可预测性和低延迟：

1. **输入**: 用户提交 URL (文章或视频链接)。
2. **获取 (Fetch)**:
    - 多策略路由：优先检测 Bilibili/微信 URL，使用特定 headers 和 cookies 绕过反爬。
    - 通用兜底：使用 Trafilatura 提取纯文本。
3. **截断与清洗**: 按 Token window 限制截断文本（目前限制前 12k 字符），去除 HTML 噪音。
4. **注入 (Context Injection)**: 将清洗后的文本注入到预定义的 System Prompt 中。
5. **推理 (Inference)**: 单次调用 LLM API。
6. **结构化 (Structured Output)**: LLM 直接输出严格的 JSON 格式（包含问题、选项、正确答案、原文证据）。
7. **持久化**: 将 JSON 结果与文章元数据存入 SQLite。

3.2 高级提示工程 (Prompt Engineering Strategy)

我们没有使用 Agent 进行多轮对话，而是设计了一个复杂的 **"认知科学家" (Cognitive Scientist)** System Prompt。

该 Prompt 也是系统的核心 IP，它指示 LLM 在一次推理中完成以下三个步骤：

1. **内容重构**: 将碎片化的视频字幕或口语转录为严谨的书面报道。
2. **预测市场问题设计**: 挖掘文中 3 个反直觉或有争议的观点，构造二元对立的“对赌”问题。
3. **格式约束**: 强制输出为纯 JSON 格式，包含原文证据引用。

3.3 状态管理 (State Management)

对应文件: `frontend/App.tsx`, `frontend/views/Dashboard.tsx`

前端采用 React Context 和本地 State 结合的方式：

- **单一数据源**: `App.tsx` 持有全局状态（如 `dashboardQuizData`），确保用户在浏览不同视图时数据不丢失。
- **交互逻辑**: `QuizCard.tsx` 封装了翻转逻辑。用户点击选项 -> 卡片翻转 -> 展示 `evidence` (原文证据)。这种 "Question -> Interaction -> Evidence" 的闭环是核心体验。

---

4. 关键特性与机制 (Key Features)

4.1 通用内容解析器 (Universal Content Parser)

后端实现了针对主流内容平台的特定适配：

- **微信公众号**: 解决了图片懒加载和动态渲染文本的问题，利用 Jina.ai 代理增强抓取成功率。
- **Bilibili**: 自动提取视频字幕（CC Subtitles），并自动补全标点符号，将视频内容转化为可阅读的文本。

4.2 "直觉对赌" 交互 (The Betting Interaction)

不同于传统的 "Summary" (摘要) 工具，Collector + 不直接给结果。

- **机制**: 提问 -> 用户预判 -> 揭晓答案。
- **价值**: 这种机制迫使大脑从“被动接收”切换到“主动预测”模式，显著提高了记忆留存率。

4.3 延迟优化 (Latency Optimization)

由于弃用了多 Agent 协作（通常需要数分钟的 Agent 间来回通信），本系统实现了 **E2E (End-to-End) 15-30 秒** 的生成速度。这对用户留存至关重要。

---

5. 技术栈清单 (Tech Stack)

领域	技术/库	用途
Backend Framework	**FastAPI**	高性能异步 API 服务
AI Inference	**OpenAI SDK (MiniMax)**	调用 MiniMax abab6.5s 模型
Web Server	**Uvicorn**	ASGI 服务器
Scraper	**Trafilatura / Requests / BS4**	网页与字幕抓取
Frontend Framework	**React 19**	构建用户界面
Build Tool	**Vite**	极速前端构建
Styling	**Tailwind CSS**	统一样式系统
Deployment	**Docker**	容器化交付 (ModelScope/Vercel)

---

6. 未来路线图 (Future Roadmap)

虽然当前线性架构在性能上表现优异，但随着功能扩展，我们将考虑以下演进路径：

1. **引入多 Agent (针对复杂任务)**:
    - 当需要生成深度研报（需联网搜索补充信息、交叉验证事实）时，当前的单次 Prompt 将不再从容。届时我们将引入 **Researcher Agent** (负责搜索) 和 **Writer Agent** (负责整合) 的协作模式。
2. **数据库升级**:
    - 从 SQLite 迁移至 **PostgreSQL**，以支持更高并发的用户历史记录和社交化功能（排行榜、分享挑战）。
3. **边缘计算**:
    - 考虑将部分内容清洗逻辑下放到 Edge Function (如 Cloudflare Workers)，进一步降低中心服务器负载。

结论: Collector + 目前的技术架构是 **务实且高效** 的。它用最简洁的工程手段最大化了 LLM 的能力，避免了过度设计的陷阱，成功验证了 "Gamified Reading" 的产品价值。