AI PRODUCT PORTFOLIO

AI产品经理实习作品集

AI 产品 · Agent · Skill · Workflow

苏州大学 | 建筑学 | AI 产品经理实习方向

姓名：赵子翰

SELECTED WORKS / 2026

01 **Store Price Checker**

AI PRODUCT

02 **Music Agent**

AGENT / MCP

03 **EVA Photo Redraw**

AI SKILL

04 **Knowledge Runtime**

KNOWLEDGE / CONTEXT

z987535@gmail.com

SUZHOU, CN

Store Price Checker｜从“顾客问价格”到自助查价闭环

V1 聚焦本店售价查询与未识别商品补库，先完成单店场景下的查价闭环。

顾客端：使用扫码枪或键盘输入商品编码，命中后立即展示商品名称、规格与本店售价。

顾客端不使用手机摄像头扫码；手机拍照仅用于店主补库时的商品识别。

问题定义

真正的问题不是“怎么识别更多商品”，而是顾客无法快速获取店主维护的本店售价，导致查价长期依赖人工问询。

因此，V1 只做一条能够独立完成的核心闭环：

店主维护商品与售价 → 顾客自助查价 → 未识别商品进入补库 → 店主确认资料并定价 → 补库后再次扫码即可命中。

- 价格以店主维护的本店售价为准（系统字段 priceCents），不涉及采购成本或供应商报价。
- V1 不扩成 POS、库存、会员、多店或云同步。
- 顾客同步查价仅依赖本地数据，不被 AI 或外部网络阻塞。

简单查价不够：真正的问题发生在“未命中”之后

当商品不在本地库中，系统既要快速响应顾客，也要把未知商品以低成本、可追踪的方式纳入补库闭环。

典型 Bad Case

如果只做一个“编码查询页”，产品会很快遇到几个无法回避的问题：

- 条码合法，但本地数据库没有对应商品。
- 同一个未知商品可能被顾客反复查询，不能每次都重新建任务。
- 网络搜索对精确 GTIN 的召回不稳定，图片识别也可能误判。
- AI 可以辅助识别商品资料，但不能决定售价，也不能直接生成正式商品。
- 补库前产生的未命中扫码也需要保留，否则真实的顾客需求会在商品入库后丢失。

所以真正需要设计的是：

> 未知商品如何形成唯一补库任务，经过候选识别与店主确认后正式入库，并保留此前的未命中记录。

新增 / 补库入口

AI 识别候选资料

唯一补库任务

同一 storeId + code 只保留一个 ProductDiscoveryTask；重复扫码继续记录 ScanLog，但不会重复创建补库任务。避免因重复扫码反复创建任务和重复触发补库流程。

AI 只给候选

Search / DeepSeek / Qwen Vision 只产生候选信息；没有可靠证据时保持“未识别（unidentified）”，不让模型猜商品，也不自动定价或入库。

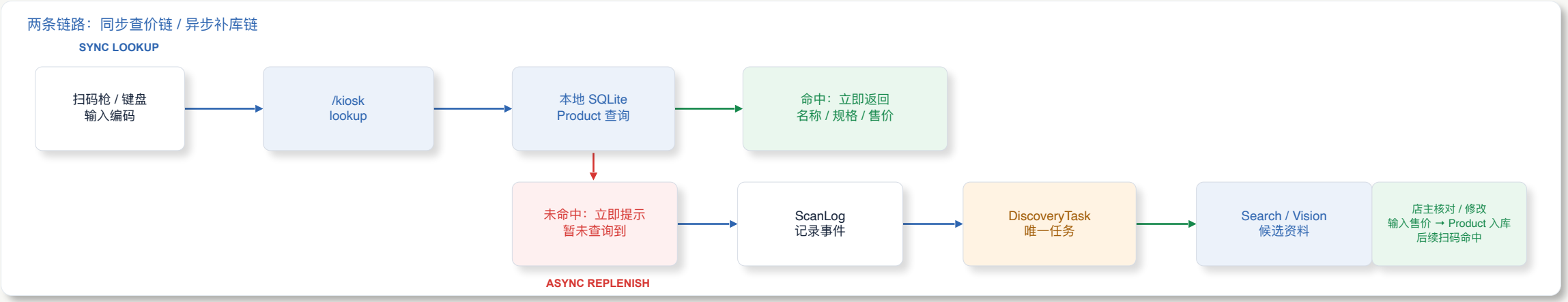
历史需求保留

补库完成后，历史 not_found 按原 code 回填 productId；补库前的真实需求仍然可以追溯。

设计转折：重点不是提高一次识别成功率，而是把 not_found → 候选 → 人工确认 → 正式商品 做成可控、可追踪的状态转换。

解决思路：核心查价保持确定性，AI 只进入异步补库

将顾客同步查价与店主异步补库拆成两条链，让已有商品的查价不依赖模型或外部网络。



AI 负责不确定信息提取

- 从搜索证据或图片中提取品牌、名称、规格。
- 识别结果是候选，不直接进入正式商品表。
- 无证据时不让模型猜商品。

后端负责确定性约束

- GTIN 形状、Check Digit、逐位一致性校验。
- 同店 code 唯一，未知任务唯一。
- 售价只能由店主输入，AI 永不定价。

产品负责闭环体验

- 顾客立即得到明确反馈，不等待识别。
- 店主在后台核对、修改、定价后保存。
- 补库完成后，后续扫码立即命中。

产品判断：不是所有环节都需要 AI——核心查价保持确定性，只在补库环节用 AI 提效。

关键落地：把商品、价格与 AI 候选的边界固化为产品规则

这不是“接入模型”的 Demo，而是用数据真相源、AI 安全边界和实机验证保证可用性。

图片识别结果必须经过条码与置信度校验，且只能作为候选资料回填。

三个业务真相源

Product

正式商品与售价真相源
管理：名称 / 品牌 / 规格 / active / priceCents

ScanLog

原始扫码事件真相源
记录：found / not_found / code / scannedAt / productId

DiscoveryTask

未知商品处理状态
负责：pending → processing → pending_review / unidentified / failed→ pending_price → completed
只管理补库流程，不等于正式商品。

数据边界清晰，后续统计、补库与价格维护才不会混乱。

AI 边界

Search / DeepSeek / Qwen Vision 只提供候选；AI 只生成候选资料，不填写售价、不决定正式商品编码；缺少真实证据时不猜商品。

安全校验

Qwen 请求不携带任务 GTIN；图片条码必须通过 Check Digit，并与任务 / 输入 GTIN 做确定性比较。发生冲突时禁止应用识别结果；识别置信度与“是否为同一商品”分开判断。GTIN conflict → blockApply=true

真实环境

Windows 生产模式、局域网二维码、手机后台，以及 iPhone 多图上传和视觉识别正反测试均有实机记录。

704
TESTS PASSED

5/5
LINT · TYPECHECK · TEST · BUILD · WINDOWS BUILD

58/58
HISTORY SCAN BACKFILL

WINDOWS + iPhone
REAL DEVICE VERIFIED

Music Agent｜从单次推荐到长期个性化

目标不是重新做一个音乐播放器，而是让收藏、播放与反馈能够持续参与后续推荐。

真实界面：个性化入口围绕当前播放、自然语言需求与快捷动作组织。

问题定义

现有音乐产品已经解决搜索、播放与内容分发，但一次推荐之后，用户产生的新收藏、播放与反馈，如何继续影响下一次推荐？如果每次交互都重新开始，Agent 就只是一个会推荐歌曲的聊天入口，而不是长期个性化服务。

产品目标

01｜保留长期状态
持续保存收藏、偏好画像、推荐历史与用户反馈。

02｜结合当前需求
推荐时同时参考长期偏好、当前播放内容与即时意图，而不是只依赖静态画像。

03｜形成反馈闭环
让推荐结果继续进入试听、播放与反馈，并将新的使用结果带入下一轮推荐。

产品边界

Music Agent 不替代 Apple Music。
它复用 Apple Music 与本地音乐数据，重点处理个人状态、推荐决策与反馈回流；播放与音乐内容仍由真实音乐服务承载。

核心判断

不是“这次帮我推荐 5 首歌”，而是让今天的收藏、播放与反馈，成为下一次推荐真正会使用的信息。

PRODUCT POSITION

用户 → **Music Agent**
个性化理解 / 推荐 / 反馈 → **Apple Music +**
本地音乐数据

Product Goal｜让每一次使用，都为下一次推荐留下可复用状态。

个性化不是一句 Prompt，而是一组持续更新的用户状态

把收藏、偏好画像、推荐历史与反馈分开保存，让下一次推荐真正建立在过去的使用结果上。

PERSONAL MUSIC STATE

“记住我”不是保存一段聊天记录，而是让关键状态持续存在并被后续推荐重新使用。

Library｜收藏
用户明确收藏或已确认喜欢的音乐，构成最直接的个人音乐数据。

Profile｜偏好画像
系统当前对用户长期音乐偏好的结构化判断。

Recommendation History｜推荐历史
记录过去已经推荐过的歌曲与候选，减少重复推荐。

Feedback｜反馈
记录用户针对歌曲或推荐结果产生的明确反馈事实。

状态如何进入推荐

Feedback ≠ Learning｜跳过 ≠ 不喜欢；听完 ≠ 喜欢。系统先记录事实，证据足够后再更新长期偏好。

当前意图

“找类似这首 / 换个心情”等请求，会改变本次推荐的判断条件。

推荐迭代

推荐历史与最新反馈继续参与下一轮推荐，而不是每次从零开始。

核心机制｜状态持续存在 → 新行为先记录为事实 → 证据足够后更新长期偏好 → 影响下一轮推荐。

一次推荐：长期偏好 + 当前意图 + 可引用的歌曲身份

长期偏好相对稳定，但每一次音乐需求都有不同上下文；自然语言可以模糊，最终绑定的歌曲对象必须明确。

RECOMMENDATION INPUTS

长期状态
收藏 / 偏好 / 历史反馈

+

当前播放内容

+

当前意图
推荐 / 找类似 / 换心情

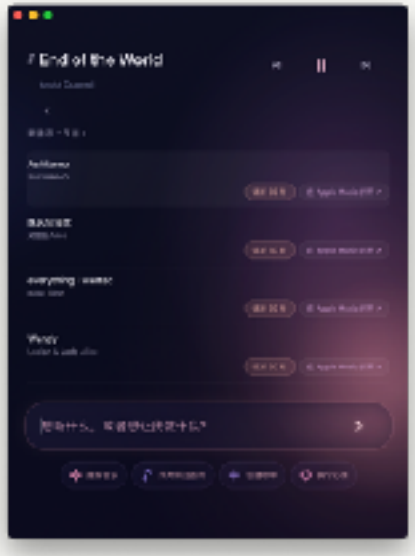
↓

生成本次 Recommendation Run

IDENTITY BINDING

“刚才第二首不错” → Recommendation Run → 第 2 个 Candidate → Canonical Track → 记录反馈

无法可靠解析时应追问，而不是猜测歌曲 ID；否则会污染推荐历史、反馈与后续学习。



01 长期偏好 ≠ 当前意图

喜欢某类音乐，不代表此刻一定想听它；“找类似这首的”会把当前歌曲纳入判断。

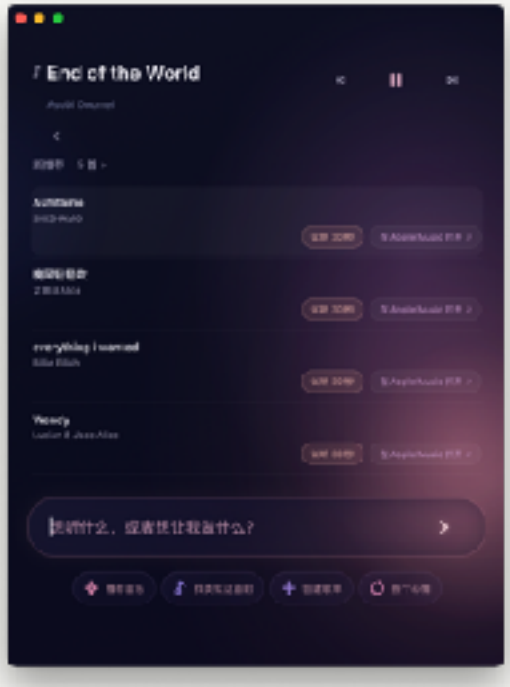
02 推荐不是文本答案

每次推荐都会形成可追踪的 Recommendation Run 与 Candidate，后续反馈才能准确绑定到具体歌曲。

核心判断｜推荐质量不只取决于模型生成，也取决于状态、上下文与歌曲身份是否被可靠解析。

推荐不是终点，Agent 还需要把结果变成行动

将自然语言需求与真实音乐操作连接起来，让推荐继续进入试听、播放与外部音乐服务。



推荐结果不是文本终点：用户可以继续试听、播放，或进入 Apple Music 完成后续操作。

TOOL SEMANTICS

Preview	临时试听推荐结果，不改变当前正式播放状态。
Play	恢复当前 Music.app 播放上下文，不等于“播放指定歌曲”。
Play Track	播放已可靠确定身份的指定歌曲，需要 canonical track ID。
Open Apple Music	在用户明确选择后，打开对应的 Apple Music 页面。

设计原则

Agent 能执行，不代表可以自行执行。涉及播放、试听或打开外部应用的动作，都必须来自当前用户意图；执行失败时，也不能擅自切换为其他 Tool。

一个 Music Agent，多种 AI 客户端；一个状态源，多种展示层

MCP 只负责连接与能力暴露，不重新实现推荐、偏好与学习；产品边界则在真实 Bad Case 中持续收紧。

01 SOURCE OF TRUTH

SQLite
音乐身份 / 偏好 / 推荐 / 反馈
→ Projection
→ Obsidian / UI
人类可读视图

界面可以重建，但音乐身份、偏好、反馈与推荐历史只能有一个权威来源。

02 MCP ADAPTER

Claude Code / Codex
→ MCP local stdio
→ SharedAgentService
→ Music Agent Domain

MCP 负责工具投影、身份、请求与边界；不复制数据库，不重写推荐逻辑。

03 CAPABILITY BOUNDARY

- READ
  查询状态 / 歌曲 / 推荐历史
- MUTATE
  生成推荐 / 记录反馈 / 应用学习
- VISIBLE / AUDIBLE
  Preview / Play / Apple Music
- INTERNAL
  不开放内部写入与修复接口

04 BAD CASE → ITERATION

推荐重复：历史推荐与反馈纳入候选过滤

试听 / 播放混淆：收紧 tool 语义与路由边界

“第二首不错”：先解析 run + candidate，再记录反馈

产品化问题：启动入口、大小窗口切换、演示稳定性持续优化

PROJECT EVOLUTION

单次推荐 → 持久化音乐状态 → 反馈与学习分离 → 个性化推荐 → 真实试听 / 播放 → MCP 对外连接 → 产品化迭代

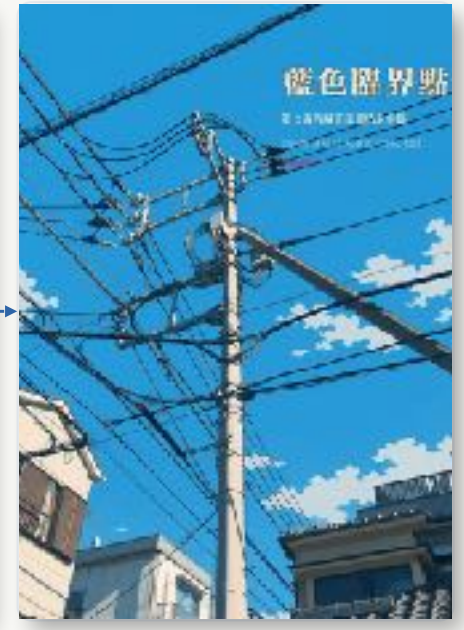
3686 TESTS

EVA Photo Redraw｜从 Prompt 到 Skill

目标是把照片重绘中的分析、风格、文字规划和检查过程变成可复用流程。

原始照片

SOURCE PHOTO



Skill 输出结果

V1 OUTPUT

问题重定义

早期做法可以通过长 Prompt 描述风格、字体与构图要求，但实际输出仍不稳定：有时退化为普通海报，有时技术信息漂浮，有时为了满足风格特征而机械加入竖排、边框或伪代码。

开发中逐渐发现，问题并不是 Prompt 写得不够详细，而是**图片分析、风格判断、文字规划、生成与结果检查等不同职责被混在同一次生成里**。一旦其中某一步判断错误，很难定位失败发生在哪里，也很难只修正其中一个环节。

因此，项目的核心问题从：

“怎样写一个更完整的 Prompt？”

转向：

“怎样把生成任务拆成职责清晰、可以检查和针对性修正的流程？”

核心判断：要解决的不是 Prompt 长度，而是生成过程缺少可追踪的控制链。

一次性 Prompt 的问题：能描述，但很难稳定控制

实际开发中，失败并不只是“画得不好看”，而是多个责任混在同一个 Prompt 里，导致结果不可追踪、不可复用。

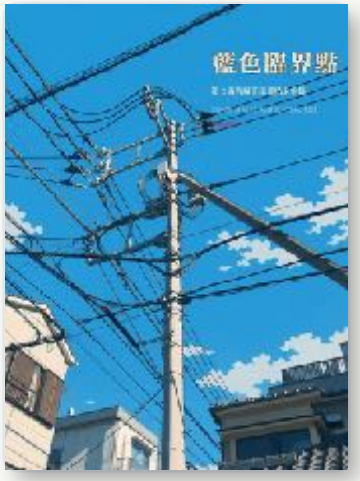
典型 Bad Case

开发记录中反复出现普通左上角标题、主副标题脱节、Technical 漂浮、Warning 像批注、大框/面板式拼贴、标题过小、机械竖排和 2+2 方块式主标题。

这类问题说明：生成模型可以模仿表层视觉元素，却不能天然保证主体关系、文字层级、版面轴线与信息授权。

所以后续重点不再是继续增加 EVA 特征词，而是把“风格判断、文字规划、布局校验、生成评价”拆分成不同阶段。

不是 Prompt 长度问题，而是控制链问题。



TV-CEL

COMMAND

RED

HYBRID

规则过硬会机械套版

把“竖排、框线、横线、切边”写成 checklist 后，Director 会为了凑条件输出拼贴式版面，而不是做真实构图。

Style 不能只靠换色

如果差异主要来自蓝天、暗部、云量或饱和度，TV、Cinematic、Command 会像同一底图的调色变体。

文字不能随意生成

Command 和技术感不能通过伪造编号、坐标、状态码来补密度；copy authorization 必须优先。

解决思路：把一次生成拆成可检查、可修正的 Skill 流程

EVA v1 将视觉判断交给 Agent，把可以明确计算和验证的规则交给确定性 Runtime。

01 **INPUT** 原图输入 → 02 **ANALYZER** 主体/结构/负空间 → 03 **DIRECTOR** Style 与改图策略 → 04 **TYPOGRAPHY** 文字规划 → 05 **COMPILER** 生成请求 → 06 **GENERATION** 外部图像生成 → 07 **EVALUATOR** 结果检查

Agent 负责视觉判断

原图进入系统后，不再直接生成。Analyzer 先识别主体、结构、透视关系与可用空间；Director 再决定适合的 Style 与构图策略；Typography 阶段规划标题、副标题与图像之间的位置关系，最后由 Evaluator 对生成结果进行检查。

结果出现问题时，可以定位到图片分析、Style 选择、Typography 规划或图像生成中的具体环节，而不必重新修改整段 Prompt。

过程证据

Runtime 负责确定性约束

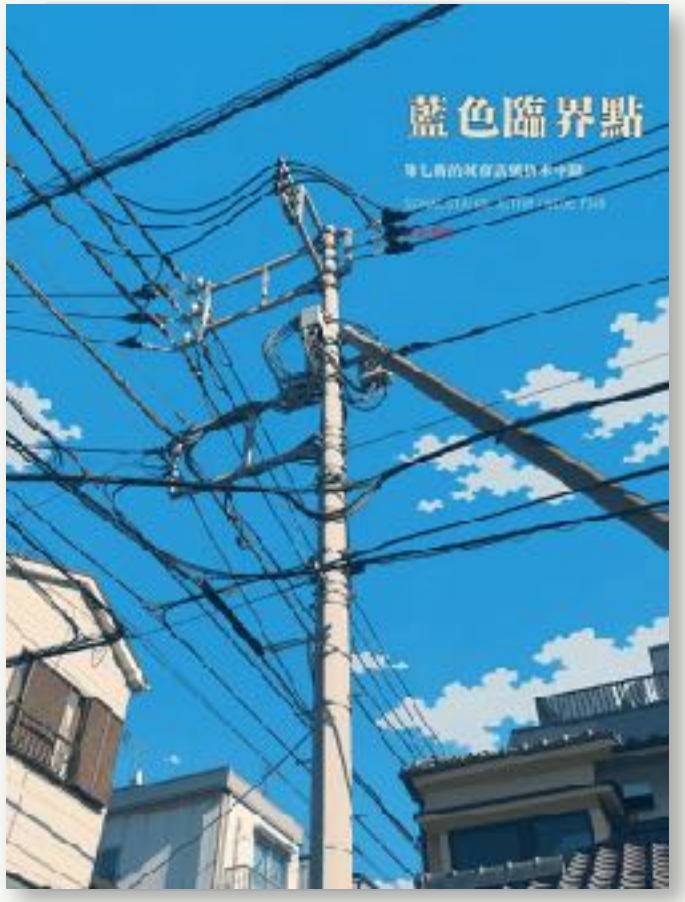
并不是所有问题都需要交给模型判断。对于可以明确计算的规则，例如真实字宽、文字碰撞、安全区域、主体遮挡与字体调用，EVA 将其交给确定性 Runtime 检查。

如果某个排版方案违反这些规则，系统会直接判定方案不合格并返回重新规划，而不是在合成阶段偷偷移动或缩放文字。

Agent 负责需要判断的事，Runtime 负责不能只靠“尽量遵守”的规则。

关键落地：Typography 从字体装饰变成构图合同

EVA v1 最成熟的部分不是“生成图像本身”，而是把文字系统做成可执行、可验证的机械约束。



Typography 不是后期贴字，而是参与画面结构。

从“排版建议”到“可执行约束”

问题

早期 Typography 主要依赖 Prompt，容易出现普通海报式标题、意外换行、文字碰撞和遮挡主体等问题。

改造

为了减少排版结果对 Prompt 的依赖，v1 将部分关键规则做成可执行检查：包括主标题层级、多行排版、真实字宽、文字关系、安全区域、碰撞与主体保护等，使生成结果更稳定、可验证。

结果

不再只要求模型“尽量遵守”。不合法的排版方案会被直接拒绝并返回重新规划，使结果更稳定、可验证。

文案与字体

将原始文案、最终展示文案和实际使用字体统一记录，减少繁简转换、字体替换和不同机器环境造成的不一致。

排版约束

将标题层级、文字关系、轴线、安全区域、裁切和元素边界纳入统一规则，减少漂浮、碰撞、意外换行等排版问题。

主体保护

为主体、人脸、手部和关键物体设置保护区域；文字一旦遮挡关键内容，方案会被判定为不合格。

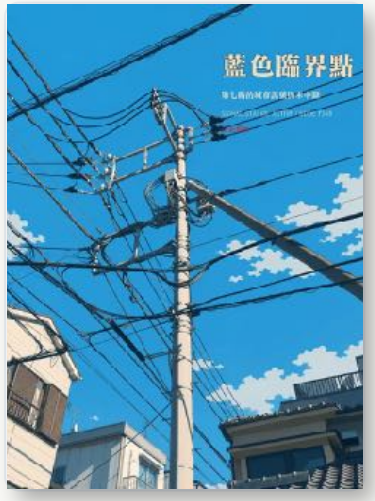
结果校验

合成器只负责检查并拒绝不合法的排版，不会偷偷移动、缩放或修改设计；问题会返回上一步重新规划。

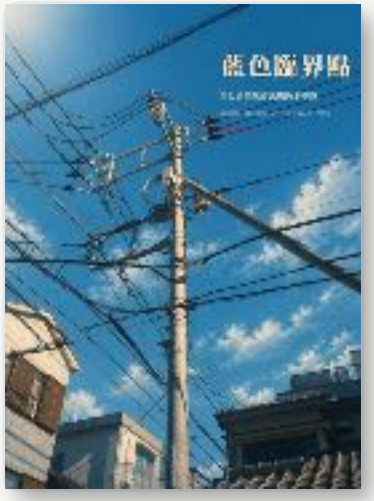
85 项自动化测试通过 | 覆盖字体、排版、碰撞、安全区与主体保护

V1 收口：把不稳定的生成能力，沉淀成可复用流程

一个可复用 Skill 工作流 + 已测试的 Typography Runtime。



TV Cel



Cinematic

Command

Red

最终形成的系统

EVA 没有把所有视觉决策都写死成程序：风格与构图仍由 Agent 判断；而字体、排版碰撞、安全区域和主体遮挡等可明确验证的问题，则交给确定性 Runtime 检查。这样既保留生成式 AI 的灵活性，也降低了关键排版规则失控的风险。

项目价值

EVA 将原本依赖反复试 Prompt 的图片重绘任务，整理为可复用的 Skill 工作流。面对风格不稳定、文字失控、主体遮挡等问题，可以定位到具体阶段并针对性修正，而不是重新从头生成。

关键推进

从真实 Bad Case 出发，持续拆解问题并调整产品机制：重新定义 Style 与 Typography 的职责，引入生成前后的检查流程，并将字宽、碰撞、安全区和主体保护等可判断规则推进到确定性 Runtime。

验证结果

EVA v1 采用“Agent 判断 + Runtime 校验”的结构：图片分析、风格和构图仍由 Agent 完成；字体、字宽、碰撞、安全区域和主体保护等机械约束由 Runtime 执行。当前 85 项生产测试验证的是 Typography Runtime，而不是整套图像生成链路。

当 AI 项目开始长期运行，知识本身也需要被管理

它解决的不是“文件存在在哪里”，而是 AI 在大量历史材料中如何判断哪些信息更可信、更当前。

问题重定义

随着 AI 项目持续迭代，同一个项目会不断产生不同性质的资料：早期记录、Phase Summary、Verification、Retrospective、Current State、正式规则等。它们并不是简单的“更多文档”，而是分别处在不同时间、成熟度和权威等级。

例如，一个长期运行的项目可能同时保留早期实验方案、已经废弃的实现路径、阶段性结论、验证证据，以及当前真正生效的规则。这些内容都有保留价值，但并不意味着它们应该以相同优先级进入 AI 的 Context。

如果 AI 每次处理任务时都直接搜索全部历史资料，两个问题会逐渐出现：

一是旧知识干扰当前判断。
已经被新方案替代的历史结论仍可能被检索出来，与当前规则同时进入上下文。

二是同一来源被重复消费。
如果系统不知道一份文档是否已经完成知识提炼，就可能重复处理同一材料，产生重复知识或不同版本的相似结论。

因此，当项目规模开始增长以后，真正的问题已经从“AI 能不能找到资料”，转变成：

AI 怎么知道哪一份资料现在更值得相信?

产品判断

Knowledge Runtime 不把 Vault 设计成一个“文件越多、搜索越全越好”的知识库，而是在现有文件系统之上增加一层知识运行机制。

它需要持续判断：哪些内容已经成熟、哪些内容已经处理、哪些内容发生了实质变化、哪些内容需要人工复核，以及 AI 在真正使用知识时应该优先读取哪一层信息。

因此，Knowledge Runtime 管理的不只是“知识放在哪里”，还包括**来源边界、处理状态、增量更新、复核机制和读取优先级。**

核心问题：AI 缺的不是更多资料，而是一套判断“什么是当前知识、什么只是历史证据”的机制。

问题演化

真实例子：Reflection System

当前生效规则放在 00_System；开发历史、POC、Closeout 与验证记录放在项目档案中，避免“现行规则”和“历史证据”等价读取。

00_System
当前规则

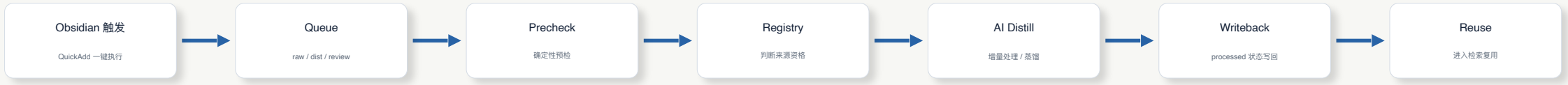
04_Projects
历史证据

Registry
来源边界

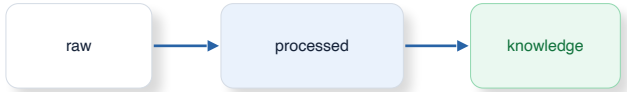
历史保留为证据，但不能默认与当前知识同权进入 Context。

从“新增文档”到“可复用知识”，每一步都有状态和边界

Runtime 先判断状态、来源资格和运行条件，再决定是否调用 AI，避免重复消费与错误写入。



状态与处理历史



- **status** 描述笔记本身处于哪一层：raw → processed → knowledge。
- **knowledge_status**记录“这份文件是否已经作为知识来源被 Runtime 完整处理过”。即使没有产生新 Knowledge，只要已经完成判断，也会写回 processed，避免下一次重复消费。
- 出现冲突、过时或来源不清时，不让 AI 自行裁决，而是进入**needs_review** 等待人工复核。

运行可靠性：先判断，再调用 AI

NO-OP

预检没有候选时直接退出，不调用 Claude，不消耗 token。

RUN LOCK

已有 Runtime 运行时，重复触发直接退出，避免并发写入。

JOURNAL

RUN_BEGIN / SOURCE_COMMIT / RUN_DONE 留下可追踪记录。

失败时不写入 processed；源文件状态是处理历史的正式依据，Journal 只提供运行证据。

Registry：先定义什么内容有资格进入

Runtime 不是扫描整个项目，而是由 Registry 指定“哪个项目、哪个目录、满足什么成熟条件”才可以进入知识蒸馏。

Music Agent

仅达到 SEALED / 封存条件的 Phase Summary 自动进入蒸馏；Checkpoint 只在需要追溯证据时下钻，不自动成为知识。

EVA

Retrospective 与 Verification 分别注册为独立成熟来源；未处理内容进入候选，而不是扫描整个项目目录。

来源边界先于 AI 处理

不是让 AI 读得更多，而是让它优先读对

知识沉淀完成后，检索不做全库无差别搜索，而是按权威层级逐层下钻。

固定检索链：上一层足够即停

这条链解决的不是“AI 找不到文件”，而是面对大量历史资料时，AI 不知道哪些信息更应该被优先相信。

为什么这样读?

当前知识优先

已提炼、当前有效的 Knowledge 先进入 Context，避免历史版本和当前规则同时参与判断。

上一层够用即停

检索不是为了“搜全”，而是为了找到足够可靠的信息；上一层已经能回答，就不继续下钻。

历史作为证据

Projects / Raw 保留，但只在项目任务或需要溯源时读取，不默认进入日常 Context。

它不是删除历史，而是给历史设置读取边界。

Case Study：Reflection System

同一个项目同时存在当前正式规则、当前状态、阶段结论、验证证据与早期历史。Runtime 不删除这些材料，而是让它们拥有不同的读取优先级。

```
knowledge_status: processed
knowledge_processed_at: 2026-08-30
knowledge_runtime_version: v2.1
```

已经处理过的项目材料默认不重复消费；只有发生实质更新时，才重新进入增量判断。

CLOSING / PRODUCT THESIS

让 AI 能力落到产品闭环

确定性产品 · 有状态 Agent · 可控 Skill · 可靠 Runtime

感谢阅读，期待进一步交流。

FOUR PROJECTS / ONE METHOD

01 **Store Price Checker**

DETERMINISTIC PRODUCT

02 **Music Agent**

STATEFUL AGENT

03 **EVA Photo Redraw**

CONTROLLABLE SKILL

04 **Knowledge Runtime**

RELIABLE RUNTIME

z987535@gmail.com

SUZHOU, CN