腾讯学堂 · 60 分钟

# AI 原生思维

从 AI 产品需求、边界到落地——以及我们每个人如何与 AI 共同进化

宝玉 • 2026 年 8 月

开场

# 今天回答两个问题

问题一 · 上半场

## AI 产品到底怎么做?

从找需求、判边界，到重设计、落地验证

问题二 · 下半场

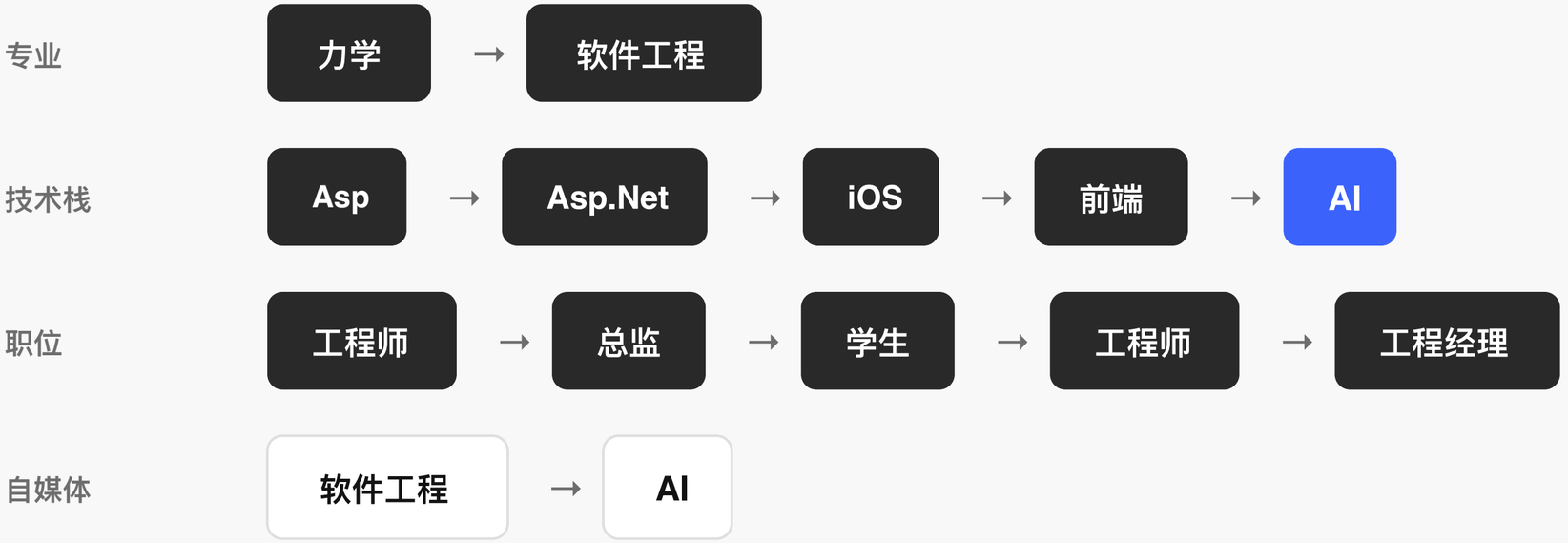
## 我们会不会被 AI 替代？该学什么?

跟在座每个人都有关——不只是做产品的人

所有答案来自同一个地方：过去三年，我用 AI 做产品、也被 AI 重塑的亲身经历。

自我介绍

# 一部不断转型的历史




**AI 来了，我知道该干嘛：学习 + 转型。**

一条暗线 · 贯穿全场

# 像训练大模型一样，训练自己。

建立反馈循环 → 跟着模型一起进化 → 敢于推翻自己的旧权重

上半场 · 41 分钟

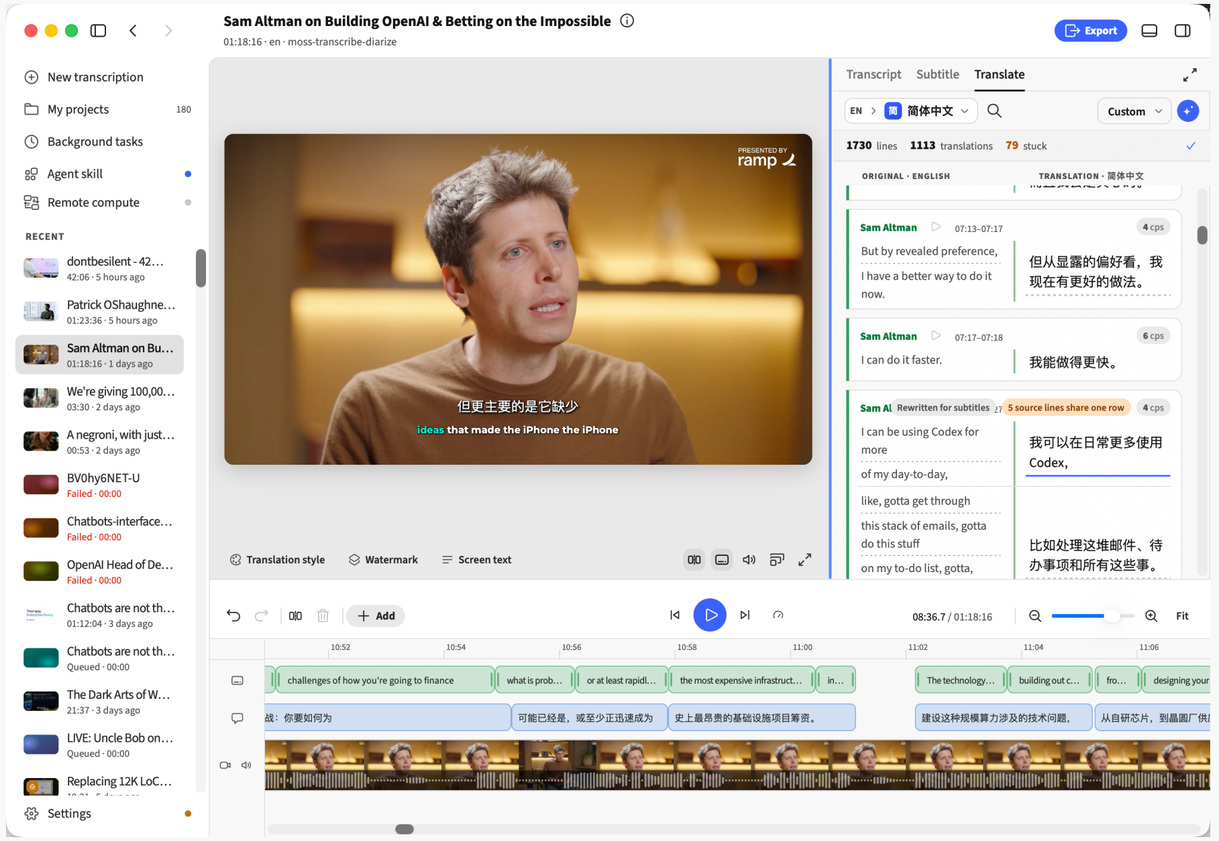
# AI 产品思维：从需求、边界到落地

一 找需求 → 二 判边界 → 三 最小验证 → 四 重设计 → 五 落地验证

上半场 · 主案例

# BaoCut：一个字幕转录翻译 App

- 从第一天做到今天：每个判断为什么做、每个坑为什么踩，都能讲出当时的真实想法
- 起点：从美剧《24 小时》认识字幕组——又快又好，能不能把我文章翻译的经验迁移过来？




**别人的案例只能讲结果，自己的案例才能讲决策。**

一 · 找需求

# 判断一个 AI 需求的三问

| | | |
|---|---|---|
| 1 | **痛点够不够硬?** | 一集剧的字幕 = 十几人的字幕组干几小时；痛到有人长期免费干，不是伪需求 |
| 2 | **我是不是用户?** | 自己每天用，反馈循环最短。反例：一小时做出漂亮的 X 客户端原型，但我没有每天想用的冲动——放下 |
| 3 | **AI 是不是刚好够到?** | 需求本身不新（字幕翻译几十年了），"AI 刚好够到"才是新的：够不到是半成品，早就够到是红海 |

快速原型的一大价值，恰恰是让坏想法死得早。

一 · 找需求 · 第三问的另一个案例

# 一个画图提示词：先死，后活，走过三级

GPT-4o 时代

**自己写、自己用**

发布没人理，只有围观点赞

Nano Banana 时代

**别人也能用**

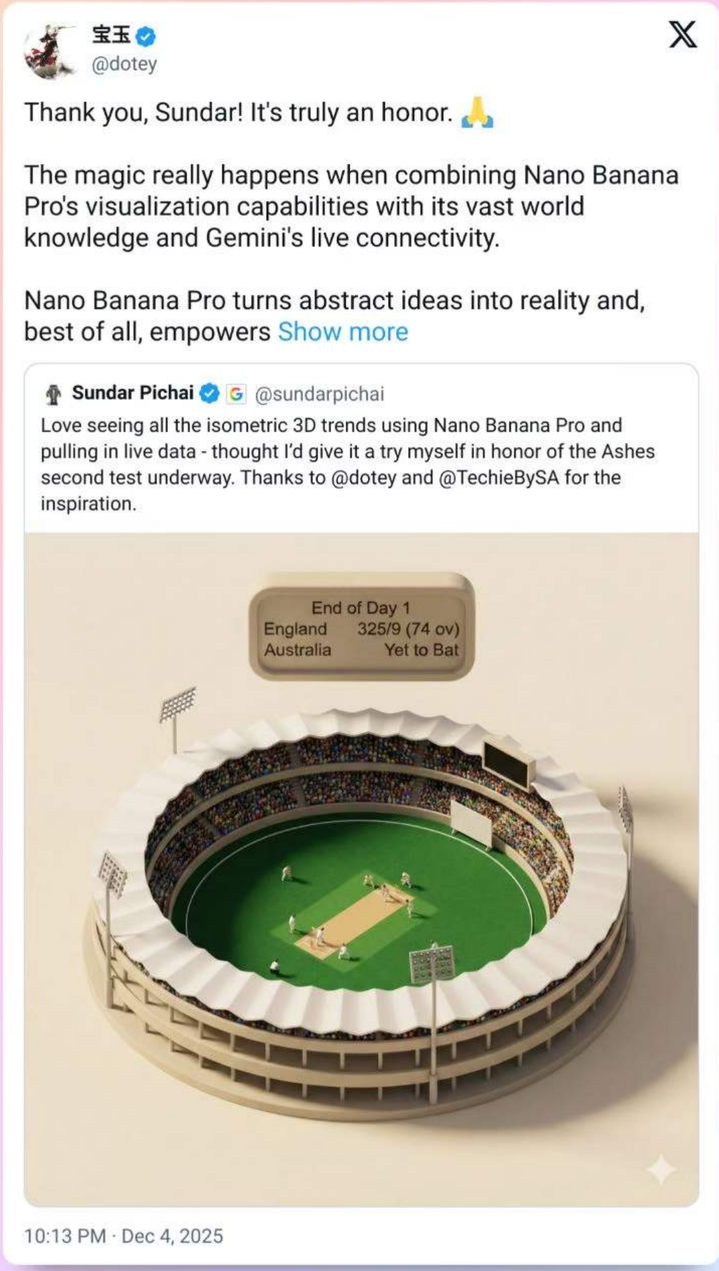
只输一个城市名，门槛降到一步

质变

**用户创作自己的作品**

自己的城市、自己的纪念日，分享回流成反馈，转到 Google CEO 发推致谢

**体验差一步，命运差十倍。**



一 · 找需求 · 两句产品课

## 01　需求一直都在，能力边界刚刚扫过它。

## 02　做出好东西靠品味，让它长大靠降门槛。

这些提示词本身也是和 AI 共创循环磨出来的——人的品味决定反馈质量。

一 · 找需求 · 收拢

# 盯着模型能力的边界线找需求。

每次模型升级，边界线外扩一圈——扫过的地方，全是新需求。

二 · 判边界

# AI 产品的三条边界

一个需求要活下来，得连闯三关——多数人只盯着第一关。

二 · 判边界 · 能力边界

# 同一个想法，边界两侧，两种命运

## AutoGPT

2023

通用 Agent 的想法，火到 GitHub 史上最快破十万 Star

但模型的 Agent 能力不够：任务循环打转，烧掉大把 Token 交付不了结果

**火一阵，就凉了**

**想法不是关键**
**模型能力边界才是**

→

## Manus

2025.3

同一个"通用 Agent"的想法

等到模型能力刚好够到的时刻，第一个把它产品化

**一夜刷屏，开创先河**

**想法太早，烧的只是 Token。**

二 · 判边界 · 能力边界

# 边界线没停：一年多，扫出四个新坐标

## Claude Code

**cli 翻红**

Agent 靠文本工作，被淘汰二十年的终端反而是主场——形态服务于能力，不服务于潮流

## OpenClaw

**Skill 交给用户**

写一个 Markdown，Agent 就能学会你的私有知识——扩展边界的权力第一次交到用户手里

## Codex App

**操作电脑过线**

Agent 看屏幕、点鼠标、打字，接管没有 API 的应用——一半用户已在做非编码任务

## "work" 潮 · 2026

**办公是第二站**

Claude Cowork、ChatGPT Work、腾讯自家的 WorkBuddy——编程里打磨的本事外溢到写文档、做表格、做研究

**每过一道能力临界点，产品形态就重新洗一次牌**

二 · 判边界 · 能力边界 · 我自己的版本

# BaoCut 的三层能力解锁

| | | | |
|---|---|---|---|
| Whisper | → | **机器听写** | 解锁前：字幕只能人听 |
| 大模型 | → | **机器翻译、拆字幕** | 解锁前：只能人翻译、人拆 |
| Agent 能力 | → | **自主执行、自主验收** | 解锁前：人全程盯着验收 |

**每解锁一层能力，产品就能重设计一次。**

# 落成规则：产品三问

| | | |
|---|---|---|
| 现在能做 | → | 马上做 |
| 半年后能做 | → | 现在搭架子等模型 （模型进化是你的顺风） |
| 很久做不了 | → | 工程补，或者别做 |

---

**留给你的问题：你的领域里，哪个"AutoGPT"正在等它的"Manus 时刻"?**

二 · 判边界 · 成本边界

# 账单会直接杀死产品

功能上用户点赞，账单上用户跑掉。

它会杀产品

### AutoGPT 死了两次

能力不够是一次，"烧大把 Token 换不来结果"是另一次。BaoCut V2 也挨过这刀：“功能是好，Token 烧太快"——用户直接不用了

它塑造行业

### RAG 是成本边界催生的品类

全部文档塞进上下文，能力上早就可以，成本上不行。长上下文降价，"RAG 还需不需要"立刻变成行业争论

它也在动

### 动一次，洗一次牌

DeepSeek 把推理成本打下来一个量级，"烧不起"的场景一夜成立；OpenClaw 那种 7×24 挂机的形态，也是成本线外移后才敢有

**同等能力的 Token 价格一两年降一个量级——成本线外扩扫过的地方，和能力线一样，全是新需求。**

# 一次真实的成本优化

**33 → 12**

模型调用次数

**31 → 18 分钟**

单集处理时长

**7 小时 → 42 分钟**

访谈完整润色

根因　让模型输出的 JSON 太长。翻转取舍：模型输出简单纯文本，复杂解析交给程序。

**用户量放大一万倍，账单还成立吗?**

**模型再降价十倍，哪些今天烧不起的功能会成立?**

二 · 判边界 · 价值边界

# 能力够、账单成立——用户还买吗?

行业案例

**Klarna**：AI 客服顶替七百人，成本账漂亮极了；一年多后 CEO 承认砍过头、重新招人——用户要的服务里包含"有一个人对我负责"

同一个我 · 字幕翻译

## 全部交给 AI

用户买的是准确的字幕，没人在乎是谁翻的

同一个我 · 写作

## 一个字一个字收回来人肉写

读者买的是"宝玉怎么看"——AI 味一出来，价值就没了

**用户买的是结果，还是结果背后的人?**

这条线还有一个个人版——什么活该交给 AI、什么必须留给自己?今天最想让你带走的一句话，下半场揭晓。

二 · 判边界 · 小结

# 两条会动的线上找机会，那条不动的线内侧建护城河。

今天

## AI 已经够到

能力和账单都成立——正当时，或已是红海

能力线 →

成本线 ⇄

会动

明天 · 线一动就扫过这里

## 新需求区

模型升级、价格下降扫过的地方，全是新需求

在这里找机会

价值线 ⊣

不动 锚在人性

永远 · 线扫不进来

## 用户买的是人

观点 · 责任 · 关系 · 品味——交给 AI 就是自毁

在这里建护城河

技术进步的方向 →

三 · 最小验证

# 先让它活下来：BaoCut 差点死掉

Whisper 时间戳对不准

视频编辑器工作量巨大

业余时间老被打断

→ 眼看进"半成品坟场"

破局

## Claude Design 高保真原型

不写代码，分钟级做出可交互原型，马上发现问题马上改。MVP 决策：只做 Mac、只做最简单的播放和字幕

关键选择

## 用我完全不熟的 Swift

AI 实施、我指挥验收——正因为不熟，才能真正放手。在不熟悉的领域开始，反而更接近 AI 原生

三 · 最小验证 · 一句话

# 最小验证是需求和边界判断的"实弹测试"——不只让好想法长得快，更让坏想法死得早。

| | |
|---|---|
| X 客户端原型 | 死得早，只花一小时 |
| **BaoCut** | 被原型救活 |

四 · 重设计 · 上半场高潮

# 不要让 AI 模仿人

接下来的故事：一个做废了一半的产品，和推倒重来之后学到的三条准则。

四 · 重设计 · V1（2023–2024 的直觉做法）

# V1：模拟人类字幕组


- 替代了大量人工，但 LLM 自作主张合并字幕、时间戳乱、人工校对依然繁重
- V1.5 用工作流编排和算法打补丁——有效，但收益递减

**在旧框架里打补丁，收益递减。**

四 · 重设计 · V2 · 推翻旧权重（第一次）

# V2：推倒重来，为 Agent 重新设计

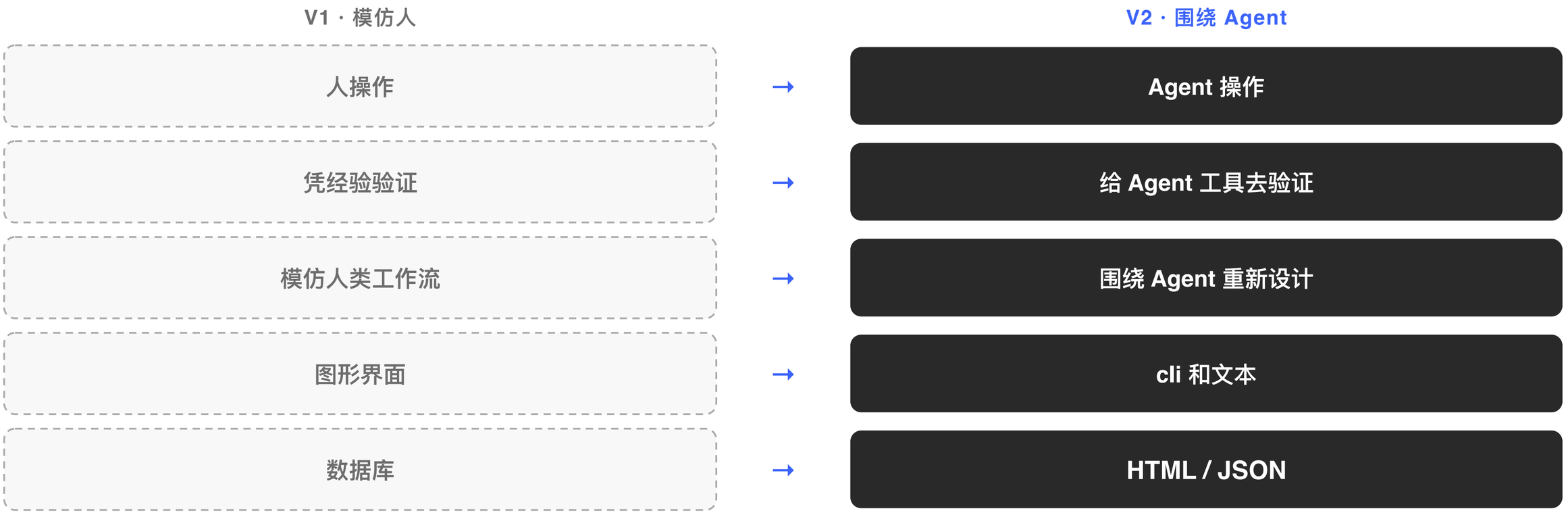
不问"原流程哪一步还能优化"——以终为始：我要的结果到底需要什么？三连追问：

| | | |
|---|---|---|
| **为什么对不齐?** | → | 中英语序不同，srt 级对不上但句子级可以——为什么非要先拆 srt？因为在模仿字幕组 |
| **为什么翻译质量差?** | → | 分页割裂上下文——先整体分析、提术语表，程序注入全局信息 |
| **为什么这么多人工?** | → | 在模仿字幕组每步人工校验——每步给 Agent 验证工具，Agent 自己验收，人只做最后兜底 |

效果：流程节点大幅缩减，人工从每步盯守变成最后验收。

四 · 重设计 · 一页看懂

# 模仿人 vs 围绕 Agent 重设



# 准则一 & 准则二

1

## 让 Agent 操作它最擅长的格式

训练数据里管够的 HTML、Markdown、纯文本。选型先问一句：这个格式，模型见得多吗?

2

## 给设计规范，不给模板

模板是专家捷径，规范 + 模型能力是笨办法——"苦涩的教训"：通用方法终将战胜专家捷径。模型每升级，产品不改一行代码自动变强——这就是"搭架子等模型"的具体做法

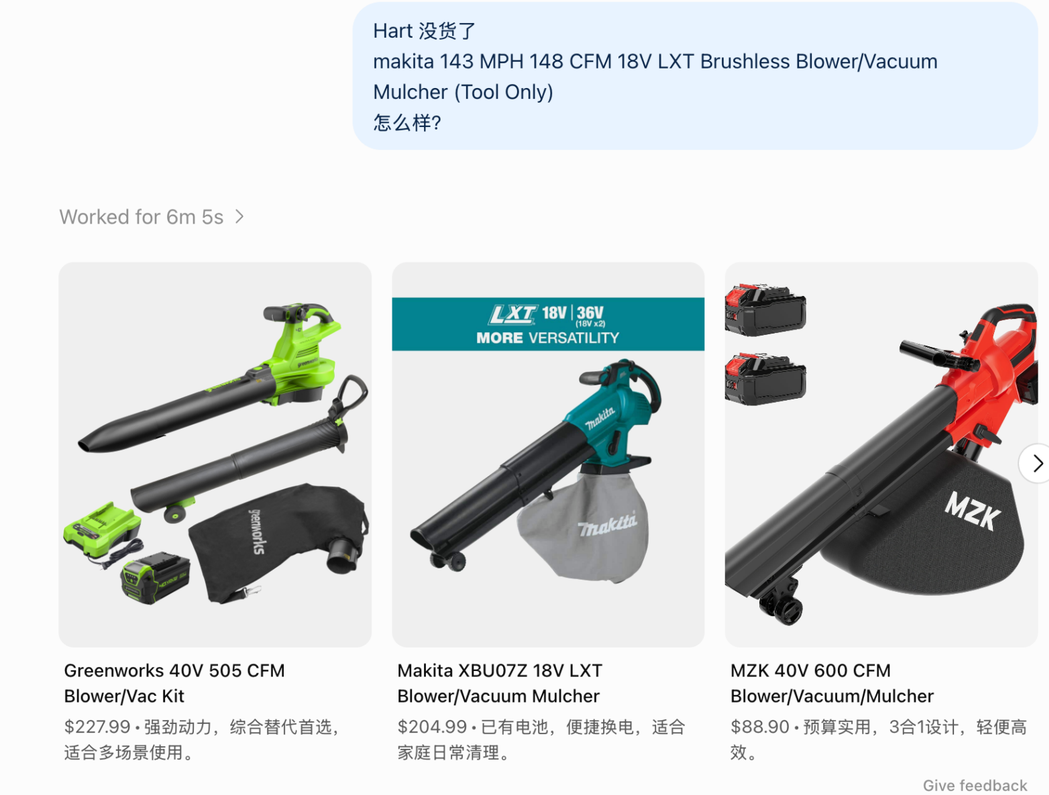
四 · 重设计 · 准则三：把 App 做成 Agent 的插件

# 变化一：用户先打开的，变成了 Agent

买一台吸树叶的机器：我没有逛电商，先让 Agent 帮我找。把我看中的一款发给它，它告诉我为什么它选的更好——吹叶和吸叶的参数是两回事、裸机不含电池的真实性价比、电池生态锁定……

分析深度远超我自己刷商品页。

**当购买决策发生在 Agent 的对话里，你的商品详情页就变成了 Agent 的数据源——Agent 读不懂你，你就错过这一单。**

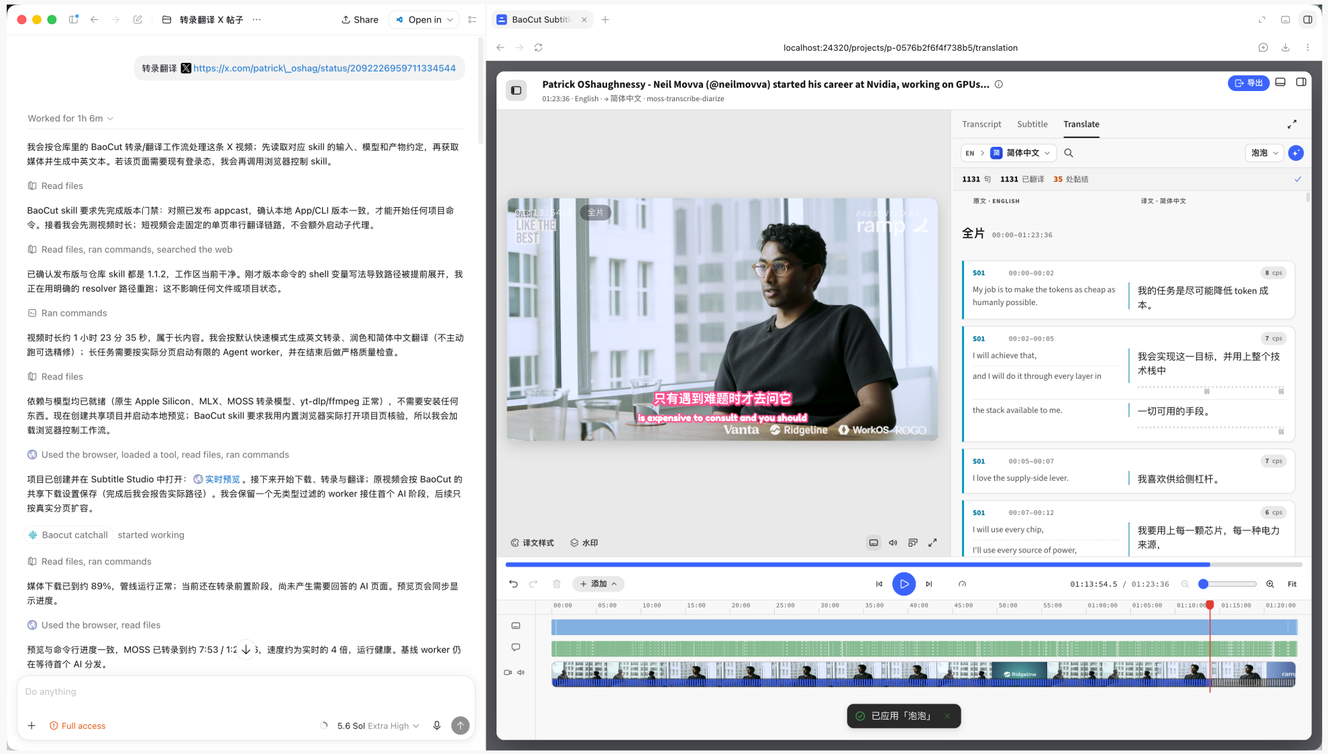


# 变化二：人不操作了，Agent 操作，人只确认

做 PPT，我从自己在 Google Slides 里拖拽，换成了让 Claude Code 写——当然，大纲是我的。

最诚实的信号：BaoCut 是我自己写的 App，连我自己都不打开它的界面了——发一个 YouTube 链接给 Agent，它调用 BaoCut 的 cli 转录翻译，我只看结果。

**作者本人都绕过了自己的图形界面——这就是入口迁移最诚实的证据。**



四 · 重设计 · 放进历史里看

# 入口的第三次迁移

第一次

**门户 → 搜索框**

谁不在搜索结果里，谁消失

第二次

**PC → 手机 App**

没有 App 的产品，淡出视野

第三次 · 正在发生

**App → Agent**

用户从"打开你"变成让 Agent"调用你"——你不在 Agent 够得着的地方，你就不存在

**我踩过的坑**

Figma 稿生成 UI 代码的 Agent，功能完整没人用；改成 Skill 让开发者自己的 Coding Agent 调用，问题消失。一年后 Figma 官方做了同一个选择：不内置 Agent，开放服务让 Cursor、Claude Code 直接读设计稿。

四 · 重设计 · 收拢

搜索时代 · 拼 SEO → App 时代 · 拼买量 → **Agent 时代 · ?**

# 做好被调用（skill / cli / 机器可读），就是 Agent 时代的新 SEO。

你的产品，是抢着当入口，还是做好被调用?

五 · 落地验证 · 上周的真实案例

# 从一条 GitHub Issue 到功能上线

BaoCut 远程转录：用户想把办公电脑上的转录，丢给家里有显卡的电脑跑。三问快速过：值得做。但我没有打开编辑器。

五 · 落地验证

# 五关：流程没变，执行主体变了


执行主体全是 Agent；蓝色标注 = 人的位置。对照一下：你团队的流程里，人现在站在哪几格?

五 · 落地验证 · 第一关

# 可行性：先决定要不要做

需求发给 Claude Code → 它给出四个方案 → **我扫一遍，拍板方案 A**

**模式：Agent 分析罗列，人判断拍板。**

诚实的反例

"本地模型辅助拆分字幕"，可行性分析也做了，做完体验很糟砍掉，浪费好几天和一堆 Token——可行性分析是期望值的赌注：不保证不看走眼，但没有它，坑只会更多。

五 · 落地验证 · 第二关

# 设计文档：Agent 时代的接力棒

反直觉：传统开发里文档是负担，AI 原生开发里文档是一等公民。

职能一 · 给人

## 确认修改的载体

在文档上改，成本远低于写完代码再推翻

职能二 · 给 Agent

## Session 之间传上下文

新 Session 一句"按 docs/remote-transcription.md 实现"就能开工

需求 → 设计 → 原型 → 代码 → 测试

制品链本身就是审计记录

五 · 落地验证 · 第三关

# 高保真原型：需求、原型、UI 三合一

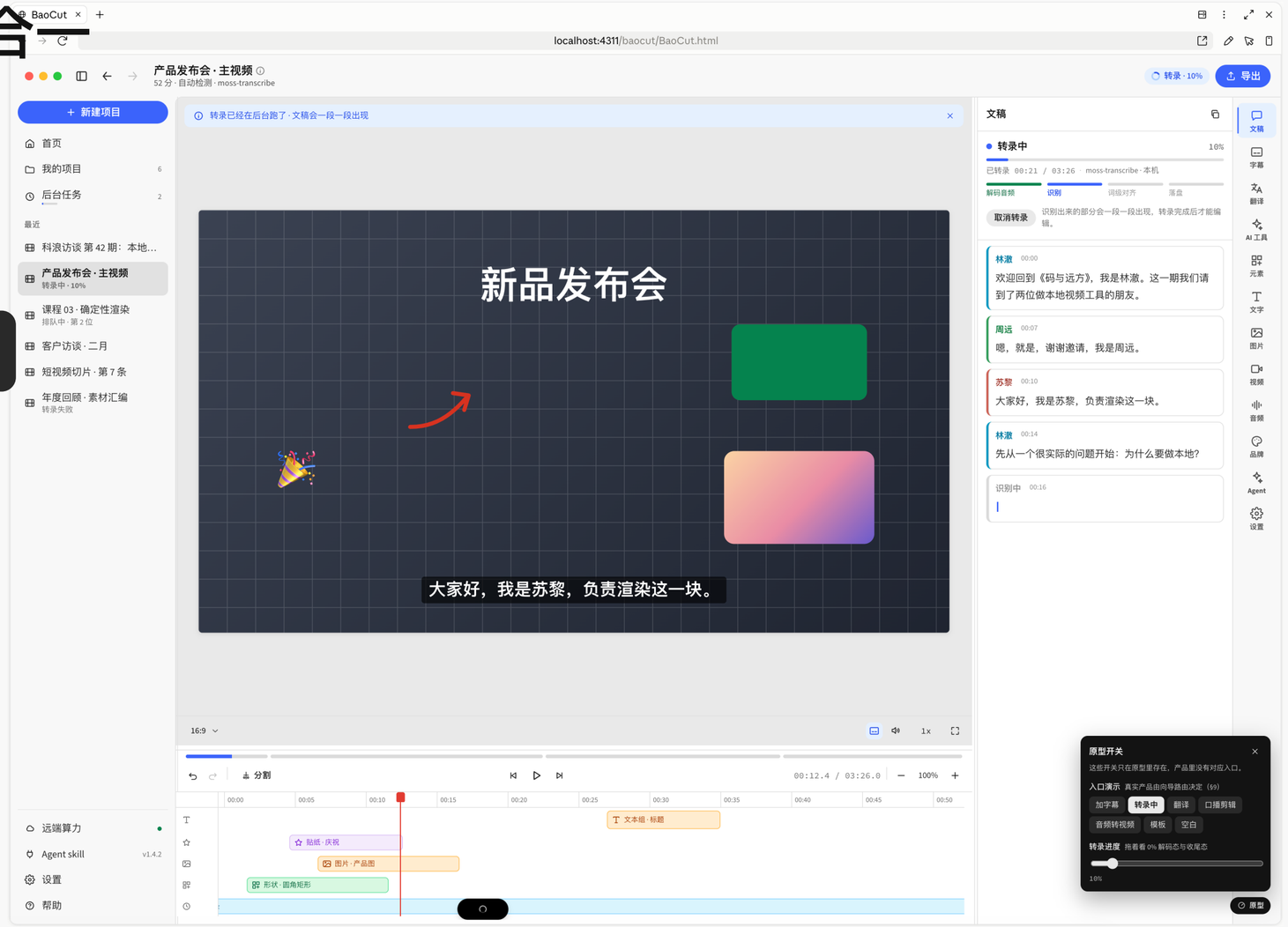
PM　交互　UI　→　一个 Agent + Design Skill

1　布局列表改 Tab

2　加状态图标

3　发现埋在设置页太深，整个挪到主界面

**原型阶段改布局是一句话的事，开发完再改成本指数级上升——这一步能磨就磨。**



五 · 落地验证 · 第四、五关

# 实现与测试：人主要在等

实现

## 文档 + 原型交给 Agent

按里程碑推进——自己写码、跑测试、截图验证。给 Agent 自己的反馈循环，人拿到的已经是自验证过的结果

测试

## 把自己当普通用户

不当开发者：凭直觉乱用，问题扔回给 Agent

**"那你 review 代码了吗？"**

这个问题先按下，下半场交代 →

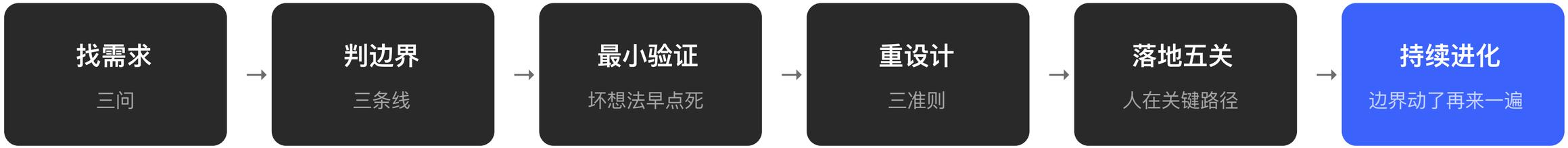
五 · 落地验证 · 复盘

# 到底什么变了：四句话

| | |
|---|---|
| **流程没变，角色变了** | 五关一个没少；执行主体全是 Agent，人从一线程序员变成技术总监 |
| **确认可以合并，不能省略** | 加速确认，不跳过确认——人的判断力是流程里不可替代的部分 |
| **验证题给 Agent，判断题留给人** | 跑测试、编译、截图比对全部 Agent 自查；人只留"方向对不对、体验好不好、值不值得做"——别让人的确认卡住 Agent 的迭代 |
| **瓶颈转移到代码两侧** | Agent 几小时写完码，左边设计确认、右边测试部署还在人类速度——开发类 Skills 大多没必要，精力花在流程设计上比 prompt 优化回报大得多 |

上半场收拢

# 一页落地地图




**结构即答案：这张图就是上半场的目录，记住目录就记住了方法。**

这张图是 BaoCut 一步步踩出来的，每一格背后都是刚才的一个故事。产品要进化，做产品的人呢?

下半场 · 17.5 分钟

# 我们每个人如何与 AI 共同进化

这半场说的就是你我：写代码的、做设计的、写文档的、做运营的。答案是三句话，每句配一段亲身经历。

六

**建立快速反馈循环**

七

**与模型共同进化**

八

**执行给 AI，思考留自己**

六 · 建立快速反馈循环 · GPT-3.5 时代

# 一个翻译提示词的六步进化


**弯路即阶梯，灵感是循环转出来的。**

这个故事还没完——结局出乎所有人意料。

六 · 建立快速反馈循环 · 分享

# 一个人的迭代到第六步就停了

直译 → 意译

读者建议

→

直译 → 反思 → 意译

分享让循环接上了别人的反馈

**放大版 · baoyu-skills 开源**

翻译、插图、发布这些重复工作全部 Skill 化开源，收到大量一个人永远遇不到的反馈：风格定制、不同 Agent 兼容。

**分享哪怕免费，也能收获金钱之外的东西：经验、口碑、反馈。**

六 · 建立快速反馈循环 · 方法

# 循环分层：周、天、分钟

| | |
|---|---|
| 周 | 写作 · 学习 · 分享 |
| 天 | 想法 · 代码 · 验证 |
| 分钟 | 原型 · 验证 |

**循环自己会转，用不着意志力——我这个转了三年没停。**

这个循环不挑职业：把"翻译"换成你手里的代码、设计稿、周报、运营文案——用它 → 观察结果 → 改进 → 分享拿反馈，循环就转起来了。

七 · 与模型共同进化 · 推翻旧权重（第二次）

# 那个广为流传的提示词，今天已经没法用了。废掉它的人——是我自己。

o1 发布，模型自己会思考——提示词里的 CoT 和它打架，效果反而变差。改成目标导向一句话："用中文重写这篇文章"。

**上半场推翻的是流程，这次推翻的是自己最出名的作品——和模型共同进化，第一步是敢于否定自己的旧答案。**

七 · 与模型共同进化

# 一个提示词的一生 = 三年人机交互进化史

GPT-3.5 时代

**人用的提示词**

直译 → 反思 → 意译

→

o1 时代

**目标导向一句话**

"用中文重写这篇文章"

→

Agent 时代

**baoyu-translate Skill**

Agent 自己调用的技能

**给个人的三问：这事 AI 现在能做吗？将来能做吗？多久？——每半年重问一次，你的工作方式就跟着模型一起进化。**

八 · 执行给 AI，思考留自己 · 上半场那个问题

# "我现在不 review AI 写的代码了。"

2000 年开始写代码，26 年——这句话说出口，比听起来难得多。中间发生了什么？

八 · 执行给 AI · 三次推翻自己

# 从不信一个函数，到不 review：四个阶段

**抵触期**

不信任，但立规矩刻意多用——避免偏见做出错误判断

**o1 之后**

写码接近超过我，架构不行——我设计，AI 写，我 review 每一行

**Claude Code**

真的理解项目了——还是 review

**Fable 5 之后**

各方面碾压——两层验证足够：Agent 自验证 + 我黑盒测试（金融核心逻辑另说）

**把自己变成 PM + EM：Token 比人力廉价，人的位置只能一直上移。**

坐 IC 位置 AI 是威胁，坐管理者位置 AI 是最强下属

八 · 执行给 AI · 推翻旧权重（第三次）

"我是写代码的" → "我是定义问题、验收结果的"

# 把"代码"换成你手里的 PPT、报表、文案、设计稿，道理一模一样——执行交给 AI，你上移到定义要什么、验收好不好。

身份的旧权重最难推，但最值得推。

八 · 执行给 AI · 反面教材

# 同一个人，两个极端

代码

## 敢不 review

有客观验证，且写码不是我的思考方式

写作

## 不敢全交给 AI

试过全套 AI：AI 味读者看得出来；更严重的是，失去了借助写作思考的过程——写作就是我的思考本身

现在的分工

翻译总结 · AI

素材 · AI 辅助

**草稿 · 人肉写**

初稿后 · AI 提意见我手改

上半场欠的那个标准，现在还上

判断标准就一个：产出是"结果"，还是"你的思考"？

# 结果可以外包，思考不行。

不要外包自己的大脑——外包大脑的人，最终会被拿回大脑的人替代。

收尾 · 开场的第二个问题

# AI 时代该学什么：三层能力

2019 年写《软件工程之美》时的答案，今天一个字不用改——AI 改不掉这三层，改变的是每层的杠杆。


三层能力 · 2019 年就写下，AI 改不掉

×

**管理 Agent，就是新的影响力**

管好一群 Agent，一个人就有一个团队的产出

×

**与模型共同进化**

一起进化的人，先看到边界线扫出的新需求

×

**快速反馈循环**

跨界循环现在转得起来——BaoCut 就是用完全不熟的 Swift 做的

AI 时代的杠杆 · 正对应下半场三句话

**担心被替代的人在和 AI 抢执行。AI 替代的是执行，放大的是能力——站在哪边，你自己选。**

收尾 · 今天出现了三次的母题

# 推翻旧权重的阶梯

**提示词**

废掉直译-意译

**流程**

BaoCut V2 推倒重来

**身份**

写代码的 → 定义问题的

**组织**

组织的旧权重就是流程和架构图：先建验证基础设施再放权，度量"想法到上线的时间"

与模型共同进化的完整含义：同一个动作，尺度越来越大。

带走两个问题

给你的产品

哪一环还在让 AI 模仿人？围绕 AI 重新设计，哪些环节会直接消失？

给你自己

AI 帮你省下来的时间和精力，你打算投到哪一层能力上？

**知 AI，善用 AI。**

谢谢 · Q&A 15 分钟