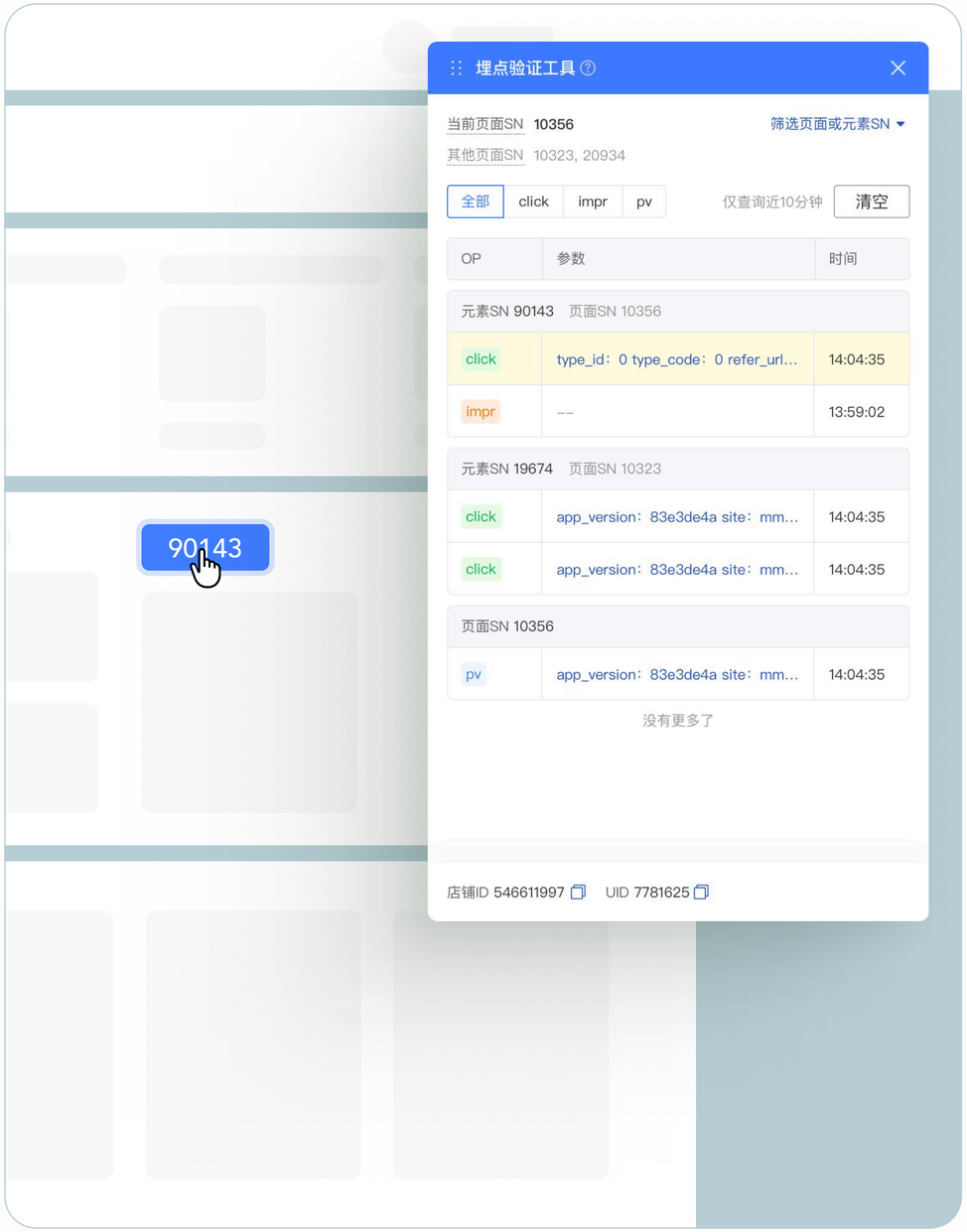


设计自驱 | Vibe Coding | 端到端交付 | 企业内部提效

埋点验收Chrome插件:
从个人工具到可靠的企业产品

即插即用的一站式H5埋点验收工具，让设计/产品在原页面快速、完整地验证埋点和参数上报，避免流程门槛导致的验收疏漏，提升验收效率，同时减少AB实验重跑

2025/02–2026/03　　Codex(GPT-5.2), Google AI Studio(Gemini 3 pro)

结果预告

产品/设计参与内测

30+人

次均埋点验收时长

-57.2%

概览

重点要讲的三件事

01

业务问题

- 埋点验收的用户, 场景, 痛点
- 为什么要现在解决这个问题

02

从Demo到可用的个人工具

- Demo快速验证核心流程
- 3个关键体验优化

03

从个人工具到可靠的企业产品

- 业务流程梳理和产品边界定义
- UI对齐平台调性
- 更多用户反馈及解决方案
- 端到端自动化测试, 持续内测和迭代

业务

埋点验收: 关键时候很重要的Dirtywork

谁验收, 为什么要验收

商家端的产品/设计, 100+人的大团队。
埋点验证是AB实验上线的强制前置条件，由对应的需求负责人(即产品/设计)执行

Dirty在于...

现有工具使用成本高

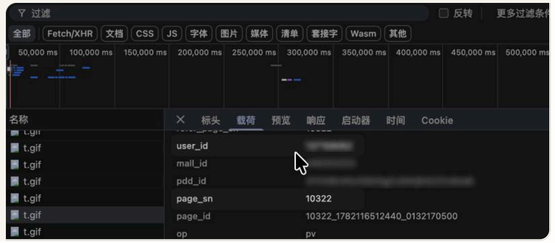
执行成本高,可持续性差: 三轮反复验证, 极其耗时, 实际执行中容易被压缩或跳过

系统性问题: 高风险任务只由一人执行和担责, 缺乏交叉验证

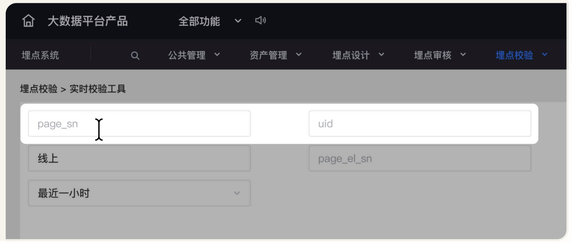
为什么解决这个问题重要?

团队对埋点验证效率和准确性要求剧增:团队AB实验量急剧增加，短期内已发生2次因埋点疏漏导致的实验重启，项目周期延长半个月～1个月，问题到了不可忽视的阈值

STEP 1 在待验收页面打开DevTools获取Page SN和UID



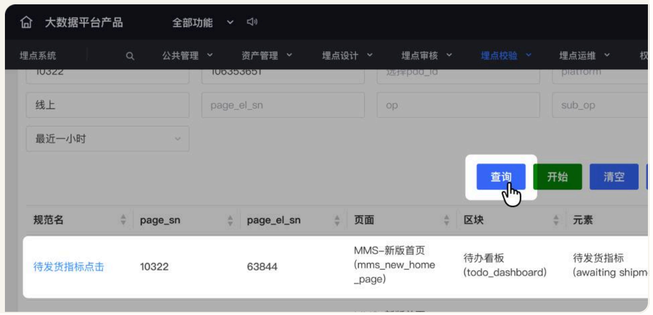
STEP 2 在YOLO(拼多多大数据平台)输入Page SN和UID



STEP 3

到待验收页面触发埋点
如: 点击/ 曝光目标埋点的热区

STEP 4 回到YOLO上点击“查询”，确认列表是否有目标埋点, 触发时间/次数是否符合预期



STEP 5 如埋点正确, 打开参数详情页看参数是否正确

业务

解决不了机制问题, 能否先让流程更高效?

我有个不成熟的想法...

已知　Devtools/Network中能实时查询到埋点上报数据

HMW　在待验证页触发埋点后, 将Devtools的埋点信息立刻回显在当前页,免去跳转\查询\权限校验等一系列漏斗?

DEMO SPRINT

快速可行性验证: 触发埋点, 查询, 看到上报, 没了

Google AI Studio

你是一位Chrome扩展开发专家，目前正在带一名电商设计师从零实现一个网页端h5埋点验证工具。

教学模式：不要一次性给出所有代码。每一步都要先解释“为什么要这么做”，然后再给出代码。

技术栈：使用Chrome Extension Manifest V3规范。UI采用原生的HTML/CSS，风格遵循Material Design

我想做一个Chrome扩展，用来验证公司h5页面里的埋点是否正常触发。

背景：
我们AB实验前需要经常验收页面pv、按钮点击click、曝光impr 等埋点是否正确上报。现在的方式主要是打开Chrome DevTools的Network，手动找到t.gif请求，再在请求参数里找 page_sn、page_el_sn、op等字段。这个流程很重复，也容易漏看，验证门槛高。

目标：
在指定业务域名页面里自动监听埋点请求。当页面触发 t.gif 上报时，扩展能解析请求参数，并把关键字段展示出来，方便我快速判断：

后来为什么从Google AI Studio迁移到Codex

1. 上下文窗口太短, 经常改动指令外的地方, 沟通成本过高
2. 运行在云端环境，没法直接增删改本地文件, 需要人工搬运代码, 操作成本高且易出错

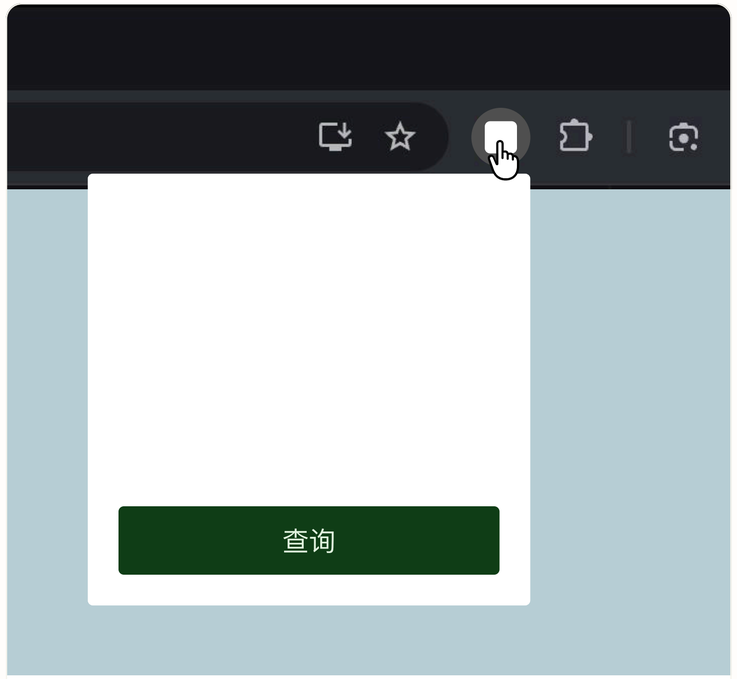
ALPHA PHASE

从 DEMO 到可用的个人工具

和设计/产品同事调研了一轮使用场景, 我做了几个关键优化

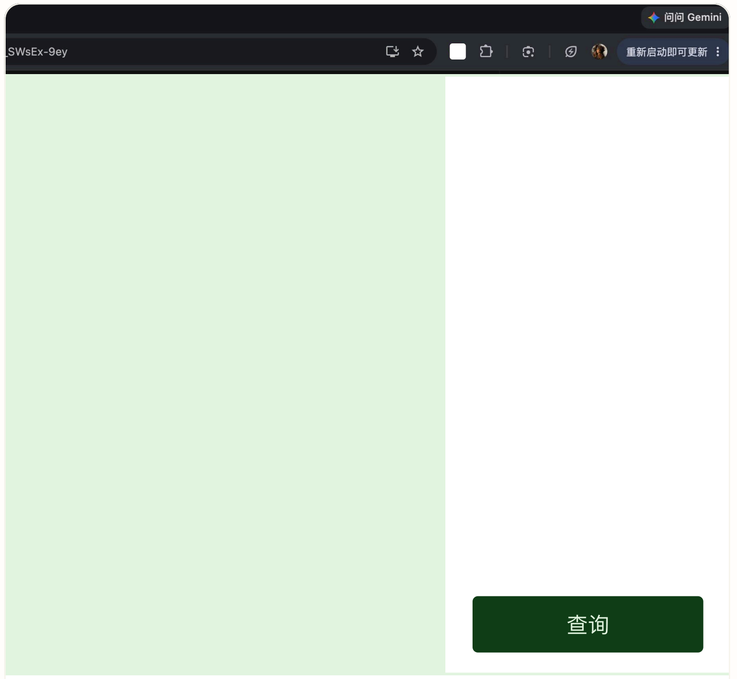
ALPHA PHASE

从临时浮层到常驻悬浮窗: 彻底消除上报和查看动作间的漏斗



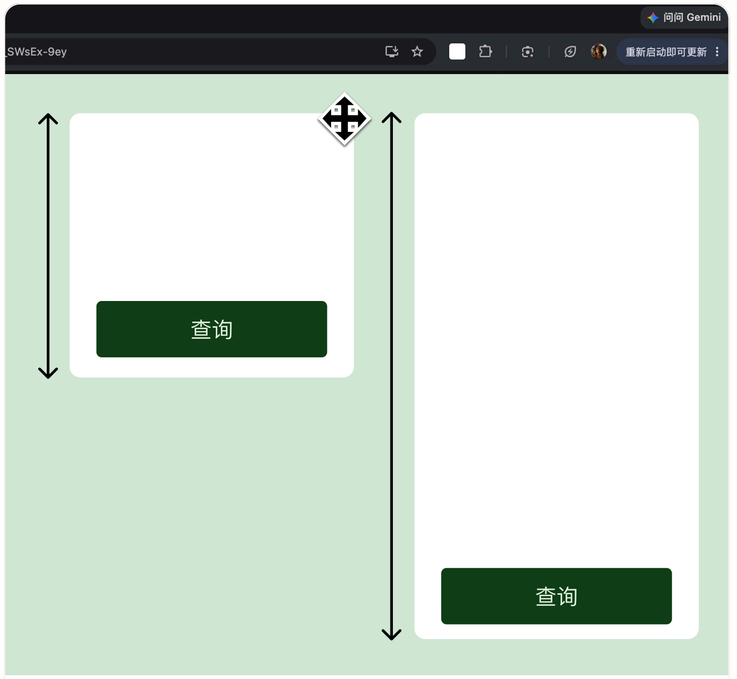
浮层

- ❌ 每次查看数据都多一步点击



侧边栏

- ✅ 触发埋点后实时看数, 无中间步骤
- ❌ 小屏(产品使用较多)受限于自适应规则, 可能遮挡部分元素, 影响对这些元素的验收



悬浮窗

- ✅ 触发埋点后实时看数, 无中间步骤
- ✅ 支持移动和自定义高度, 不遮挡页面元素

ALPHA PHASE

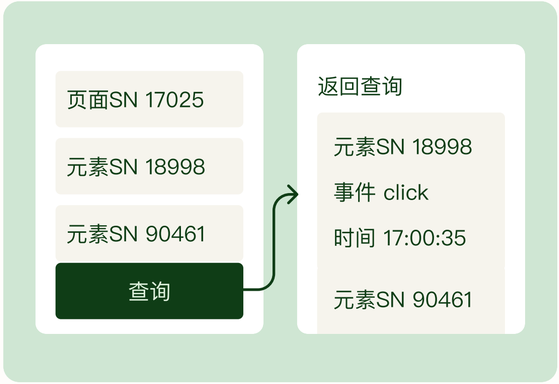
去掉强制查询的流程, 兼容多URL的数据抓取

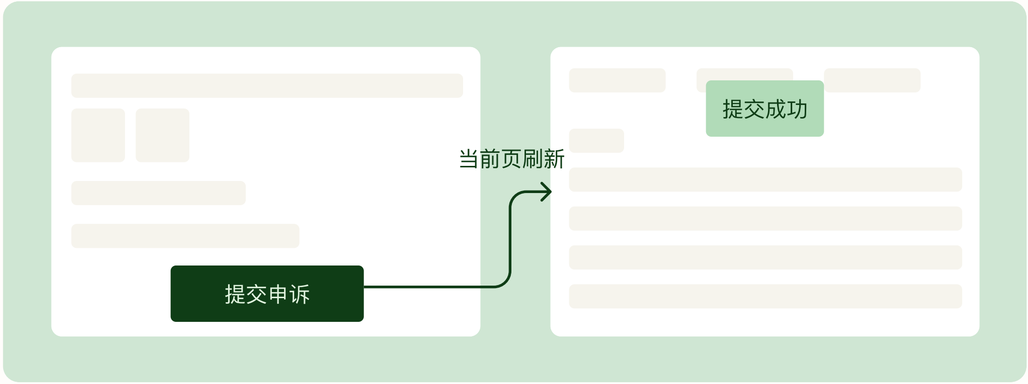
关键优化二 强制查询目标埋点，其实是以我为中心的设计, 改为默认展示实时上报, 查询降级为次级路径

关键优化三 兼容当前TAB下的多URL数据抓取, 以兼容当前页刷新导致上一URL埋点数据丢失

我以为的核心流程

- ✅ 低噪, 无其他埋点信息干扰
- ❌ 查看数据前的路径长, 操作成本高





事实上的核心流程

- ✅ 操作路径短, 高效


```
704
705         const globalWindowStart = clearTs > 0
706             ? Math.max(clearTs, Date.now() - 10 * 60 * 1000)
707             : (Date.now() - 10 * 60 * 1000);
708         updateTimeTip(clearTs);
709         if (currentQueryMode === 'all' && targetPsn && targetPsn !== 'DETECTING') {
710             lastCurrentPsnText = targetPsn;
711         }
712
713         const filterByWindow = () => {
714             const allowedOps = ['click', 'impr', 'pv'];
715             return logs.filter(l => {
716                 if (Number(l && l.tabId) !== tabId) return false;
717                 if (l.timestamp < globalWindowStart || l.timestamp > windowEnd) return false;
718                 const op = String(l.op || '').toLowerCase();
```

BETA PHASE

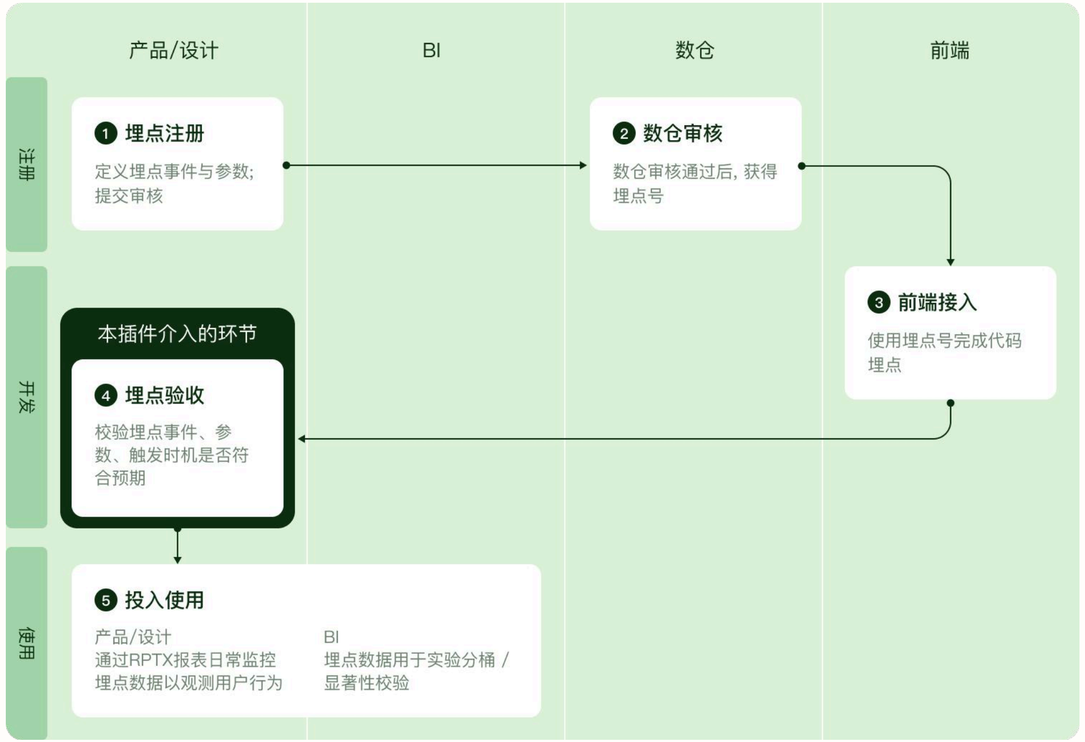
从个人工具到可靠的企业产品

进入方案评审和内测阶段, 发现事情远没有我想的这么简单!

BETA PHASE

它只是链路的小小一环, 无法承担超出边界的责任

向BI、数仓了解埋点的全生命周期 ⟶ 对用户做预期管理: 产品内补充透传免责声明, 增加新手弹窗



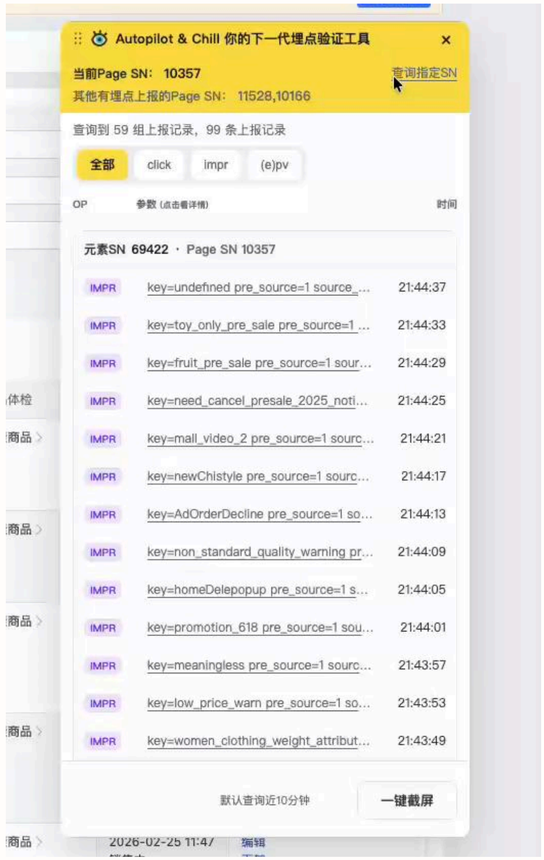
浮窗顶部常驻说明

安装后首次打开触发新手弹窗

BETA PHASE

企业工具不能只服务个人偏好，需要服从平台调性

Before



After

ROUND 1 方法: Codex通过Sketch MCP读取设计稿并还原
效果: 还原度55%, 整体接近规范, 但存在大量细节瑕疵

Sketch Codex

ROUND 2 方法: 将涉及到的主要规范转译为md文件, 增加Asset文件夹放Icon切图, 在Agent.md中增加相应指向性描述
效果: 还原度90%, 仍存在明显不到位细节

Design.md Asset Agent.md

ROUND 3 方法: 人工沟通细节并修改
效果: 100%符合规范

人工Prompt

这个工作流并非best practice, 但在缺少AI-friendly设计规范基建, 且只有3个页面的情况下, 已经足够用。
对于有成熟设计系统且要彻底转向AI的设计团队, 应先建设AI可调的组件库, 形成完善的Design.md和Skill

BETA PHASE

当直面更多用户，问题一股脑浮现

用户提出了新的跨页面验收场景：在页面A验收注册在页面B的埋点

👩 我要测活动首页的弹窗，但弹窗埋点是埋在全局首页的，怎么能在活动首页看到全局首页的数据上报呢？

👨🏽‍💻 侧边导航栏也是埋在全局首页的，我不想每次都要点到首页再测...

根据在当前URL是否有pv上报区分当前/其他页面SN

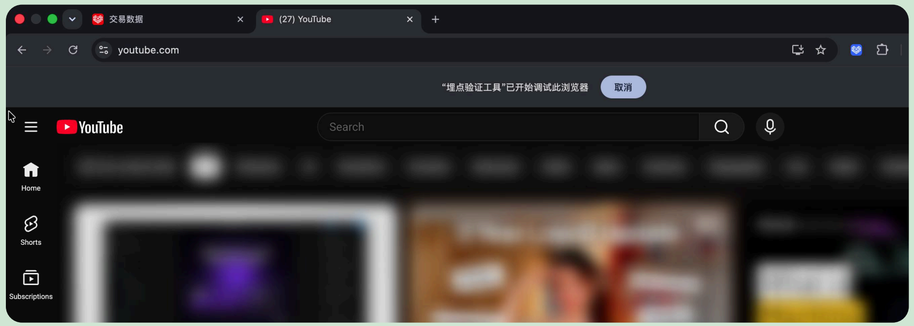
列表放开对页面SN的过滤

筛选兼容多页面SN

BETA PHASE

当直面更多用户，问题一股脑浮现

用户反馈使用插件时Chrome顶部出现如下全局提示，严重影响体验



这个顶部的条儿能叉掉吗？叉掉影响插件功能吗？

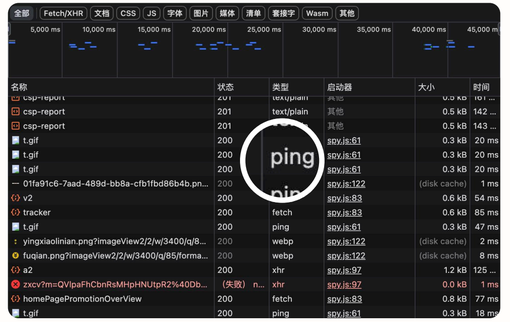
怎么这个提示条在chrome开的所有tab都会展示啊？能不能去掉？太大一条了

问题本质：为保证数据抓取完整带来的体验妥协

Codex为了解决漏抓数据的问题，选择了最保险也最重的方案chrome.debugger全量监听。这个全局提示是Chrome对debugger权限的安全披露，影响面大，必须让用户知晓

但实则是AI早期对埋点类型的错误假设，推导了不必要的技术方案

早期Gemini和Codex把问题当成普通网络监听来处理，先走webRequest，抓不到关键数据后又转向debugger。
后来我把Network截图给Codex，发现这类埋点实际是ping类型，上报方式更接近navigator.sendBeacon。于是方案从debugger改成页面主世界hook原生API拿数据。这个调整同时支持拿到埋点数据，且没有全局提示



我的收获

执行前先引导AI暴露约束(做了哪些假设、需要我确认的信息)，达成共识后再执行

BETA PHASE

我能相信这个数据吗 —— 如何保障数据工具的核心价值

人机协作QA: AI做自动化稳定回归，人在真实页面做环境和边界验证

第一步: Codex输出QA用例, 我审查和优化；
第二步: Codex根据用例写自动化脚本，模拟核心路径。每次迭代后重跑，确认关键流程没有被破坏；
第三步: 我人工进行真实环境验证

问题：开发和QA二位一体, 又当运动员又当裁判, 测试质量差

开发和QA在同一线程，导致QA subagent继承开发逻辑写用例。
典型案例：真实用户连续点击同一个按钮两次，两次payload完全一样, Codex默认去重，QA subagent按此逻辑输出了用例

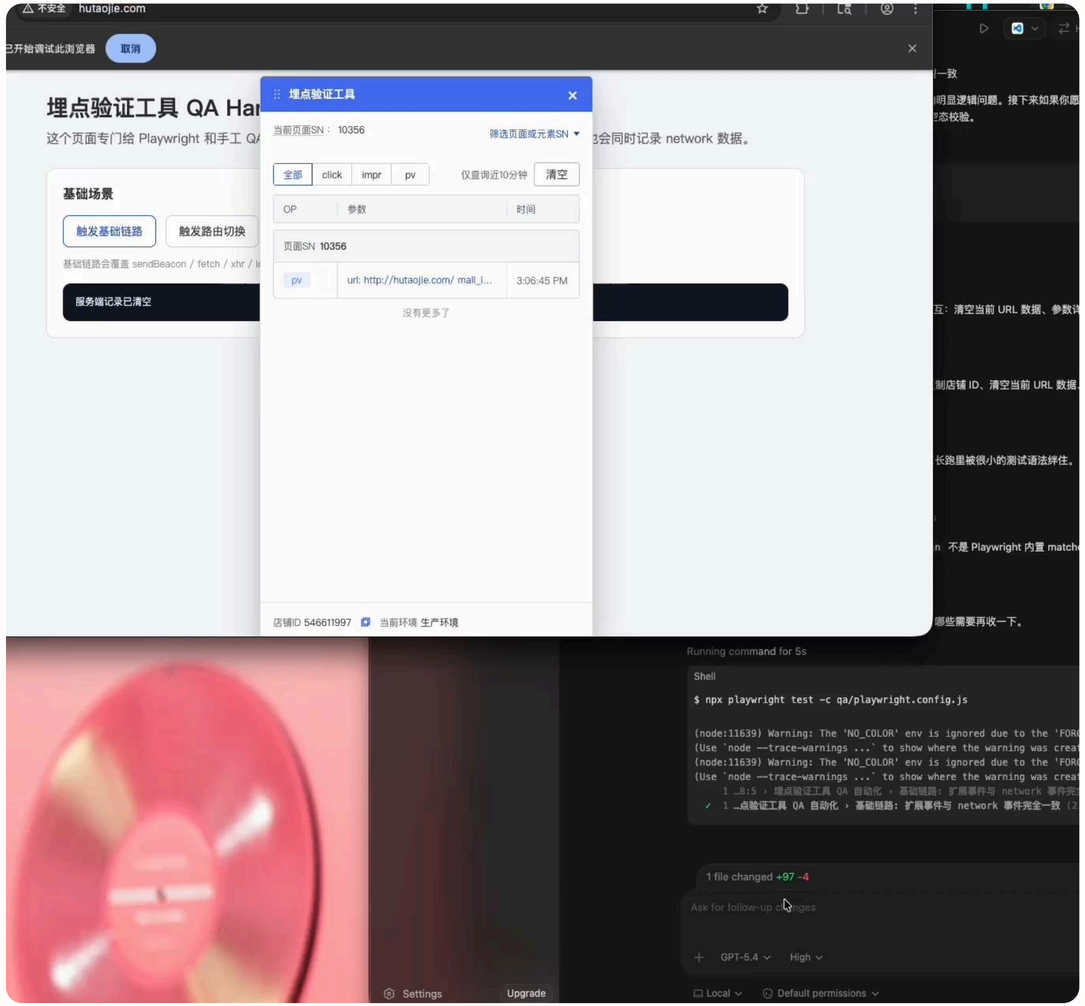
改用双Agent架构解决问题

在Agent.md中配置独立的QA agent, 参考资料仅提供交互说明和环境文档, 独立线程, 避免上下文污染;
要求其站在用户风险、误导场景、失败路径的角度写用例

Agent.md　Design_spec.md　Env_test_context.md　独立线程　对抗性视角

我的收获

开发阶段用主子Agent提效, 适合快速反馈，不要求完全独立; 关键QA阶段：用双Agent做独立审查。
严格管理文档权限和内容以避免上下文污染: Agent之间只共享交互说明&环境文档, 不夹杂实现细节



BETA PHASE

信任的建立并非只靠QA, 还有持续的内测和迭代

产品/设计参与内测

30+人

覆盖商家端核心设计和产品团队

次均埋点验收时长

-57.2%

统计自用户调研问卷数据

因埋点验证出错而重跑实验

0次

26年1季度团队内有2个需求因此问题重跑实验

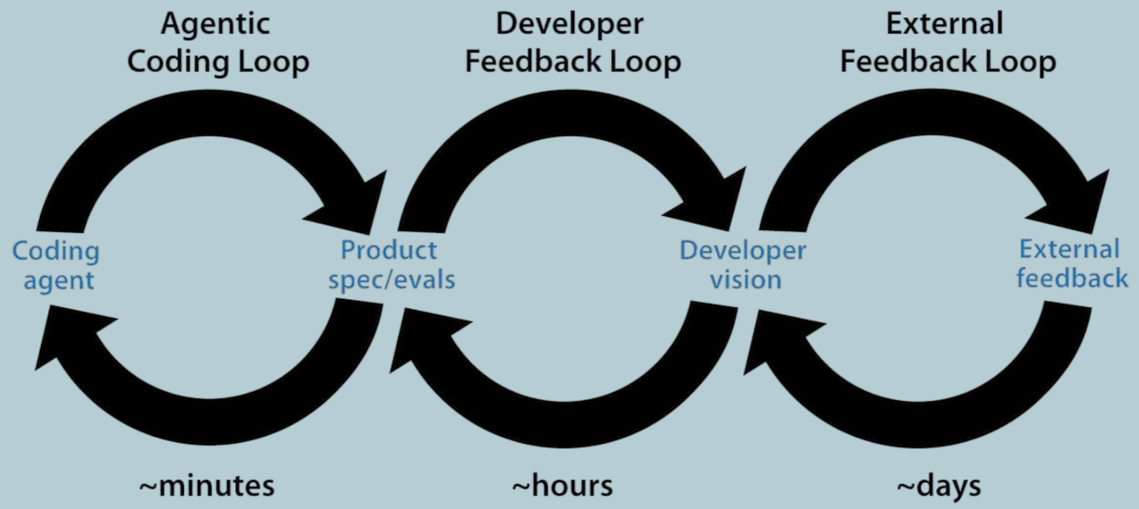
总结

AI加速开发, 但企业产品仍要一步步建立专业性和可信度

🤖 AI让我快速做出过去做不到的东西, 但并不代表产品成立

🧠 我的关键价值，是把对用户场景、业务边界和可信度的判断持续补进上下文

💬 设计判断、边界定义、外部反馈和真实内测才推动它从个人工具变成可被广泛使用的企业产品



三个关键的产品开发循环
引自: X @AndrewYNg(吴恩达)