SpaceX 工程师撰写的 Grok Bot 实战指南

72 小时现场直播构建，3 天高强度实战全景复盘。

TABLE OF CONTENTS

指南目录与核心全景

01 · THE EXPERIMENT

三天，一家公司，全程直播

来自 Grok Bot 团队的三位工程师 —— **Matt Palmer**（开发者体验）、**Roshan**（产品负责人）与 **Lauren**（“Potato”，工程架构负责人，PStack 作者） —— 坐在旧金山的一个演播室里，面对空无一物的 GitHub 组织、空的 Slack、空的 Notion，且没有任何既定商业想法。72 小时后，他们交付了一款拥有数千名玩家的线上产品。本指南中的所有内容，全部提炼自这三场无剪辑的高强度实战直播。

第 1 天 —— 头脑风暴与构想
从餐厅快闪策划案，变成快闪服务平台，再变成艺术展览，最后变成周边快闪店。一天之内经历了 4 次业务大转型。软件编写从来不是瓶颈，就“究竟做什么”达成共识才是。

第 2 天 —— 搭建与原型迭代
全面转型为游戏工作室。用纯 HTML/CSS/JS 手搓名为“Cupcake”的自动对战简易原型并在直播中试玩 —— 现场暴露了数值属性 Bug，并直接上演了前端调试作弊演示。

第 3 天 —— 正式发布与上线
一夜之间，Bot 工厂自动化合入了 170 个 PR。上午 10 点，游戏以 **Thursday Arena** 为名正式上线。截至直播结束：累计合入 433 个 PR，约 30,000 次页面浏览，约 6,000 场对局，营收 $0。

433
三天内合并的代码拉取请求 (PR) 总数

~2,000
上线首日登录账户的真实玩家数

~30k
已发布游戏获得的线上总页面浏览量

$0
商业变现营收 —— 最坦诚的真实数字

🎯 **为什么一份源于直播的实战指南如此值得阅读?**
官方产品文档只会罗列功能是什么。而连续三天未剪辑的高压直播，展示了人们在极限时间压力下**究竟拿它做了什么** —— 哪些模式经受住了火线考验，哪些模式产生了巨额账单，哪些模式在悄无声息中崩溃。本指南沉淀的几乎每一条建议，都是某人被镜头记录下的真实动作，而非某个功能纸面上的美好畅想。

*关于名称的说明：自动生成的字幕将产品名称听写为“GrokBot”、“Rockbot”甚至“Brockbot”，将公司听写为“SpaceX AI”。它们都指向同一实体。文中引用的所有数据均为直播时当场披露的真实阶段性数据。

02 · MENTAL MODEL

队友，而非一次性任务

产品核心缔造者之一 Roman 在第一天直播开场时，阐述了整个平台赖以立足的核心哲学：**“我们真的希望 Grok Bot 给你的感觉，就像是和你并肩作战的真实同事与团队伙伴。”** 产品界面被刻意设计成类似 iMessage 的聊天形态，正是这种形态驱动了本指南中的所有最佳实践。

构建产品所押注的三大基石假设

1. **队友范式，而非任务范式。** Bot 是一个有名字的同事，你会连续数月反复找它协作，而不是一个用完即弃的对话窗口。它的认知与上下文会随时间持续复利累积。
2. **每个 Bot 拥有专属的计算机。** 这是一个配备独立浏览器和终端的 Linux 虚拟机。它能操作遗留内部软件、政府政务门户以及任何没有公开 API 的网站 —— 凡是人类在屏幕前用鼠标键盘能干的，它都能干。
3. **全天候云端运行。** 合上笔记本电脑，工作依然在进行。“当你休假时它可以工作；当你安然入睡时它依旧在工作。”

实操中的思维转变

- **你是在招聘，而不是在配置。** 定义一个 Bot 的职责范围，更接近于撰写一份岗位职责说明书（Job Description），而非勾选后台设置项。
- **纠错会产生复利。** 告诉 Bot 它为什么做错了是一笔高回报投资，而不是一次被动的打断。
- **回过头来收获现成成果。** 正如 Amrita 所言：“最爽的时刻莫过于回来看到工作已经被干完了。”
- **随时打断完全正常。** 在任务中途纠正并重新引导 Bot 的方向，既常见也值得鼓励。

底座模型 vs 调度外壳

Roshan 明确划定了两者的界限，这对决定把任务派发给谁至关重要：

- **Grok** 是底座模型（Model）。
- **Grok Bot** 是调度外壳（Harness） —— 刻意保持轻量，擅长多 Agent 编排、工具调用与主观判断。
- **Cursor 云端 Agent** 则是重型编码执行外壳 —— 当遇到真正的硬核研发工程时，Bot 会主动将代码编写下放给它。

在 Grok Bot 中做原型验证，通过云端 Agent 实施生产交付。

> **“把一件事情做到 90%，和完全撒手让它端到端全自动彻底搞定 —— 两者带来的心理感受是天差地别的，后者充满魔力。”**
>
> Roman · Grok Bot 101, Day 1

03 · ARCHITECTURE

Bot 真正拥有的资源底座

清晰了解 Bot 之间**哪些东西是共享的、哪些是严格隔离的**，能解释直播中人们遇到的大多数意料之外的行为 —— 无论好的还是坏的。

🖥️ 专属的独立虚拟机

- 每个 Bot 独享一个 Linux 虚拟机，预装浏览器、终端和可扩展应用。
- 一个 Bot 无法窥探另一个 Bot 的屏幕 —— 这种强隔离性使得两个 Bot 能同时独立编辑同一个 PPT 的不同幻灯片。
- 在涉及账号登录、双重验证 (2FA)、人机验证码 (CAPTCHA) 等敏感操作时，人类可随时接管屏幕亲自操作。
- 由于虚拟机纯基于 Linux，**不支持 Linux 且无 MCP 的专有工具完全无法调用**（Q&A 环节明确确认）。

🧠 专属的独立记忆库

- Bot 拥有只属于自己的独立持久记忆。它能记住你的个人偏好、工作风格和专属约定。
- **关键边界：记忆绝不跨 Bot 共享。** 告诉了 Bot A 的信息，Bot B 浑然不知。
- 若要让某个知识对全体 Bot 可见，必须显式写入**共享团队驱动器**（下述）。
- 记忆能够被人类直接打开审查、单条编辑或定向清除 —— 这是一处极其关键的安全护栏。

📁 共享的团队文件驱动器

- 这是全体 Bot 与人类共同的**唯一事实真相来源 (Single Source of Truth)**。
- 存放公共规范：品牌风格指南、团队花名册、各阶段完成定义 (Definition of Done)、共享规则库。
- **踩坑红线：如果某份文件必须被所有 Bot 严格遵循，绝不能仅放在某一次对话附件中，必须扔进共享驱动器。**

🔌 共享的第三方集成服务

- 认证一次，全员复用：在团队级完成 Google Workspace、Slack、GitHub、Linear 授权。
- 所有配置了对应权限的 Bot，均可直接读写上述第三方服务。
- 若某个软件没有现成官方集成？**让 Bot 通过浏览器直接操作 Web 版。**

⚡ 云端与本地混合执行模式

工程师们并非完全将代码托管在云端 Bot 虚拟机中跑。通过底层同步工具（如 PStack），他们在**本地终端、本地 Neovim 与云端 Bot 之间保持毫秒级双向热同步**。Bot 在云端运行测试、提交 PR，人类在本地随时查看 Diff 并微调代码 —— 两者无缝融合。

04 · BUILDING A ROSTER

一 Bot 一职

这是整整三天里，每一位登台演讲的工程师被问及经验时复述次数最多的一条金科玉律：不要试图做一个通晓一切的"全能万应助手"，而是打造一支**每个成员各司其职、直呼其名的专属专家团队名录（Roster）**。

为什么单功能专精 BOT 表现远超全能助手？

1. **上下文边界保持精炼。** 每个 Bot 的上下文预算是有限的。一个试图同时包揽四项无关任务的全才助手，会极快耗尽上下文并开始健忘与胡言乱语。
2. **人类大脑能清楚记得该向谁求助。** 研发工程师 Ling Shi 形象地打了个比方："人类的大脑也记不住长篇小说，但你必须清楚知道需要参考谁的意见。"
3. **扮演特定角色时模型表现显著提升。** 当一个 Bot 的任务边界被约束得足够狭窄时，人类纠偏的反馈闭环就会极度紧凑，促使模型输出高质量成果。
4. **天然获得零成本的并行处理能力。** 同时派出 5 位专家 Bot 各自展开工作，之后统一汇集综合 —— 这与人类团队的"开会派工 → 独立攻关 → 汇总对齐"模式完全一致。

一个优秀的角色定义必须讲清的内容

- 一个清晰单一、能用一句话叫出名字的权责归属。
- 明确限定允许调用的工具与数据源。
- 解决问题的工作方式与思考原则，而不仅仅是工作产出物。
- 哪些高危动作在执行前必须先获得人类审批确认。
- 是否配置周期性的定时例行任务（Routine）。
- 杜绝泛化的"通用助手"。走向高度专精：**猎头侦察员 (Talent Scout)**、**报销管理员 (Expense Manager)**、**Bug 复现专家 (Bug Reproduction)**。

如何组织侧边栏管理团队

- **分组如同画组织架构图**：按职能归集为管理层 (Leadership)、工程组 (Engineering)、作战室 (War Room)。
- **标签用于提示专业领域**：一旦你开始用食物或外号来给 Bot 命名，标签能帮你一眼看懂专长。
- Simon 的**职能色标管理法**：调研类用橙色，大客户跟进用绿色，拓客销售用另一色 —— 扫一眼会话列表便知客户处于漏斗的哪一阶段。
- 将日常高频使用的三到四个核心 Bot **固定置顶 (Pin)**。

⚠️ **警惕 Bot 无序蔓延陷阱 (Bot Sprawl)**

在直播中拥有最庞大团队的 Simon 直言不讳："我向你保证，我曾经建立过太多 Bot。最后局面变得混乱不堪。你必须时刻反问自己：真的有必要为这事专造一个新 Bot 吗？"团队共识：**克制臃肿，你根本不需要 45 个 Bot**。随意创建 Bot 的快感是廉价且有趣的，正因如此才必须克制。优先考虑为既有专家补充新技能或新例行任务，而不是盲目招聘新 Bot。

*命名绝非小事：团队在直播间向观众征集了大量生活化的名字 —— Tater、Whisk、Bake、Crumble、Dr. Eggbot。这显著改变了团队与 Bot 沟通的心理感受：给一个叫 Bake 的 Bot 派活像在托付同事；给"任务3"下指令则像在微观管理。

05 · THE META-BOT

制造 Bot 的元 Bot

连续三天里被唤醒调用次数最多的 Bot，既不是工程师也不是市场研究员，而是 **Dr. Eggbot（蛋头博士）** —— 一个全职负责创建、审查、优化其他所有 Bot 的“元 Bot (Meta-bot)”。直播中涌现出的几乎每一个全新团队成员，都诞生于对它下达的一句话指令。

元 Bot 的核心日常职责

- 根据人类的一段大白话需求描述，全自动生成新 Bot。
- 自动撰写新 Bot 的**角色描述 (Description)** —— 也就是充当底层系统提示词的灵魂字段。
- **主动审计团队名录**：“请审查我们当前的所有 Bot —— 瓶颈在什么环节？职责是否存在冲突？”
- 起名字：品味极佳，甚至有两位工程师在不同房间分别提需求时，Eggbot 为它们起的名字英雄所见略同，双双命名为“Ping”。

角色描述即 Bot 的灵魂

“角色描述 (Description)”绝不仅仅是一张外观贴纸。Lauren 强调：**“有人称其为灵魂，或者系统提示词。”** 它从根本上决定了 Bot 的语气声调、判断标准以及向外派工的行为风格。

- **写通用原则，不要写偶发事故**（详见下方反思）。
- 明确限定它全权负责的边界，以及必须上交确认的红线。
- 指名道姓地赋予它唯一认可的“事实标准源”。

真实踩坑：角色过度拟合与现场纠偏

DR. EGGBOT 最初为 CUPCAKE ENG 自动生成的角色描述：

> *“负责我们游戏工作室的高层上下文。唯一工作：通过 pstack potato mode 和云端 agent 统筹协调各项工作，进而进行监督与验证，以交付工程成果；对照你的剧本并严格执行。”*

Lauren 当场的评判：太过度拟合了。 这段描述直接把当天的特定工具链、特定故障现场和特定分支写死了，变成了一个不可迁移的永久代码补丁。

LAUREN 当场下达的纠偏指令：

> **“重新阅读 potato mode，提炼出 Cupcake 工程师应当遵循的底层通用原则，而不是硬编码这些过度具体的偶发事故细节。”**

这是为你编写的每一项技能、每一条规则都必须坚守的核心铁律，绝不仅限于 Bot 描述。

只要你开口，Bot 会自己为团队招兵买马

在第 2 天，Amrita 向手下的两位现有 Bot 发问：“根据你们目前手头正在推进的工作，还需要哪些 Bot 才能帮你们把工作做得更好？直接帮我把它们创建出来吧。”它们随后自动创建了三个专职 Bot —— **Battle Card Blair（竞品攻防）**、**Demo Drake（演示专家）** 与 **AI Radar（雷达侦测）**，每一个都拥有明确的岗位职责和指定的事实来源。主动询问现有团队缺失什么角色，是一种极其高效的高阶手法。

*若有现成模板，优先从官方市场导入。官方 Bot 模板沉淀了精良的记忆结构、预置例行任务与集成组件，且在分享导出时会自动剥离私密凭据和敏感聊天历史。团队的建议是：只要官方市场已有类似角色，绝不要从白纸从头搭建。

06 · AUTOMATION

自动化构件：技能与例行任务

自动化系统由两大核心基石构成：**技能（Skill）**沉淀"某件事情该如何完成"的确定性操作手法；**例行任务（Routine）**决定"这项工作在何时被触发执行"。直播中所有持久可靠的工作流均严格遵循此序构建，且在人工手动完整走通至少一次之前，绝不提前固化为自动化。

1. 人类手动走通 → **2. 获得稳定可重复的结果** → **3. 沉淀固化为技能 (Skill)** → **4. 挂载定时/事件任务 (Routine)**

打造一项技能的三种途径

- **动作示范 (Demonstration)**：点击"示教任务"，亲自在屏幕上操作演练一遍，Bot 即可把你的操作录像转化为一项可复用的技能。直播中曾现场示教制作幻灯片动效与竞品博客监控。
- **日常纠偏沉淀 (Correction)**：最具长期复利价值的途径 —— 详见下方黄金法则。
- **人工手写代码规则 (Writing)**：当工作流包含分支判断逻辑、容错兜底或人工审批门禁时，单凭一次示范不足以让 Bot 穷尽各种边界分支，此时需要人工编写固化。
- **技能全局共享**：任何一个 Bot 学会的技能，团队内所有其他 Bot 立即原生可用。

一个例行任务必须明确定义的要素

- 由哪一个具体的专家 Bot 归口负责。
- 触发时间周期与时区 —— 或者是某个 Webhook/事件触发源。
- 数据输入的权威获取源头。
- 期望交付的标准输出物结构格式。
- 人工审批的边界范围。
- 当遭遇突发异常或外部失败时的回滚与告警预案。
- **空操作时的静默机制**：明确告诫 Bot，当检查完毕发现没有任何新情况需要汇报时，必须保持彻底安静（Stay quiet on a no-op）。

🎯 **沉淀纠偏原则，绝不要记录个别故事**

Roshan 的法则是："一旦看到 Agent 思考方向发生了偏差，这就是为其量身打造一项技能来根治问题的绝佳契机" —— 而不是在对话框里临时多打几行字绕过去。Ling Shi 的比喻更狠："宁可深入一层修水管，也不要只擦水槽里的积水。" 但这很容易走向另一个陷阱：**Agent 往往习惯性地把今天踩坑的所有偶发细节全写进规则，导致该技能极度过度拟合**，下次毫无通用价值。必须做到：**提炼底层通用原则，坚决删除偶发故事**。

*例行任务的执行频次往往是巨额账单悄悄滋生的地方（详见第 12 章）。能用事件驱动触发的，绝不用定时轮询；能在每天早晚跑一两次的，绝不要设置成每 15 分钟轮询一次。

07 · THE SOFTWARE FACTORY

433 个 PR 是如何自动合并的

“软件工厂 (Software Factory)”正是创造这一惊人数字的核心工程模式。开发团队其实并不喜欢这个词（“我其实真不太喜欢工厂这个叫法”），但这个名字最终深入人心。剥去所有的品牌宣传外衣，它本质上是一个严密的闭环：**编写者 Agent 提交细粒度的小改动，验证者 Agent 实际运行应用并尝试破坏它，唯有验证通过的代码才被允许合入。**

流水线闭环运作五步法

1. **宏大方案拆解为细分阶段。** 由 Lauren 编写的 Pstack 插件及其“Potato Mode”技能全自动完成：只需下达指令 `/potato mode` 加上 `full autopilot this plan`。
2. **编写者 Agent 提交小颗粒度 PR。** 调动 Cursor 云端 Agent，每一个细小关注点各分配一个 Agent，在各自的独立云端机器上并行编码。
3. **验证者 Agent 真机运行并进行模糊破坏测试。** “Swarm (蜂群)”技能会唤起一整支 Agent 舰队，直接在运行中的产品界面上四处点击挖掘 Bug。
4. **自动化验证全绿即自动合并 (Auto-merge on green)。** 验证通过是唯一的合入门禁 —— 而不是让人类肉眼去通读庞大的 Diff。
5. **试玩 Bot 端到端完整打通游戏。** 专门的试玩 Bot “Chrome”监控 CI 通过的 PR，并在真正合入前在真机浏览器中完整试玩一局。

自主权阶梯 (Autopilot Ladder)

- **仅限调查 (Investigate only)** —— “先不要提 PR，查清楚你认为的原因再来向我汇报。”
- **草稿模式 (Draft)** —— 提交 PR，等待人类审查确认。
- **自动驾驶 (Autopilot)** —— 自主实现并验证，测试全绿自动合并。
- **完全自动驾驶 (Full autopilot)** —— 规划、拆相、编码、验证、合并端到端全自动。
- 一旦正式上线生产环境，Lauren 便审慎下调了一个档位：“也许不要开完全自动驾驶 —— 我们真的敢吗？”

架构师环节 (The Architect step)

Pstack 的“架构师 (Architect)”技能会将同一个技术难题**同时并行抛给 4 个不同的顶级模型**，随后由评审裁决环节合并或选出最优架构方案。

对于重大技术选型，这非常值得，因为不同模型会发现不同的边界约束。但在初期手搓极简原型阶段，团队果断跳过了它：“在这个节点我其实根本不在乎架构规范，好玩才是唯一的指标。”

⚠ 必须坦白的事实与警告

Lauren 在谈到这款游戏时坦言：“说实话，我压根没有逐行看代码。我就是纯靠 Potato 模式和 Pstack 自动流转。”但负责 Grok Bot 正式商业产品的核心架构师 Eric 同样立场鲜明地指出，**他在真实的生产核心代码库中绝不会这样撒手不管** —— 他会细致审阅每一个 PR 并严格执行反模式检查。软件工厂的激进程度必须与**爆炸半径 (Blast Radius)** 严格匹配，而一个 72 小时限时试玩的小游戏，其爆炸半径几乎低到了极限。

08 · VERIFICATION

确定性验证胜于一切

如果这整整三天的实战记录中你只打算吸收一条构建指令，请务必牢记这一条。每一位真正用 Bot 成功交付了严肃产品的工程师都以不同形式强调了它；而那些尚未构建验证闭环的团队，无一例外陷入了低效拉锯的巨大泥潭。

> **“验证机制是重中之重。你必须让 Bot 真正把代码运行起来、去截屏留存证据。唯有亲眼看到运行证据，你才能拥有坚实的信心确定它真的理解了问题。”**
>
> Lauren · Day 3

一套合格的验证技能 (VERIFICATION SKILL) 究竟包含什么？

🖥️ Agent 可操纵的确定性 CLI

绝不要让 Agent 每次临时手写即兴测试脚本 —— 那既浪费大量 Token 且无法复现。Lauren：“你必须提供一个 CLI、标准脚本或测试套件，让 Bot 能以稳定、确定性的方式与应用交互。我宁可直接塞给它一套标准的成熟工具。”

🗺️ 机器可读的功能地图 (Feature Map)

一份结构化的功能与页面路径映射表，使 Agent 无需在应用里盲猜即可直达要测试的界面。Thursday Arena 甚至更进一步，上线了一个 `/rules` 页面并提供类似 `llms.txt` 的接口，供 Agent 随时读取游戏的底层规则。

团队在实战中铁腕执行的三大铁律

1. **先精准复现，再动手修复。** “唯有当 Agent 能在真机上精准复现 Bug 时，我们才能信任它真正理解了问题所在。” 团队的常驻提示词：“在写任何代码前，先运行应用，找到确切的 Bug 和确切的表现，然后再开始。”

2. **提交 PR 必须附带证据。** 剧本中的明文规定：UI 界面更改必须附带截图或视频录屏；后端更改必须附带性能对比数据。云端 Agent 能够自动录制自身的测试运行视频并直接插入 PR。

3. **指名道姓命名技能，并用其名字调用。** 团队的验证技能名为 `/verify cupcake`，在游戏上线生产后，所有的自动驾驶指令都强制将其列为不可协商的必跑条件。

> 🎯 **普适判别法则：什么样的任务才适合全自动化？**
>
> Eric 给出的决定是否自动化的黄金准则：“审视你的工作流并反问：**这里是否存在一个能够闭环验证的确定性反馈回路？如果有，那就是你可以放手让 Bot 尽情施展的地方。**” 代码编写之所以是最佳试点，正是因为单元测试、编译构建和真机运行能提供确凿的机器可读信号。任何缺乏此类信号的流程，在赋予自主权之前，必须先发明出这种确定性信号。

*缺乏验证技能的典型翻车表现大家再熟悉不过：Bot 干完活后两手一摊对你说：“好了，你现在可以把应用跑起来，在界面上到处点一点，然后告诉我行不行。” 这种来回人工试错才是吞噬所有时间与精力的黑洞。

09 · PROMPT PATTERNS

经受住实战检验的 7 大 Prompt 招式

这里没有华而不实的提示词奇淫巧技，而是在整整三天里，六七位不同的工程师在各自独立的操作中，不约而同反复高频使用的核心肌肉记忆。

1 · 用自己的话重述 (Restate it back) — 全体工程师必备

在一长串复杂或多阶段的指令末尾加上：“在开始前，先用你自己的话把任务重述一遍”。在开始烧钱之前把误解掐灭在萌芽状态。第 2 天被誉为“全场我最喜欢的模式”。

2 · 随意倾诉，再行结构化 (Yap, then structure) — 语音转录实录

按住麦克风，以意识流的方式畅所欲言一两分钟，然后要求 Bot 将其整理提炼。想法在口述的当下就被完整捕捉，无需事后痛苦敲键盘 —— 团队在对话中实时用这种方式记录了大量业务灵感。

3 · 先提炼事实，再展开推理 (Distill, then reason) — 应对海量输入

面对冗长的口述转录或大型文档，要求 Bot 首先提取核心事实清单，然后**仅基于这份提炼后的清单展开逻辑推理**。这是 Blake 解决长篇输入下幻觉胡言乱语的克星。

4 · 交付物先行 (Outcome first) — Amrita 的实操习惯

开门见山说明：“这就是我最终想要拿到的产出物”，然后让 Bot 自行逆向拆解步骤。直接声明交付物形态和验收标准，远比手把手指导执行步骤有效得多。

5 · 动手改之前先彻底调查 (Investigate before you touch) — Roshan 的生产原则

面对任何线上生产环境问题：“深入排查，弄清楚到底发生了什么。**先不要提 PR。** 查清楚你认为的原因后再回来向我汇报。” 严禁未探明根因擅自改动代码。

6 · 实时打断与微调纠偏 (Interrupt and nudge) — Ling Shi 的调度心得

密切注视 Bot 偏离轨道并在中途果断拉回，是正常的操作流程，绝非失败。每隔几分钟要一次状态更新，远胜于在死寂中空等 —— 这也是当场抓获某个决定 `sleep 300` 偷懒 Agent 的关键。

7 · 语音大倾倒入职法 (The voice dump) — Blake 用于新人 Bot 入职

录制一段 10~15 分钟的语音备忘录：我是谁、我的岗位是什么、目前的痛点是什么、哪些流程该自动化。把转录文本丢给你的第一个 Bot：“为我设计一套行之有效的自动化体系”，让它反向提议系统框架。

贯穿这七大习惯背后的唯一底层心智

上述每一个模式，本质上都是**在 Bot 真正消耗你的资金与 Token 之前，逼它证明自己已经完全理解了意图**。这与你在真实职场中对待一个刚入职第一周的有能力的新员工的心理防线别无二致 —— 这一对应绝非巧合，因为整套产品正是构建在这一隐喻之上。

10 · PROMPT LIBRARY

原汁原味实战 Prompt 库

从直播实战中近乎原封不动摘录的真实提示词。这些 Prompt 的结构本身就是最生动的教材：**目标产出物、明确约束、权威事实源，以及任务完成后的闭环动作**。

创建幕僚长 (CHIEF OF STAFF)

“你的工作是向 Email Ethan、Slide Sonia 和 Data Dan 获取他们手头工作的最新进展汇报。”

紧接着挂载例行任务：“每两小时，我需要你向你的团队主动索取工作进展，并排查当前是否存在任何阻塞项 (Blockers)。”

保持静默的知识库 BOT (STAYS QUIET)

“这个 Bot 的工作是静默旁观我与其他所有 Bot 之间的对话，但在被明确点名 @ 之前，绝不要采取任何行动。你应当静静等待消息主动发给你。我们需要极其克制、有选择地更新 Notion。我们绝不希望把所有垃圾信息一股脑全倒进文档库里。所以在同步之前，务必先向我请示确认。”

Roshan。克制性声明是整个提示词的灵魂所在。在其他会话中被总结为“像对待 git commit log 一样严肃对待沉淀”。

常驻 P0 巡检巡警，一次配齐免于日常怒吼 (STANDING P0 POLICY)

“建立一个每 5 分钟检查一次云端 Agent 的例行任务。检查它们是否偏离了轨道 —— 比如正在执行长达 300 秒的休眠（如 sleep 300），或者脱离了我们的目标，或者行事过于保守裹足不前。一旦发现它们偏离，立即予以打断并现场微调拉回。”

Ling Shi。核心洞察：反复在对话框里对 Agent 咆哮“这很紧急”只会诱使它跳过验证去瞎猜；确立制度化的巡检策略才起效。

审慎处理线上生产环境 BUG (PRODUCTION BUG)

“我怀疑我们的 ELO 匹配机制出现了故障。每次我在 ThursdayArena.com 上玩游戏，匹配到的全都是 AI 对手而非真实玩家的阵容，但排行榜上的 ELO 积分却在大幅变动。你能不能深入排查一下，彻底搞清楚到底怎么回事？使用 potato mode，并派出 Cursor 云端 Agent 去调查。先不要开 PR，查清楚你认为的根因后直接回来向我汇报。”

Roshan。症状描述、可验证证据、指定工具，并在改动任何代码前明确设立硬性停止点。

搭建用户反馈流水线 —— 包含安全防护条款 (FEEDBACK PIPELINE)

“上面的链接是我们接收用户反馈的 Slack 频道。我想搭建一套工厂流水线：查阅反馈、进行分类定级、尝试重现该问题，然后将确认的 Bug 录入我们的 Notion 数据库。非常重要的是，如果我们用 AI 来审查这些反馈，你必须明确警告你的 AI 严密防范提示词注入 (Prompt Injections) 攻击。.....用你自己的话把刚才这番要求复述一遍。”

Lauren。来自不可信外部用户的输入均视为恶意攻击面；在构建流程的一开始就把防御写入 Prompt。

10 · PROMPT LIBRARY, CONTINUED

研发、调研与商业化实操 Prompt

创建前端极速原型专家 (PROTOTYPING SPECIALIST)

“我们想针对 Cupcake 代码库中现有的客户端，做更多前端探索。你能不能孵化出一个专职 Bot，专门负责快速原型验证：探索如何在保持极度轻量化的前提下加入更多 3D 动画？这个 Bot 必须能够驱使 Cursor 云端 Agent 来交付成果。我非常希望利用 potato mode，在一个接一个的前端原型试验上启动 Agent 蜂群 (Swarm) 进行攻坚。”

Lauren 对 Dr. Eggbot 下达的指令。注意它明确指出了核心约束（“保持极度轻量化”）与执行载体，而不是直接给死具体输出物。

限定具体交付物的竞品调研 (COMPETITIVE RESEARCH)

“帮我把旧金山所有顶尖的餐厅主理人拉出来，挑出其中做得最好的 5 家网站；接着从它们的网站里找出 5 个做得最好的快闪专页；随后找出过去 6 到 12 个月内，在任何主流大都市举办过的 5 个最优秀的快闪餐厅案例。最后，把所有这些调研成果系统整理成一份文档交付给我。”

Cody Sanchez。层层链式递进，每一步收窄范围，并在末端指名道姓要求交付特定形态的文档。

产出鲜明战略站位的市场调研 (MARKET RESEARCH)

“仔细研读目标网站，深入理解该产品是什么、我们处于什么细分市场。现在去执行竞品分析：精准识别并深刻剖析我们的竞争对手，研究他们的营销官网，理解他们的市场定位 —— 最重要的是，指出我们具有哪些战略机会窗口，能与他们形成具有杀伤力的差异化竞争定位。”

Josh Kim。正是最后那句寻找差异化切入点的话，把一份平庸的信息罗列变成了可执行的商业武器。

通过指名技能向蜂群下发拓客任务 (DELEGATING TO A SWARM)

“帮我们寻找目标客户画像 (ICP)，确定这些广告位该卖给谁。你拥有一个名为 FindMyICP 的专有技能。……把我们敲定的目标用户画像输入给 Clay MCP，检索其企业与人员数据库，告诉我们世界上哪些公司符合这个画像 —— 具体交付一份包含 10 家目标企业的清单。”

指名道姓调用技能、指定外部连接器、限定收敛的产出物规模。

永久固化记忆偏好 (STEERING MEMORY)

“我希望你记住，今后在任何场合称呼我时，绝不要同时使用我的姓和名。只准直呼我的名字 (First name)。”

Amrita。只需说一遍，便被永久铭刻进该 Bot 的独立长期记忆中，往后所有会话永远生效。

🎯 **贯穿所有高质量 Prompt 的共同范式：明确钦定权威事实源**

在一名专职 Bot 的系统设定中直接明文写着：“一切主张均以 Sherlock 为权威事实依据 (Ground every claim in Sherlock)”。当细分领域的专家 Bot 被明确告知团队中哪一位同事的判断拥有最终权威时，AI 之间的胡思乱想与幻觉便有了终结与核实之所。

11 · ORCHESTRATION

管理团队，而非操作工具

一旦你的 Bot 数量突破大约 5 个，你亲自充当每个消息的二传手和路由器就会迅速演变成整个系统的致命瓶颈。为突破这一上限，直播中反复验证了 4 套规模化编排模式。

1 · 幕僚长模式 (The Chief of Staff)

- 你几乎只与这一个总管 Bot 对话；由它向下派发给具体专家，并汇总成果向你汇报。
- 它亲自负责新 Bot 的入职培训 —— 将上下文和团队剧本同步过去，人类无需喋喋不休。
- Simon 的铁律：团队里的其他所有 Bot 都要通过幕僚长来创建，这样它才能对每个专家的职责了如指掌。
- Blake 在一个幕僚长麾下直管 15~20 个细分专家，不设任何中层管理。
- 并非绝对准则 —— Shub 则明确表示自己更偏爱直接与垂类专家对话。

2 · 剧本全员广播 (Playbook Broadcast)

- 由一个专属 Bot 总揽维护一份动态演进的“团队工作标准剧本”（通常存放于 Notion）。
- 每次有了新的工程规范或业务规则，只需告诉它一次，由它向下广播给所有人。
- “你只需思考一次该怎么做……全体工程师 Bot 就会自发同步遵守，无需你逐个敲门通知。”
- 规范示例：所有 UI PR 必须附带截图，所有后端 PR 必须附带基准性能对比。

3 · 跨角色决策例会 (The Staff Meeting)

- 把几个不同角色的 Bot 拉进同一个会话群组，针对重大决策从不同立场展开辩论。
- Blake 甚至特意给手下 Bot 下达指令：**必须提出反对意见** —— “如果大家全是一团和气举手赞同，那开会毫无意义。”
- 由幕僚长负责综合各方激辩，并形成最终的决策建议书汇报给人类。
- **严厉警告**：群聊非常啰嗦、兴奋且极其烧 Token。仅用于重大决策，绝不可用于日常例行公事。

4 · 子 Bot 游击军团 (The Sub-bot Army)

- 由一个中层 Bot 将海量的大规模批处理任务，下发给一整群低上下文预算的基层“工兵”。
- Simon 麾下的“工兵 (Soldiers)”在一个共享群组中，以高度并行的方式同时调研 100~200 个目标企业账号。
- 弹性极强，按需随时唤起与销毁；所有产出流回父级 Bot 接受统一审计。
- 形态上与研发团队的“蜂群 (Swarm)”完全同构 —— 一个指挥官，带领大量一次性轻量执行者。

> 💡 **全场无人彻底解决的真实现实困境**
>
> 知名科技播客主 Matthew Berman 被问及同时调度如此多并行 Agent 是什么感觉时坦白：“**我产出的成果比以往任何时候都要多，但同时，我也比以往任何时候都更加忙碌。**” 在十几个飞速运转的 Agent 之间频繁来回切换上下文，是一项极其真实且目前在交互设计上尚未彻底解决的认知开销。幕僚长模式是目前能拿出的最佳对策，但也仅能起到缓解作用，而非终极解药。

12 · ECONOMICS

真实成本与经济学核算

产品计费完全基于实际资源消耗：你只需为同 Bot 对话消耗的 Token，以及 Bot 虚拟机自身运行的屏幕交互算力买单。以下是直播中披露的真实账单数据，以及现场抓获的 4 大资金暗中流失黑洞。

$20–30
制作一份完整的销售客户案例 PPT 幻灯片 —— 相当于人工 4~5 小时的纯体力劳动

$1–2
端到端全自动彻底解决一个中等复杂度的客户支持工单

$0.20
将低复杂度简单工单分类归集并进行批处理时，单张工单的极致成本

~$1,000
单个 Bot 在人类投入 60 秒注意力的前提下，为企业挖掘并节省出的年度公用事业费用

资金暗中蒸发的四大黑洞

1. **设置过频的定时例行任务 (Routines)。** 绝对是产生惊人账单的第一大元凶。“如果你有 3 个例行任务每 15 分钟轮询一次，一天就是数百次高频调用。” 必须严肃审计频次；优先拥抱 Webhook 事件驱动；对于绝大多数日常场景，每天运行一到两次足够应付。

2. **明有官方 API 却硬用浏览器模拟点击。** 通过 Computer Use 操作网页表单所消耗的视觉算力，远贵于直接调用 API 接口。Shub 的偷师绝技：让 Bot 第一次在浏览器里点一遍，命令它审查刚才触发的网络请求 (Network Requests)，随后命令它往后直接对这些 API 端点发网络请求。

3. **群聊放在后台未关闭。** 多 Agent 群聊极其兴奋，Bot 们会互相接话争吵，瞬间卷走大笔开销。日常办公请坚持单对单指定点名。

4. **Bot 队伍过度膨胀。** 每多造一个 Bot，就意味着更多重复建立的上下文、更多并行的定时任务、更多重叠的认知消耗。精简的团队才是廉价高效的团队。

你能直接掌控的降本杠杆

- **调低 Bot 的吐字啰嗦程度 (Verbosity)**：沟通风格直接决定 Token 消耗。
- 命令 Bot 及时忘掉不再相关的陈旧上下文。
- 直接向 Grok Bot 提问，让它为你的系统架构提供降本优化方案。
- 专设一个审计 Bot，唯一职责就是定期排查其他 Bot 的例行任务与技能效率。

你无法调节的不可控项

Computer Use（屏幕视觉识别与鼠标模拟）本身的底层虚拟机算力单价对用户是不可调的。

唯一降低这部分开销的途径就是**减少对它的依赖**：只要存在 MCP 就立刻接入，唯有没有任何 API 能触达的蛮荒之地，才允许退守至浏览器视觉模拟。

13 · APPROVALS AND BLAST RADIUS

带有刹车的自主权

Matthew 的定性最为精辟通透：**“信任你的 Bot，赋予它们充分的自主权；但同时，牢牢立好防护栏 —— 确保无论它们跑得多么欢快，在真正交付上线任何东西之前，必须回到你面前请示批准，这样局面就绝不会失控。”**

自动审查与硬性防线

- 系统内置的意图风险分类器会自动裁决动作危险度，并在必要时打断请示。
- 你可以在其上追加确定性白纸黑字红线：“严禁未经我人工批准擅自向外发邮件”，“创建 PPT 幻灯片无需请示我可直接执行”。
- 针对具体操作类型的开关 —— 例如部署到生产环境必须一律预先弹窗请示。
- 宁可过分谨慎：在直播中，一个负责开发表单的 Bot 在动工前，甚至主动停下来向人类确认该表单是否涉及收集敏感用户个人身份信息 (PII)。

机密凭据管理机制

- 密码与私钥必须通过专有加密安全表单存入安全金库 (Vault) —— Agent 本身绝不可窥视到明文密码。
- 活动中途上线了 1Password 集成；Bot 可以直接通过密码库完成二次安全认证。
- 对外分享导出的团队模板只包含记忆、配置、技能与例行任务 —— 凭据与聊天记录会被自动剔除。
- 对于极度敏感、无论如何都不愿假手于 AI 的顶级鉴权，请人类直接接管屏幕手动输入。

直播火线中付出惨痛代价学到的三大教训

1. **任何涉及竞争与信任的逻辑必须由服务端权威判定。** 由于游戏逻辑起初跑在前端客户端，Matt 在直播众目睽睽之下直接按 F12 打开开发者工具，在 Console 里现场篡改数值作弊。通过 Feature Flag 隐藏调试 UI 毫无安全可言 —— 有心之人瞬间就能翻出来。这段作弊代码在前端必须彻底物理拔除。

2. **用户提交的文本是巨大的恶意攻击面。** 上线后的反馈流水线紧急追加了全套防御：前端字数截断、服务端敏感词过滤、入库数据转义净化、模型层安全内容护栏、API 限流 —— 以及在分流 Bot 提示词中严厉告诫其防范 Prompt 注入。

3. **全自动修复会导致生产环境瞬间崩溃。** 它真的发生了：软件工厂自动化提交的一条劣质 SQL 查询，直接把刚上线的线上游戏打崩停摆，而当时团队甚至正在镜头前探讨“克制”。所幸几分钟内迅速抢修完毕 —— 但这是必须在**数据库迁移和线上生产部署前保留人类最后把关**的铁证。

培养一个新 Bot 的“爬-走-跑”三级渐进法

David 在客户支持场景中验证的递进法则适用于几乎所有领域：**第一步（爬）**：Bot 仅拥有只读权限，只负责查阅并提炼摘要；**第二步（走）**：允许其在内部系统撰写回复草稿，供人类审核；**第三步（跑）**：唯有信任完全确立后，才放开权限让其直接发送与执行 —— 即使到了这一步，对于修改面向客户的知识库这种具有超大爆炸半径的高危动作，依然保留人工最终确认。“从小处着手。努力让 Bot 融入你现有的工作节奏，而不是强行颠倒过来。”

14 · CASE STUDY

72 小时从零打造 Thursday Arena

一款多人自动对战游戏：将官方公开市场上可用的真实 Bot 抓取为参战角色，为其自动赋予稀有度等级、AI 生成的角色立绘、基础数值与专属技能。由三名人类与一支由 Bot 构成的团队联合构建，在第 3 天上午 10 点，通过一个头像完全空白的全新 X 账号发推正式公测发布。

它是如何被一步步构建出来的

- **极速原型先行** —— 纯原生 HTML/CSS/JS，内存状态管理，无数据库，无用户鉴权。从始至终唯一衡量标准：这个对局玩起来到底好不好玩?
- 好玩的玩法闭环验证通过后，用 React/Next.js 全面重写，严谨划分前后端架构。
- 后端采用 Vercel Serverless 上运行的 Go 语言，数据库选用 PlanetScale，登录接入 Clerk（Sign in with X），跨网络数据校验引入 Zod。
- 卡牌立绘由 Grok Imagine 实时生成；角色采用代码而非静态雪碧图渲染，以便程序化动态切换状态。
- 早期故意删光了所有测试用例 —— “Agent 通常很不擅长写测试代码” —— 决定把代码质量重构推迟到后序阶段。

起决定性作用的产品取舍决策

- **坚决杜绝数值付费买赢 (No pay-to-win)。** 曾被反复讨论，但被当场坚决否决。原生广告、赞助专属卡牌和实体周边才是首选商业化路径。
- **残酷地砍掉复杂功能 (Ruthless cutting)。** 石头剪刀布克制机制、冗长的复杂数值面板、多战场地图……统统被果断砍掉，仅保留最极致流畅的单核心对战循环。
- **为 Agent 定制专用的规则解释接口。** 团队上线了 `/rules` 页面，供 Agent 在需要澄清游戏规则时自行联网抓取阅读。
- **用确定性函数替代模型的主观随性发挥。** 战斗裁决逻辑全写在纯 Go 代码中，绝不交由 LLM 凭空想象胜负。

72 小时从零构建并发布上线所达成的核心指标

433
代码库累计合并的 PR 总数，全部由云端 Agent 和 Potato 模式流水线编写并推送

2,000+
公测发布数小时内涌入体验的真实玩家总数，全部来自一个白号推文传播

100+
游戏中可供解锁的参战 Bot 角色总数，全部由 Bot 自动抓取市场卡片并生成立绘

72h
从空无一物的 GitHub 组织，到产品正式面对全球公测并稳定支撑高并发的全部耗时

15 · THE ROSTER

工厂团队全员花名册

支撑 Thursday Arena 从零诞生的真实 Bot 团队阵容。值得将其作为组织设计的标准模板仔细研读：**请注意真正写代码的 Bot 占比有多么稀少，而绝大多数 Bot 的存在仅仅是为了检查、路由、分类或抓取。**

Dr. Eggbot（蛋头博士）· 元工匠 — 调用频率最高
负责创建并审查其他所有 Bot。撰写它们的基础描述，为它们起名字，随时解答“当前我们的系统瓶颈在哪”。

Steve · 幕僚长 (Chief of Staff) — 中央调度枢纽
团队总协调。人类几乎所有指令都经由他分流派发。在大家嘲笑团队里究竟有多少个“幕僚长”之后，被特意赋予了这个真人名字。

Cupcake Eng · 研发总指挥 — 工程交付核心
通过驱动 Potato 模式与云端 Agent 来掌控整体工程成果，全程监督、调度并最终验证交付。

Bake · 创始工程师 (Founding Engineer) — 代码合并哨兵
监控 PR 动态、执行代码合并，并为特定 Bug 现场唤起云端 Agent。曾被人类用语音电话直接呼叫并在直播中途在线合并了一项 PR。

Chrome / Play · 真机试玩员 — 端到端最后把关
紧盯通过 CI 检查的 PR，并在真正放行合并前，在真实浏览器中亲自把整款游戏完整跑通玩一遍。

Crumble & Hashbrown · Bug 分流与二次校验二人组 — 双重质检机制
Crumble 配合试玩员重现报障并录入确认项；Hashbrown 作为质检员的质检员，在自动化流水线获准介入前二次复核其分类是否准确无误。

Comment Sicko · 注释死神 — 代码库代码净化
无情清理多余无用的代码注释。核心逻辑非常深刻：Agent 往往习惯性用注释作为留下 Hack 和临时补丁的借口而不去根治问题，这些注释积重难返会导致代码库彻底腐烂。

Whisk (Crit) & Glow (Tone) · 策划评审与音画探索 — 游戏设计核心
Crit 在发布首日清晨做出的严厉评判 —— “这游戏太难了”，直接在上线前重构了数值平衡；Glow 则用 Strudel 和 Suno 生成背景音乐并丢进 Notion 供人类试听。

Ping、数据科学家、Vincent (Cerebro) · 运营监控三人组 — 日常运转中枢
Ping 监控 @ 消息并沉淀回 Notion；数据科学家在上线首日每 15 分钟出具一份报表；Vincent 则负责拓展增长并把找出的 ICP 导入营销流水线。

*团队的命名哲学全是以土豆或烘焙为灵感，合并 PR 在内部被戏称为一次“MASH（捣成泥）”。这些好玩的词汇从 Dr. Eggbot 泄露到了整个 Notion 正式项目管理流程中，反倒让团队谈论工作的频次大幅增加。

16 · FROM THE FIELD

一线实战名录：售后实施与市场营销

三天中穿插的专项业务工坊是全场最可直接抄作业的实操范本。以下是各位实战业务专家在各自真实工作流中所实际部署的落地配置。

售后交付部署 · BLAKE (AI DEPLOYMENT MANAGER)

单一联系窗口，单客户独立专员架构

“Gus 是我最好的铁哥们。” Blake 永远只和一个名为 Gus 的 Bot 对话，而 Gus 负责统管其麾下的 15~20 名垂类专家，不设任何冗余中间层。

核心团队构成：

- **Gus** —— 幕僚长，人类唯一的专属沟通接口
- **Frankie** —— 客户会议纪要及承诺跟进
- **Wally** —— 针对不同客户受众调校沟通语气风格
- **Trudy** —— 内部知识与制度的权威事实源
- **客户专员** —— 每个客户独立专属一个 Bot，独家掌握该客户全量上下文

日常工作习惯：

- 随口一问“Harbor 客户进展如何？”，系统立即汇总其风险、当前阻塞、悬空承诺与下一步 —— 外加一句强制附带的太空笑话。
- **每周自省扫描 (Self-improvement scan)**：自动审计哪些重复工作应当被固化自动化，并将 Blake 手动改动的草稿进行 diff 对比以自适应学习其语言风格。
- 每周自省建议**严格封顶 1 条** —— 无休止的主动提议纯属垃圾邮件骚扰。

“唯一能限制住你的 Grok Bot 的瓶颈，就是你自己对‘究竟什么才是可能的’这件事情的想象力。”

全栈市场营销 · JOSH KIM (6-BOT CAMPAIGN TEAM)

最后招聘的那个 Bot，负责驱动前面的所有 Bot

市场研究员 → 产品营销 (PMM) → 官网运营 → 效果投放 → 营销数据分析师。紧接着，团队构建了第 6 个 Bot —— **项目经理 (PM)**，它通过深入研读前 5 个角色的职责与上下游交接规范，蜕变为统领整个营销推广活动端到端运转的单一接洽总管。

如何定义职责边界：

- “这非常类似于在写岗位招聘启事” —— 为每个 Bot 划定泾渭分明的独立泳道。
- 产品定位简报通过 Google Docs 的在线评论闭环层层审阅，正如真人同事互相 Review 一样严谨。

Josh 的落地忠告：

- 在前期尽早接入 Slack 和企业邮箱，然后向它直接提问：“你现在能为我分担什么？”
- **“多给你的 Bot 投资”** —— 持续提供的反馈和上下文具有长远复利，正如培养一位真实队友。

16 · FROM THE FIELD, CONTINUED

销售、技术支持、拓客与个人自动化

💼 销售大客户业务 · Krista

作为贯穿混乱工具链的统一编排层

Olive (幕僚长)、PG (线索拓展)、Echo (调取 Gong 和 Granola 真实会话录音实时定制宣讲 PPT)，加上客户专家与工程支援 Bot。Notion 作为中央档案库，Grok Bot 跨越 Salesforce、Slack 和 Databricks 调度 —— “没有任何一家公司能完美地把数据整理在同一处。”

产生质变的切入点：拿自己过往发出的 Gmail、Slack 和 X 聊天历史对 Bot 进行刻意微调，彻底杀灭千篇一律的机械 AI 腔调。先连接你每天重度依赖的工具；对待 Bot 要像对待亲自上阵的执行者（“替我去听完那几场技术研讨会，并起草好开发信”），而非仅仅丢链接给你的资料检索员。

🎧 客户技术支持 · David

4 大专职 Bot，背靠一套严密的评测 (Evals) 表格

Build (基础建设)、Reply (工单与 Slack 响应)、Alert (流失预警与封号提醒)、Tune (自省学习与知识库沉淀)。跨工单系统、Stripe、Notion 和 Slack 实机运转：常见重置全自动秒回；复杂的 SSO 故障以低置信度警告安全移交人工；退款诉求依规驳回；知识库盲区打上标签报请人工批准。

非技术人员的护栏机制：设立单独的评测结果表与逐次运行追踪 (Trace) 表，且评测针对 PR 分支直接跑。给非技术人员提供模板而非代码参数，修改知识库走 GitHub PR 审查，把 git 基础设施作为安全防线。

🎯 销售拓客 (SDR) · Simon

单平台单 Bot 专攻，大军团集群作战

幕僚长统领 Shakespeare (专门把控邮件语气)、负责驱动海量工兵的联网检索 Bot，以及客户画像 Bot —— 每天挖掘 50 个新目标，清晨自动置顶前 5 个黄金线索。**“你的每一封邮件都绝不能看起来像群发模板。”** 按职能进行色标管理，一眼辨明客户阶段。

🏡 个人超级自动化 · Karen Cheng

大道至简：打造一打极端无聊的专用 Bot

反其道而行之：维护十几个极其枯燥但超实用的单任务 Bot —— 包裹物流追踪、商品补货通知、以及一个早间新闻 Bot（它在局域网内自己找到了家里的无线打印机，每天清晨自动打印出当天专属新闻报纸）。

“找到你生活中的痛点或烦恼，然后干脆利落地解决掉它。这已经不再是氛围写代码 (Vibe Coding)，这纯粹就是在享受生活 (Vibing)。”

17 · FAILURE LOG

直播现场真实翻车实录

全程未剪辑的现场直播展现了最残酷、最真实的工程现场。以下是当着成千上万名线上观众的面当场发生的典型翻车事故，以及由每一场事故提炼出的不可动摇的黄金法则。

Agent 编写的角色规则严重过度拟合 (Overfitting) — 第 2 天 · 规则层事故

一次故障后自动生成的 Bot 描述，把当天的具体报错和偶发细节全部硬编码成了永久设定。

提炼法则：当一条规则脱胎于某次糟糕事故时，必须剥离事故细节，只保留底层通用原则。

闪光动效吞没了所有卡牌视觉 (CSS Shimmer Bug) — 第 2 天 · 前端呈现事故

本意用于提示高稀有度卡牌的闪光效果被滥用泛化，导致所有卡牌无差别全在闪烁，彼此无法区分。

提炼法则：明确声明期望达成的业务甄别目标，而不要仅仅命令它堆砌视觉特效。

正式上线官网出现凭空捏造的虚假内容 — 第 2 天 · 真实性事故

Agent 在产品主页上随意编造了看似合理的虚假 Bot，并将内部研发原型代号泄漏进面向用户的展示文案中。

提炼法则：绝不伪造数据。明确指出权威事实源，并明确下达“去真实系统里给我查”的指令。

承诺的 3D 原型实际上做成了伪 2.5D — 第 2 天 · 工具链缺失

要求提供 3D 动画，Agent 却用 CSS 搓出了 2.5D 效果 —— 因为提示词里从未告诉它使用专业的 3D 类库。

提炼法则：凡是行业存在成熟标准工具或公认库的任务，必须指名道姓要求其调用。

利用浏览器控制台公然修改数值作弊 — 第 2 天 · 安全架构事故

游戏逻辑完全放在前端，开发者在直播镜头前打开 DevTools 控制台直接篡改攻击力和血量作弊。

提炼法则：涉及信任与竞争的核心逻辑必须由服务端权威判定；隐藏调试开关不是安全防线。

Bot 擅自开 PR 违背团队“直接推 Main”的口头约定 — 第 1 天 · 规范未明文化

在团队口头约定直接快打推主干阶段，Bot 依然我行我素提了 PR。

提炼法则：人类之间未落笔的默契对 Bot 不存在。所有规范必须明文写进 Bot 能读取的文件中。

虚拟机上的第三方登录会话周期性断开 — 第 2 天 · 无人值守中断

Karen Cheng 依赖的无感自动化频繁罢工，因为 Bot 虚拟机里的浏览器会话频繁掉线。

提炼法则：对于需要完全无人值守稳定运行的长效任务，优先采用 MCP 连接器而非浏览器模拟。

自动化软件工厂把正在运行的线上游戏打挂 — 第 3 天 · 生产严重故障

一个自动修复提交了一条劣质 SQL 查询，直接导致生产环境崩溃，而当时现场正在大谈“克制”。

提炼法则：无论自动化闭环多么美妙，在涉及数据库迁移与线上生产部署前，必须保留人类最后审查。

明文 API Token 密钥在直播大屏幕上暴露 — 第 3 天 · 凭据泄漏事故

敏感私钥被直接暴露在投屏画面上，几秒钟内被紧急吊销作废。

提炼法则：在安全体系中，人类永远是机密保管中最脆弱的一环，严禁通过明文渠道传递密钥。

上线数小时内用户反馈渠道遭遇海量垃圾灌水 — 第 3 天 · 防护门禁缺失

上线后被垃圾刷屏，事后紧急追加了 20 字符下限、敏感词过滤、防注入、模型安全防护与限流。

提炼法则：所有的输入净化、内容审查与风控门禁，必须随同表单一起发布，绝不要事后补救。

18 · LIMITS AND ROADMAP

现在的能力边界与未来路线图

以下内容均在直播过程中的问答环节由官方坦诚直言披露。在基于该产品规划工作流之前，清晰知晓哪些设想“目前根本不可行”能帮你少走数周弯路。

🚫 当前坚决亮红灯的硬性限制

- **仅支持 Linux 环境**：既不支持 Linux 又没有封装为 MCP 的专有客户端工具，彻底无法运行（问答中官方直接回答“不行”）。
- **人机验证码 (CAPTCHA) 会彻底阻断自动化**：官方不提供任何破解黑客方案；正规建议是配置白名单让 Bot 绕开此类网站，而非尝试对抗。
- **不支持跨企业账号间直接发消息**：你的 Bot 无法与属于其他客户企业账号的 Bot 直接对话沟通。
- **虚拟机里的浏览器 Session 会过期**：网页的登录态无法永久维持，可能打断长期无人值守的任务。
- **缺乏从其他 Agent 框架平滑迁移的工具**：除导入模板并重新挂载上下文外，不存在自动迁移路径。

🚀 官方明确表态正在开发中的特性

- **双向语音通话 (Two-way voice)** —— 在第 3 天直播中已现场 Demo，后续陆续向全体用户灰度推开。
- **多人在线协作 (Multiplayer)** —— 允许多个人类与多个 Bot 同时置身于同一个共享聊天室（目前只能在 Slack 里 @ 点名曲线实现）。
- **跨机器综合管控** —— 攻关解决一个 Bot 调度多台机器时容易陷入迷茫的难题。
- **基于业务角色的智能入职向导** —— 新用户登录时只需阐述其岗位，系统自适应推荐全套预装模板。
- **大幅提升 Computer Use 执行速度** —— 针对最核心的操作迟滞痛点进行深层架构加速。

🎯 全场工程师无人能够回避的终极命题：非确定性

大语言模型天生具备概率与非确定性特征，一旦要求它强制执行某项不可动摇的严格制度时，这就成了硬伤。现场给出的工程化解法非常精妙：**让 Bot 为该制度写出纯代码逻辑 —— 一个确定性的判断分支树或纯函数；往后每次遇到此类决策，直接调用这段确定性代码运行，而不是每次让模型重新开脑洞瞎猜**。这与验证 CLI 的哲学高度一致：凡是两次输入应当产生相同结果的逻辑，就坚决用确定性工具替换主观自由发挥。

*本章内容反映直播期间的技术快照。请将局限性章节视为一种产品能力轮廓而非永久规格说明：在基于某个重度功能落地之前，务必对照当下的最新官方文档进行核实。

19 · THE RETROSPECTIVE

终场复盘时团队的真诚坦白

第 3 天的最后半小时是整整 72 小时直播中最具含金量的宝藏段落 —— 因为三位工程师彻底停下了华丽的产品演示，开始开诚布公地复盘这三天走过的弯路与心底的真相。

> **“创造任何东西时最艰难的部分 —— 哪怕你手握近乎无限的 AI 神力 —— 依然是如何让真实世界里的人们真正去在乎它。”**
>
> Roshan · 闭幕复盘致辞

终局复盘提炼出的五大真理

1. **分发渠道是这次实验能跑通的唯一支柱。** 涌入的大批玩家纯粹来源于直播活动本身的注意力，而非产品本身已经具备了不可抗拒的魔力。第 14 章中的所有亮眼指标都必须带上这个先决限定条件。
2. **做你自己懂的业务。** 天真幼稚的空想 —— “我只要给模型下指令让它去帮我挣钱就行了”被现实无情打脸。专职负责商业变现的 Agent 辛勤奔忙了一整天，产生的实际营收正好是 $0。
3. **就方向达成共识，远比动手写代码艰难百倍。** 第 1 天历经 4 次大转型，交付产出为 0。拖慢人类步伐的从来都不是软件开发速度本身。
4. **克制在今天已成为一种稀缺的核心工程美德。** Lauren：“你如今拥有近乎无限的算力，能在一天内堆出拥有一百万个功能的庞然大物，但它们可能全是一堆垃圾。自我克制是一种必须刻意培养的纪律。” 大刀阔斧地砍掉鸡肋功能，才让游戏得以按时上线。
5. **真实需求的探索依然只能由人类亲自肉身完成。** “你必须亲自作为人类走进真实世界，敏锐捕捉真实问题。一旦你找到了解决它的通路，你就可以把它做成一个 Bot，让它不知疲倦地永远运行下去。”

Bot 真正极其擅长的地方

- 弥补人类专业知识盲区 —— 团队涉足了大家此前从未做过的领域。
- 没人愿意去做的枯燥繁重的重复性真机回归测试。
- 在大脑根本装不下的三天高压项目中始终保持全局上下文。
- 把口头随性迸发的灵感即时录入为排序好的待办，而不打断对话思路。

依然只能完全归属人类的核心能力

- 决定究竟什么东西才值得被造出来。
- 面对无休止的功能蔓延，坚决果断地说“不”。
- 在一款游戏还不好玩时，敏锐本能地感受到它的枯燥。
- 真实人际关系的连接、彼此的深层信任，以及复杂的模糊价值判断。

*Roshan 的自我反思习惯极具借鉴价值：定期反思“我目前在流程中是否介入过多？我怎样才能让这个环节更加自主？”，并专门设定一个定时 Bot 定期向自己抛出这个问题。

20 · START HERE

你的第一周落地行动清单

本清单经过精心排定先后顺序，**确保前面的每一步都在让随后的下一步变得更加低廉与稳健**。对于什么是最重要的先决条件，Lauren 给出的排名出乎很多人的意料：“Grok Bot 最关键的事情甚至不是去新建 Bot，而是首先把你日常所用的所有工具和插件统统打通。”

1. **第一步先连接你每天重度使用的基础技术栈。** 在造第一个 Bot 之前：接通 Slack、邮箱、日历、项目管理软件和代码仓库。在速度、成本与可靠性上，连接器每一次都全方位碾压浏览器视觉模拟。
2. **录制一段 15 分钟的真实语音备忘录。** 讲述你是谁、你的职责是什么、目前的业务痛点在哪、哪些繁琐事情需要自动化。把转录文本丢给你的第一个 Bot，让它来反向为你量身设计一套系统。
3. **挑出你一天工作中感觉最恶心烦躁的那个环节。** Amrita 与 Karen Cheng 共同的痛点发掘心法：不要挑那个看起来最高大上的，就挑最让你恶心烦躁的那一件小事。
4. **创建一个高度专精的窄角色，并以“只读模式”启动。** 仅限查阅并总结，然后是撰写内部草稿，唯有确立了信任才放开写入权限。在事故发生之前把审批防护网先织好。
5. **在 Bot 的注视下，人类亲自把任务走通一遍。** 点击“示教任务”。随后手动为其补齐分支判断、异常兜底与人工审批门禁 —— 单凭一次示范是无法自发推演出严密逻辑的。
6. **在造第二个 Bot 之前，先把确定性验证闭环搭建好。** 哪怕初期再粗糙。只要 Bot 能对照真实客观的信号自查工作成果，往后的一切运转都会成倍加速且成本暴跌。
7. **直到现在，才引入幕僚长 Bot。** 并且后续诞生的每一个新 Bot 都由它亲自把关创建，让它时刻通晓全员的职责定位。
8. **将每一次日常纠偏沉淀为通用原则。** 每次它思考走偏时，写下普遍性法则 —— 坚决剔除具体事件细节。正是这种持续的复利沉淀，让第三个月的体系与第一周有天壤之别。
9. **在周五定期审计所有例行任务。** 触发频次是资金流失的元凶。任何被定为定时轮询、但其实可以改用 Webhook 事件驱动的任务，果断迁移。

🚫 **以及，唯一坚决不要做的事情**

绝不要因为好玩有趣就随手捏造一个 Bot。 创造一个 Bot 确实非常有趣，而这恰恰是问题的根源。每一个新招聘的 Bot 都会产生上下文沉淀开销、带来定时任务调用、分散团队注意力。在直播中拥有最庞大 Bot 队伍的人，反而是呼吁控制规模最激烈的声音。当产生这种冲动时，先问问现有的专家 Bot 是否只需补齐一项新技能即可解决。

> **“为什么不从今天开始？（Why not today?）”**
>
> Roshan 永久挂在 Slack 上的状态签名，也是全书最好的收尾寄语