《AI原生工作报告2026》

驾驭AI

从信任鸿沟到可靠协作的十个关键词

出品：腾讯研究院 AI原生工作团队



我们正在经历一场前所未有的协作革命。人与AI之间，正在生长出一种全新的信任关系，它不来自完美，而来自持续的校准与驾驭。这份报告用十个关键词，记录我们正在走的那条学习曲线。

目录

序言

信任鸿沟：可靠不等于不出错，可靠是出错之后还能接住 ---------- 05

09

关键词1 驾驭工程

人设计环境，AI在环境中执行；核心范式跃迁

13

关键词2 记忆

精确遗忘比完美记忆更重要

17

关键词3 技能

最好用的，是自己持续维护的skills

21

关键词4 评估

可靠的评估与反馈是方向盘，是质量的前提

25

关键词5 上下文

停止把你的人生故事放进去

29

关键词6 工作流

人类工程师始终是这场演出的导演

33

关键词7 多智能体

用好一个，再用多个，可以打开更大的世界

37

关键词8 加法偏见

加法是能力，减法是智慧。

41

关键词9 去技能化

如果有一种技能我们输不起

45

关键词10 知识工程

方法会过时，工具会迭代，最终留下什么

研究团队

顾问	司　晓 | 腾讯研究院院长				
	杨　乐 | 腾讯研究院副院长				
	杨　健 | 腾讯研究院总顾问				
	冯宏声 | 腾讯研究院高级顾问				
研究策划	李瑞龙	袁晓辉			
报告主笔	李瑞龙	曹士圯	余 一	袁晓辉	
研究&实践	袁晓辉	李瑞龙	曹士圯	余　一	王　鹏
	刘莫闲	王　强	吴朋阳	白惠天	陈东明
	陈维宣	陈楚仪	胡　璇	贺泽轩	张鸿茹
	朱　敏	袁　歆	霍浩华	丁　阳	朱于增
	傅育生	陆诗雨	唐嘉晗		
产品支持	黄广民	孔德远	汪晟杰	设计支持	崔　昭

序

信任鸿沟：可靠不等于不出错，可靠是出错之后还能接住。





司晓

腾讯副总裁 腾讯研究院院长

每个用AI的人都在走一条学习曲线。从惊艳到怀疑，从怀疑到找到自己的节奏。这条路没有捷径，但有方法。我们从这里开始聊。

84%采用 vs 29%信任
为什么用了≠用好了

2023 年，全球最大开发者社区 Stack Overflow 的年度开发者调查显示 AI 编程工具的信任度是 40%。两年后，采用率从 70% 升到了 84%。信任度降到了 29%。用得越多，信得越少。为什么我们很少见到一个技术是这样的?

这个不是简单的信任问题。Stack Overflow 自己的分析给了一个精准的判断：这是一条学习曲线，伪装成了信任问题。这句话值得琢磨。以当前AI应用最广的领域为例，软件工程师的职业训练建立在确定性上。写同样的函数，传同样的参数，得到同样的结果。然而，当 AI 来了。同一个问题问两遍，两个答案，两种结构，两套取舍方案。都能跑。对于严谨的工程师来说，这样的特性，需要一个适应过程。这种感受不是程序员独有的。律师期望同一条法规的检索结果稳定一致，医生期望同一组指标指向确定的诊断方向，金融分析师期望同一套参数产出可复现的估值。概率性系统进入确定性职业的地盘，遇到的不是能力质疑，是一种更原始的不适：认知摩擦。但信任低，不意味着人们在验证。就目前来说，人们对于AI的信任偏差并没有形成体系的方法论来约束它，利用好它。

96%不信，48%不查

另一组非常矛盾的行为数据。代码质量平台 Sonar 的调查揭了一组更值得琢磨的数字：96% 的开发者不完全信任 AI 代码的功能正确性。但只有 48% 的人在提交前始终检查。几乎所有人都说“我不信”，一半人说完就点了提交。为什么不查？不是因为懒。38% 的开发者觉得审查 AI 代码比审查人类代码更费力。为什么更费力？AI 产出“看起来正确但不可靠”，不像语法错误会让构建直接失败，AI 写出来的是看着合理的逻辑，bug 藏在里面，需要更高的专业判断力才能揪出来。

这是一个隐蔽的成本转移：AI 把“写”的工作量砍下来了，但把“查”的负担顶上去了。查的成本一旦超出预期，人们的反应不是更仔细地查，是不查了。嘴上的不信任是安全的。手上的放行才危险。

更自信，但更差

除了上述的行为失控，还有更麻烦的感知失真，也就是说，你甚至不知道自己的判断已经偏了。斯坦福大学 Dan Boneh 团队在 CCS 2023（计算机安全顶会）上发了一项随机对照实验。用 AI 助手的参与者在多数安全编程任务中写出了更多不安全的代码。写出不安全代码的那批人，对 AI 的信任评分反而更高。你越觉得它帮了你，它越可能在坑你。这时候你可能会说，经验丰富的开发者是否可以避免这个问题？AI 安全评估机构 METR 在 2025 年做了另一个实验。16 名经验丰富的开源开发者，在自己贡献多年的仓库上干活，用的是前沿模型。结果：实际慢了 19%。自我感觉快了 20%。感知和现实之间差了 39 个百分点。研究团队也指出，在不熟悉的代码库或简单任务上，AI 可能确实有帮助。但在高质量标准和复杂隐含要求的场景下，验证和整合 AI 输出的开销把速度收益吃回去了。两种情况下，人们对自身表现的判断都偏向乐观。要么留下了风险代码，要么增加人工检查，反而导致效率下降。用了，并不一定等于用好了。

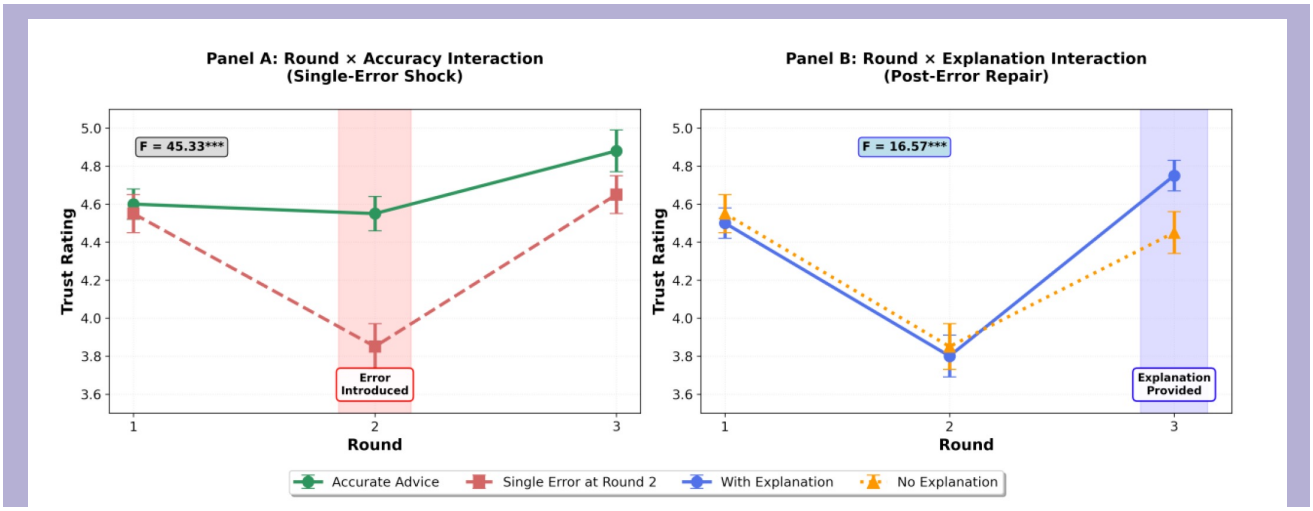
300 行代码，居然全错

在更多的真实场景里呢？一位工程师在 FDA（美国食药监局）监管的医院基础设施中用 Claude Code，AI 写了 300 行 OpenTofu 基础设施代码。语法完美。逻辑表面合理。通过了验证流程。但是，它引用的资源和配置，有很大一部分是编的，事实上并不存在。在许多专业领域当中，AI应用需要更为谨慎，因为在基础模型预训练阶段，这些资源配置和边界条件，没有在训练数据中覆盖。大量的实际应用，是需要做边界判断的，不是仅仅考虑语法逻辑，语法 AI 早就会了。AI 知道语法。不

知道的是这个 FDA 环境下哪些资源存在、哪些配置合规、哪些边界不能跨越。这不是知识问题，是判断力问题——它不理解上下文，不理解后果，不理解地“如果这样做会怎样”。法律合规、金融审计、医疗决策、科研创新，同一种困境。训练数据覆盖不了特定环境的边界条件。它不知道自己不知道。而语法完美但语义虚构的产出，恰恰最难被发现，因为它通过了所有表面的检验。

一个必经的过程，失败后的信任比从未失败的更强

如何重建信任，更准确的说，应该沿着学习曲线与AI更好的协作，我们再来看首尔国立大学做的两个实验，样本分别为 189 人和 294 人。他们发现信任不是一个静态指标，而是一条有形状曲线：第一阶段，形成。初次接触，基于能力线索建立期望，通常偏高。第二阶段，冲击。一个可见错误，信任断崖式下降。人对 AI 的容错度比对人低得多，研究者称之为“完美自动化图式”，你下意识觉得机器就应该是对的，有幻觉即不可用。第三阶段，修复。解释为什么出错，指出系统边界。信任部分恢复。意外的发现在第三阶段：修复后的信任可以超过初始基线。经历过错误并被正确解释的信任，比从未经历过错误的信任更结实。研究者称之为“信任加速悖论”。你信任一个医生，不是因为他从来没误诊过，而是因为你见过他发现误诊后怎么纠正、怎么坦诚、怎么调整方案。可靠不等于不出错。可靠是出错之后怎么处理。所以，我们应该做的，从来不是追求完全信息、零幻觉、零失败，而是设计“可控失败+透明修复”的流程。



图：人机交互中的信任动态：单次错误冲击与事后解释修复

图片来源：https://doi.org/10.3390/bs15101370

可控性的分寸在哪里

软件工程顶会 FSE 2026 的一篇论文提出了依赖-控制二维框架。22 名开发者访谈，信任问题可以用两个轴来画：纵轴是依赖程度，横轴是控制程度。“甜点”在平衡控制与适当依赖的交叉点。具体什么样？AI 生成初始测试场景，开发者改进，测试驱动开发的迭代中 AI 和人交替工作，人始终在驾驶座上。Vibe coding（氛围编程，即“你来写我不管了”）在矩阵的另一端。你交出了控制权，同时也交出了理解力。不同的工具形态天然推向不同的依赖水平。关键不在于用哪个工具，而在于使用过程中你保留了多少主动检查和决策的环节。经常有朋友问，研究院为什么选了面向开发者的 CodeBuddy IDE，而不是对话式的 OpenClaw等Agent工具。原因正是这个分寸：我们的研究日常复杂度高，需要在过程中持续审查、持续介入，IDE 形态让我们留在驾驶座上。而对大多数白领场景，Wordbuddy 的对话形态更友好、上手更快。工具没有高下，匹配你需要的控制粒度就是对的。另外，配套的一个脑电实验发现，长期使用 AI 的人在神经和行为层面持续表现不佳。研究者管这叫“认知债务”。你以为在省力，其实在预支判断力。过度依赖还有个副作用：开发者花时间调提示词，而不是和同事讨论问题。跟机器对话多了，跟人对话就少了。因此，我们还会在人机协作的过程中，有意识的保留人工讨论环节与比例，以确保 AI 和人之间的互动的同频共振，维护人的判断力、理解力，实现对AI的可持续的驾驭能力。这是一个健康的人机协作组织应具备的长期能力。

并不是简单推广，就有回报

把镜头再拉远一点，组织层面的故事更让人难受。以下是一组权威数据，MIT 斯隆管理学院的数据：95% 的企业 AI 试点未产生可衡量的业务回报。88% 的组织在用 AI，但仅 7% 真正整合进了业务流程。BCG（波士顿咨询）和哥伦比亚商学院发现了高管和员工之间的感知断层：76% 的高管认为员工对 AI 充满热情，实际只有 31% 的一线员工有此感受。42% 的高管层承认 AI 采用正在“撕裂公司”。31% 的员工承认主动破坏 AI 推广。

《福布斯》技术委员会给了另一个视角，四类抵触，每一种都是组织在发出信号：一、工具抵触：试过了，发现不好用。比如，法律团队拒绝用合同分析 AI，可能不是守旧，是出于保护公司的优先级。二、策略抵触：AI 部署的位置和价值位置不匹配。半数预算流向销售和营销工具，但数据显示最高回报来自组织、后台自动化。三、信任抵触：领导层说“AI 是来增强你的”，同时宣布裁员目标。很难让员工信服，这不是恐技心理，是对矛盾信号的理性回应。四、能力抵触：只有 44% 的人接受过 AI 培训，甚至57% 的人不愿告诉团队自己在用 AI，怕暴露自己“用了但不确定用得对”。MIT 研究者的总结一语中的：规模化的核心障碍不是基础设施、不是监管、不是人才，是学习。

腾讯研究院在自身的 AI 转型实践中也注意了这一点。我们的经验是，提供一个安全的、持续的讨论空间，打消团队成员对 AI 的抵触情绪，帮助大家把握迭代的方向，而非传授具体的 AI 技术，更难，也更重要。在团队里公开讨论 AI 的使用。不是搞培训，是坦诚的交流，同事之间，同事与AI之间。信任不是培训出来的，是在透明的对话中长出来的。

为什么要写这份报告。

过去一年，我们和很多团队聊过，发现大家遇到的困惑高度相似，不缺工具，但不确定怎么用对。我们自己也在摸索，踩过不少坑。把这些经验和教训整理出来，是希望 AI 带来的效率红利不只属于少数技术团队，而是能被更多人、更多组织真正用上。信任鸿沟不是一个需要解决然后翻篇的问题。它会一直在那儿。这不是坏事。对一个概率性系统保持警觉，本身是健康的。问题从来不是“你信不信 AI”，而是“你的信任是否经过校准”。理解这种张力是第一步。

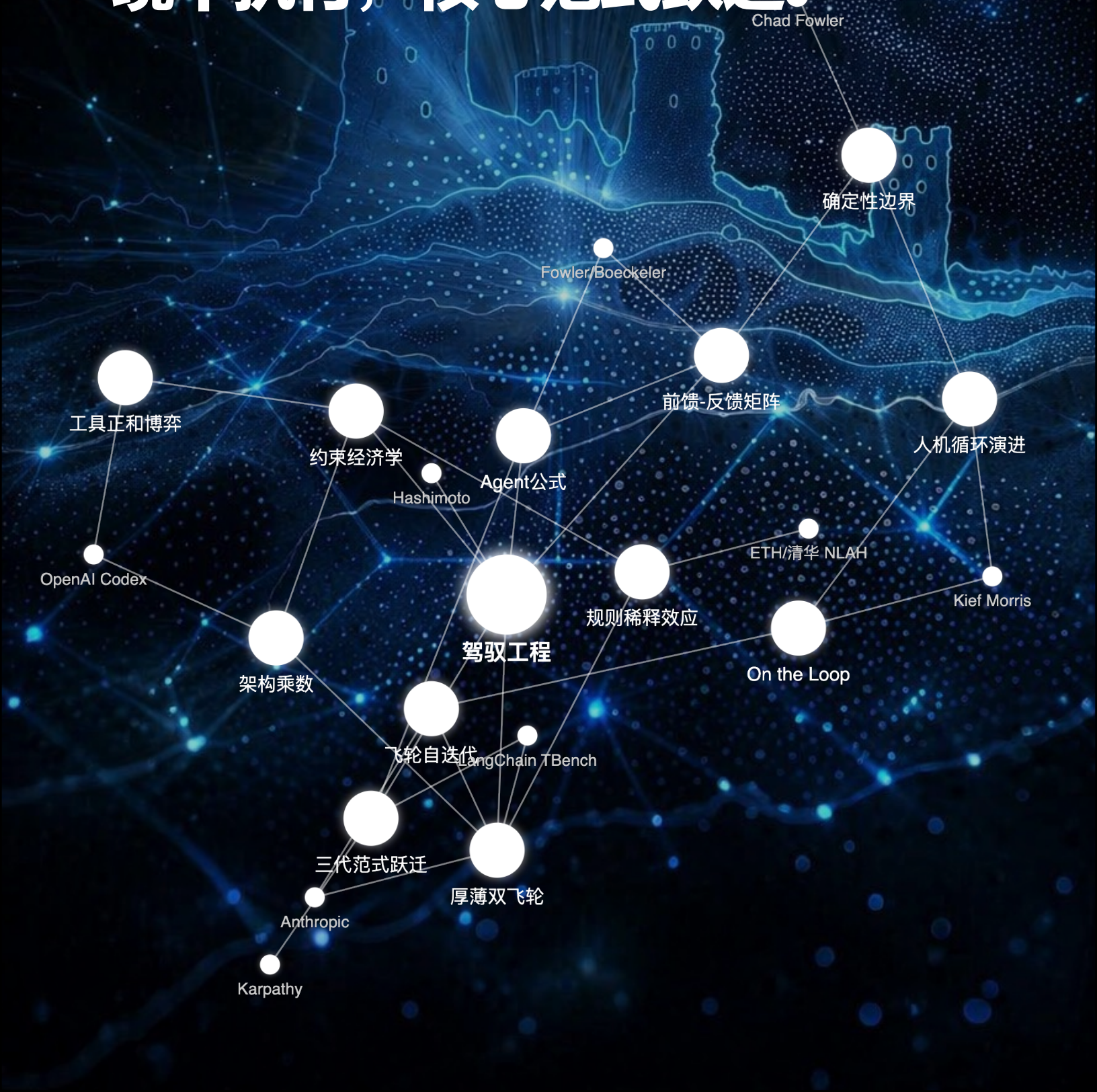
接下来，我们会通过十个关键词展开讨论。

参考文献：

1. Stack Overflow, "Mind the Gap: Closing the Developer AI Trust Gap", 2026-02-18. https://stackoverflow.blog/2026/02/18/closing-the-developer-ai-trust-gap/ ↩
2. Sonar, "The AI Trust Gap: Why Code Verification Matters", 2026-01-22. https://www.sonarsource.com/blog/ai-coding-trust-gap
3. Perry N, Srivastava M, Kumar D, Boneh D, "Do Users Write More Insecure Code with AI Assistants?", ACM CCS 2023. https://arxiv.org/abs/2211.03622
4. METR, "Measuring the Impact of Early-2025 AI on Experienced Open Source Developer Productivity", 2025-07-10. https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
5. Heinan Cabouly, "The AI Wrote 300 Lines of Infrastructure Code — None of It Was Real", 2026-04-15. https://medium.com/@heinancabouly/the-ai-wrote-300-lines-of-infrastructure-code-none-of-it-was-real-4cf3fd7c0a4a
6. Han J, "Trust Formation, Error Impact, and Repair in Human – AI Financial Advisory", Behavioral Sciences 15(10), 2025. https://doi.org/10.3390/bs15101370
7. Ferino S, Hoda R, Grundy J, Treude C, "Towards an Appropriate Level of Reliance on AI", FSE 2026. https://arxiv.org/abs/2604.10530
8. Chronus, "Why AI Adoption Fails: Enterprise Barriers AI Leaders Ignore", 2026-03-18. https://chronus.com/blog/why-ai-adoption-fails-enterprise-barriers-to-ai-leaders-ignore
9. Forbes Technology Council, "Your Resistant Employees Know Why Your AI Adoption Is Failing", 2026-02-05. https://www.forbes.com/councils/forbestechcouncil/2026/02/05/your-resistant-employees-know-why-your-ai-adoption-is-failing/

关键词一

驾驭工程：人设计环境，AI在环境中执行；核心范式跃迁。





袁晓辉

腾讯研究院副院长、资深专家

> 同一个模型，换一套周围的基础设施，表现天差地别。决定AI好不好用的，不只是模型本身，还有你为它搭的那个运行环境。

决定 AI 好不好用的在用户与模型之间，还有环境层

AI 应用开发公司 LangChain 做过的一个实验。同样的一个前沿模型，权重不动，只改模型周围的基础设施，在 TerminalBench 排行榜上的名次跃升了二十多位。更直观的感受，用过多种Agent工具的朋友，会发现Cursor、Manus、Claude Code等等，底层模型即使是同一个，用起来天差地别。决定 AI 好不好用的，不只是模型本身。另一个同样重要的因素，是模型周围的系统设计得有多好。这套系统，2026年有了一个名字：驾驭工程（Harness Engineering）。

新人入职

要理解 Harness，先想一个场景。你雇了一个极其聪明但完全不了解你们公司的新人。能力超强，什么都会。但不知道代码库长什么样，不知道哪些模块不能碰，不知道提交前要跑哪些测试，不知道老板最讨厌什么风格。两个选择。一、每次做完一件事你亲自检查；二、把规矩写下来、把自动检查跑起来、把“什么能做什么不能做”编成清单。第一个叫人工盯梢（Human in the Loop）。第二个叫驾驭工程（Harness Engineering）。基础设施工具公司 HashiCorp 的联合创始人 Mitchell Hashimoto 在2026年2月的博客里使用并引爆了这个词。要表达的意思，其实可能朴素到不能再朴素：每次发现 Agent 犯了一个错，你花时间把解决方案工程化，让它再也不能犯同样的错。就这么简单。日积月累，约束、反馈循环、自动验证、文档同步、架构强制，加在一起，就是 Harness。Agent = 模型 + Harness，顺着这个思路，软件工程领域的标志性人物 Martin Fowler 团队给出了这个定义。这样大家应该可以理解，为什么Agent时代，Harness这个概念会火，学历再高、再聪明的原工，也得通过跟老板的交互，来实现更好的工作表现。

动态演进，比想象中快

2023到2024年，提示词工程（Prompt Engineering）。研究怎么跟 AI 说话。调措辞，加示例，试格式。有用，但天花板低——优化的是一次性输入。

2025年，上下文工程（Context Engineering）。光说清楚不够，AI 还得看到正确的背景信息。文档、代码片段、历史对话、工具调用结果。前特斯拉 AI 总监 Andrej Karpathy 推火了这个概念。本质还是优化“给 AI 看什么”。

2026年，驾驭工程（Harness Engineering）。质变。不再优化输入。直接设计 AI 运行的环境——约束系统、反馈循环、验证机制、状态管理。AI 在你造的办公室里干活，而不是你在邮件里反复跟它解释该怎么干。

XP（极限编程）早期实践者 Chad Fowler 给了一个精炼的公式：概率在内部，确定性在边界（Probabilistic inside, deterministic at the edges）。让 AI 在边界内自由发挥。边界本身严格、显式、机械化强制，但目标不是限制，而是获得边界内的效益提升。做个简单的类比，就像法律体系的逻辑一样。好的法律明确限制做事的边界、限制坏结果的产生、限制不同系统之间的冲突。为了你可以在范围内更自由地活动。Harness 就是给 AI 写的法律体系。

四个象限

这似乎有点抽象，我们可以再具体一点。Harness究竟是以什么样的形态存在的。全球性软件咨询公司 ThoughtWorks 杰出工程师 Birgitta Boeckeler 提出了一种我们认为目前相对比较系统且容易理解的分类体系。包括两个维度。控制方向：AI 行动之前预防，还是之后检测纠正。前者叫前馈（feedforward），后者叫反馈（feedback）。只有前馈，AI 记住了规矩但不知道有没有用对。只有反馈，AI 反复犯同一个错直到检测机制告诉

它。两个都得有。执行类型：确定性的，还是推理性的。确定性靠自动化规则，毫秒级，零误报。推理性靠 AI 审查，慢，但能看到更高层的语义模式。

Harness（整体概念）

└── 由四类组件构成：

├── 前馈 X 确定性 → 模板、规范等

├── 前馈 X 推理性 → AGENTS.md、设计原则

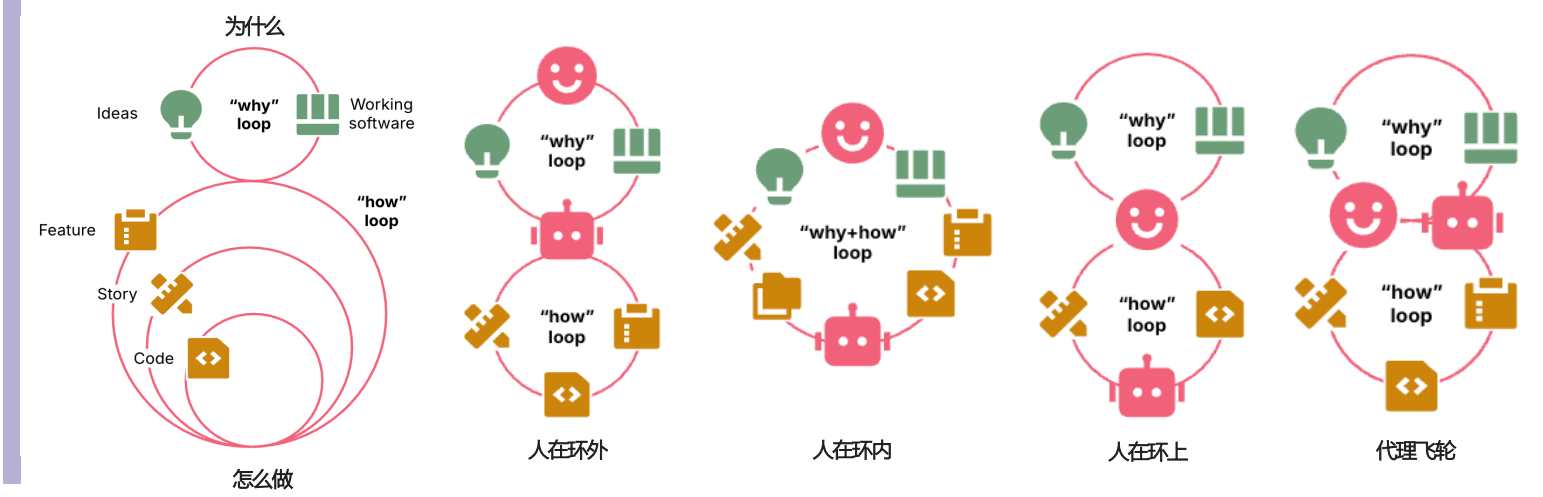
├── 反馈 X 确定性 → linter、测试、pre-commit hook

└── 反馈 X 推理性 → AI 互审、专家评审

两个维度叠在一起就是四个象限。前馈X确定性是模板和规范。前馈X推理性是设计原则和价值观。反馈X确定性是自动化检查。反馈X推理性是 AI 互审和专家评审。不管你写代码、写报告还是做分析，四个象限都适用。就目前我们了解到的情况而言：大多数人只做了前馈，很少有人做反馈，更少人将反馈建立成一套系统。

人在什么位置

另一个问题，人机协作。我们可以通过一张四阶段图来理解，摆正人类在人机协作、部署harness过程中的位置。第一阶段，人在环外（Outside the Loop）。比如，氛围编程。给 Agent 一个任务然后听天由命。第二阶段，人在环内（In the Loop）。审查每一行产出。安全了，但你成了瓶颈。很多人在这里得出结论“AI 没省时间”，审查吃掉了生成省下的时间。第三阶段，在人环上（On the Loop）。不再直接修产出物。不满意时，修产生产出物的系统。这是一个认知跳跃。很多人理解了但做不到，因为它要求你从做事的人变成设计环境的人。第四阶段，飞轮（Agentic Flywheel）。用 Agent 改进 Harness 自身。评估表现，推荐改进，审批后自动应用。Harness 开始自我迭代。在不满意的时候做什么，最能区分“人在循环中”和“人在循环上”。前者修产出物。后者修产生产出物的 Harness。



图：人机交互循环的四个状态：人在环外、人工参与、人在环中、代理飞轮

图片来源：https://martinfowler.com/articles/exploring-gen-ai/humans-and-agents.html ↩

约束多好还是少好？

不能一概而论。约束有三种完全不同的经济学。

规则约束。零和博弈。文本指令消耗上下文窗口。加一行就稀释其他行。苏黎世联邦理工和清华的研究：人工写的60行以内规则提升成功率4%，AI自动生成的200多行规则反而降低3%。Agent 太认真了，担心的不是忽略规则，是太认真地遵循，结果每个任务变得更难，规则并不是越多越好，要谨慎推进。

工具约束。正和博弈。自动化检查在上下文窗口之外运行。不消耗注意力，确定性执行。对于工具使用，一种做法叫“正向提示注入”（Positive Prompt Injection），自定义检查发现错误时，反馈信息直接包含修复方法。每次违规都是一次微型培训。对于工具约束，越强越好。

架构约束。乘数效应。不是一条一条告诉 AI “别做这个”，而是搭建一个更高效的架构。OpenAI Codex 团队的一篇博客中提到

他们把系统拆成固定的层级，每一层只能调用下一层，反向调用会被自动拦截，提交都提交不上去。3 名工程师，5 个月，零行手写代码，产出了百万行级的生产产品。这也是现在一些AI原生企业，产品更新可以实现周更、日更的原因。这些经验，对 Agent 来说是乘数，一旦写入，处处自动生效。

规则，越精越好。工具，越强越好。架构，是让前两者处处生效的乘数。约束是一个复杂工程，要看对象。对某个项目至关重要的约束，对另一个项目可能毫无意义甚至有害。

从 200 行到 50 行

腾讯研究院内部使用 AI 工作流的早期也经历过约束膨胀。大量 rules，项目规范、输出格式、禁止行为、偏好设置。规则越多，Agent 行为越不稳定。有些约束在跨项目使用时直接冲突。后来精简到核心规则。从200行砍到50行左右。表现反而好了很多。更重要的转变：从修 AI 的输出，转向修产生输出的系统。我们日常在搭建各类生产级工作流的过程，本身就是一个 Harness，文件持久化、进度追踪、阶段门禁、多 Agent 讨论。虽然不是代码层面的测试，但本质相同：确定性检查点，不依赖 Agent 的判断力。从 In the Loop 到 On the Loop 的跳跃比想象中难。需要克制直接改输出的冲动。

harness的厚薄争议

AI 社区有一场辩论。大模型派还是大 Harness 派。Anthropic 工程师 Boris Cherny 是 Claude Code 的创造者，他说架构是“模型上最薄的包装”，每三到四周重写一次。OpenAI 推理研究负责人 Noam Brown 更激进：在推理模型之上构建脚手架往往适得其反。AI 数据框架公司 LlamaIndex 的 Jerry Liu 站另一边“Harness 就是一切”。一个下午让15个大模型全部提升。Cursor 500亿估值全在 Harness 层，包括Manus出售，其实就是卖的也是 Harness 层。相关的辩论还在继续，但我们看到的实际情况是，两者都仍在发挥着非常重要的作用，是双飞轮：更好的模型让 Harness 复杂度可以删除，Harness 变薄。更好的 Harness 让模型在更复杂任务上可靠运行，释放模型能力。两个同时转。此外，组织规模也决定 Harness 厚度。一个人用 AI，薄 Harness 就够。一个团队，需要更多结构化约束。一个公司，需要 Harness 模板作为组织级基础设施。Harness 还有另一层作用，创造新的交互方式。模型变厉害了，可以衍生出更厉害的交互方式。我们认为未来还是会继续存在超级App，存在越来越丰富的交互方式，而不是让用户直接对接大模型。一些产品数据说明趋势：Manus 称开发过程中，6个月内重建5次，每次重建减少复杂度，但与此同时，更多的功能被开发出来。Anthropic 定期从 Claude Code 的 Harness 中删除规划步骤，新模型版本已经将该能力内化； 但随着开发推进，Claude Cowork、Claude Design引入了更多的harness。Harness 的可能性空间不是简单的随模型进步而缩小，它会移动，减少一部分，增加另一部分，获得进一步的能力提升。

一些待观察的问题

非编码场景的 Harness 方法论，目前没有成熟的通用框架，但我们判断它会先从有准确性要求的知识工作开始，法律、金融、研究等。这些问题在后续的关键词中继续讨论。

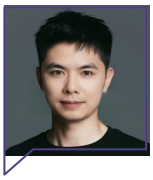
参考文献：

1. Daily Dose of DS, "The Anatomy of an Agent Harness", 2026. https://blog.dailydoseofds.com/p/the-anatomy-of-an-agent-harness
2. Terminal-Bench 2.0 Leaderboard. https://www.tbench.ai/leaderboard/terminal-bench/2.0
3. Mitchell Hashimoto, "My AI Adoption Journey", 2026-02-05. https://mitchellh.com/writing/my-ai-adoption-journey
4. Birgitta Boeckeler (ThoughtWorks), "Harness Engineering for Coding Agent Users", 2026-04. https://martinfowler.com/articles/harness-engineering.html
5. Chad Fowler, "Relocating Rigor — The Phoenix Architecture", 2026-01-07. https://aicoding.leaflet.pub/3mbrvhyye4k2e
6. Kief Morris, "Humans and Agents in Loops", 2026. https://martinfowler.com/articles/exploring-gen-ai/humans-and-agents.html
7. Mündler et al. (ETH Zurich/清华), "NLAH", 2026. https://arxiv.org/html/2603.25723v1
8. OpenAI, "Harness Engineering: Leveraging Codex in an Agent-First World", 2026-02-11. https://openai.com/index/harness-engineering/

关键词二

记忆：精确遗忘比完美记忆更重要。





李瑞龙

腾讯研究院AI原生团队负责人

> 你不会记住昨天午饭的每一口，但记得第一次吃到某道菜的惊艳。AI的记忆正在学同一课。核心挑战不仅仅是怎么记住更多，还要忘得更精确。

人的记忆不是硬盘。你不会记住昨天午饭的每一口，但你记得第一次吃到某道菜时的惊艳。认知科学早就知道这件事：遗忘不是系统故障，是设计特性，大脑通过主动丢弃低价值信息来保护高价值信息的检索效率。AI 的记忆正在学同一课。2026 年，记忆从“把对话历史塞进上下文窗口”升级成了一级架构组件。有专门的框架，有基准测试，有可量化的工程取舍。而最反直觉的发现是：记忆系统的核心挑战不是“怎么记住更多”，是“怎么忘得更精确”。特别是面向长期的记忆系统，至关重要。

跳过的那一步

记忆系统的基本动作一共有三个。写入（Write）、管理（Manage）、读取（Read）。大多数实现只做了写入和读取，跳过了管理，甚至乱放。一个笔记本，每天往里记东西，从不整理、从不删除、从不合并重复内容。三个月后想找上次看过的方案，几乎不可能。旅行时疯狂拍照的人都有体会，拍了两千张，回来不整理，照片不会再看了，跟没拍一样。管理包含四件事。修剪，把不重要的记忆降权或删掉。压缩，把冗长历史提炼成关键事实。巩固，把反复出现的信息从短期升级为长期。冲突解决，两条记忆矛盾时决定保留哪个。忽视记忆管理的代价已经被研究量化，名为 FadeMem 项目中，有一个多跳推理测试，没有记忆管理的 Agent 的信息检索准确率（F1 分数）仅为5.17，而有记忆管理的系统达到29.43，差距近5倍。

五条认知科学教训

回顾认知科学研究记忆已经 60 年了。虽然说，人脑是生物神经网络，LLM 是 Transformer，存储机制完全不同，但认知科学还是可以提供许多的设计启发， 包括现在诸多的记忆项目、插件其实都是基于这几十年的认知科学研究而构建的。

第一，容量在未来将不会是瓶颈，注意力才是。把全部对话历史塞进上下文窗口，相当于绕过了大脑的注意力过滤器，应该做的事情，应该是在模型看到信息之前就替它筛好。

第二，怎么存决定了能不能找到。逐字存是浅层处理，把对话压缩成摘要也只是换了个浅法。更为有效的是提取离散事实，进行深层处理，反而可以大大提升检索的准确率。

第三，只追加、不整理的记忆存储，最终会变成一台干扰引擎。认知科学里叫干扰理论（Interference Theory）：旧信息干扰新信息，新信息也干扰旧信息。当记忆规模逐渐增大，去重已经不是可选的清洁工作，是让记忆能用、好用的前提。

第四，遗忘。认知科学家 Robert Bjork 提出过“新废弃理论”：遗忘是大脑主动执行的自适应过程，低相关的内容随时间衰减，给高价值的信息腾出检索通道。

第五，巩固需要时间。不是所有短期记住的东西都该立刻升格为长期记忆。需要一个考察期，逐层提升。

关于最近各家模型公司提出的 AutoDream 类的功能，也就是做梦机制。名字叫“做梦”，实际用的是 LLM 推理。推理后的记忆可以提供泛化性，像一个人打完一个游戏换到另一个游戏，靠的不是肌肉记忆迁移，是推理抽象出来的策略。在一定程度上实现在线学习的效果，通过记忆推理适应新任务。

精确遗忘

精确遗忘比完美记忆更重要。如何管理，FadeMem 的研究项目试着把这件事工程化。这是一篇 2026 年初的 arXiv 论文，思路很直接：模仿 19 世纪心理学家艾宾浩斯（Ebbinghaus）的遗忘曲线，给 AI 记忆设计两层不同的衰减速度。长期记忆层衰减慢，半衰期约 11 天；短期记忆层衰减快，约 5 天。重要的信息被反复访问就会加固，不重要的自然淡出。结果似乎是另人满意

Method 方法	Critical Facts 关键事实	Contextual Info 上下文信息	Storage Used 已用存储
Fixed-16K	50.2%	44.8%	100%
LangChain	71.2%	65.3%	100%
Mem0	78.4%	69.1%	100%
MemGPT	75.6%	62.8%	85.3%
FadeMem	**82.1%**	**71.0%**	**55.0%**

图：LTI-Bench 在连续交互 30 天后的记忆保留分析

图片来源：https://arxiv.org/html/2601.18642v2

的：存储量砍掉 45%，关键事实保留了 82.1%。少记了将近一半的东西，真正重要的几乎没丢。更值得一提的是，项目还进行了消融实验，把各个模块逐个拆掉看效果变化。拆掉记忆融合模块（负责把相关记忆合并归纳），准确率暴降 53.7%，有效的说明了，光是“忘掉不重要的”还不够，还得把留下来的信息整合起来。看到这里，是不是觉得项目很完美？在一些场景，是的，比如生活日常、创意写作，对于关键事实并不要求100%精确的时候，这个项目可以大幅提升工作效率，并带来更少的算力消耗；但不是所有场景。82.1%意味着还是有17.9%的关键事实被遗忘了。如果这是在医疗领域，那17.9%包含一条药物过敏信息呢？因此，还要提醒，记忆策略必须根据场景区别对待。日常助手忘记上周推荐的电影问题不大。医疗、法律、金融，遗忘可能是灾难。遗忘是有条件的功能。条件是被遗忘的信息不具有安全关键性。对于特定数据，需要更精确的遗忘策略。

三条路线

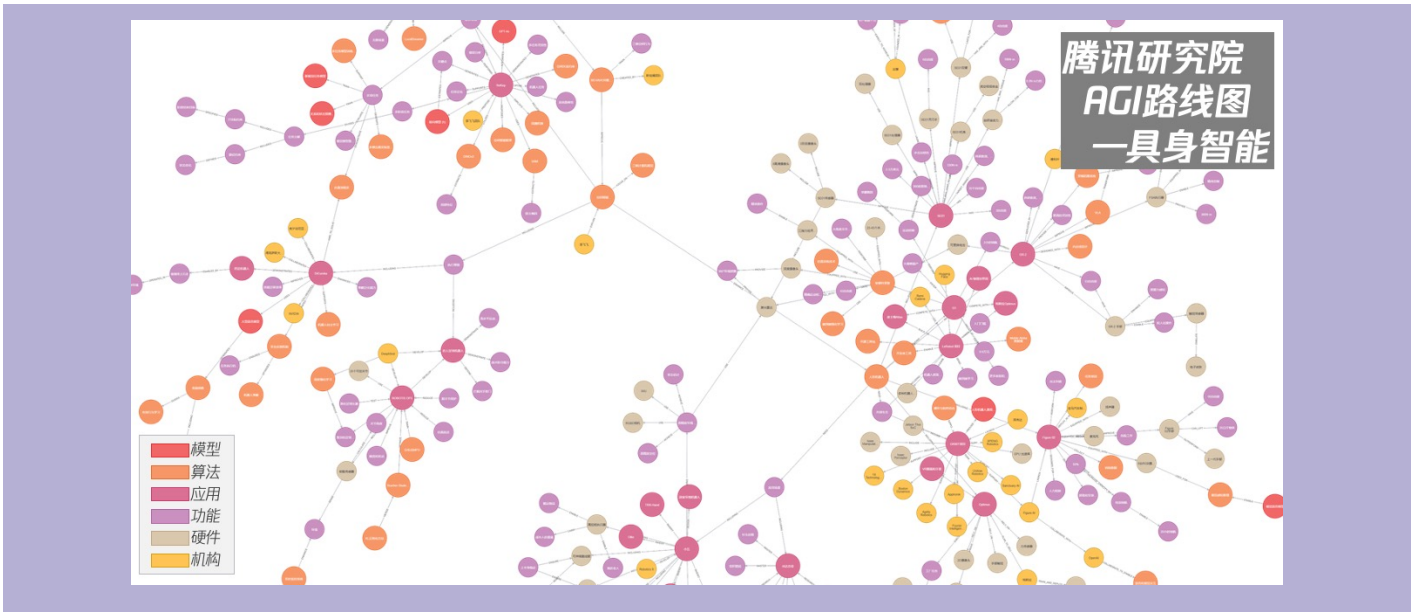
2026年记忆领域最有意思的结构性变化是三条工程路线的分化。目前仍处于同步发展的阶段，还没有形成技术成熟度的梯级，但我们可以试着理解其中的设计哲学与分歧。

路线A，选择性事实提取（Selective Fact Extraction）。AI 记忆项目 Mem0 路线。从对话中自动提取离散事实，去重、更新、按需检索。信任的是系统的提取管线。路线B，文档化自主管理（Document-based Self-Management）。Anthropic 路线。记忆不是离散事实，是 Agent 自己维护的文档集合。自己决定写什么、怎么组织、何时更新。信任的是 Agent 的组织能力。多个小而聚焦的文件，优于少量大文件。路线C，结构化知识图谱（Structured Knowledge Graph）。Neo4j/Zep 路线。记忆是实体+关系+时间的图谱。最大的创新是推理记忆，不只记住事实，还记录怎么解决问题的完整推理痕迹。有意思的是，三条路线大致对应认知科学中三种不同的记忆类型：Mem0 对应语义记忆（我知道什么），Anthropic 对应情景记忆（发生了什么、怎么处理的），Neo4j 对应关系记忆（什么跟什么有关）。这个对应也许并不是严格的科学映射，但它大致的反映了不同项目的设计哲学以及结构性分化的客观事实，不同场景天然需要不同类型的记忆，目前还没有一条路线通吃。

幸运的实际使用中三条路线并非互斥。我们正在使用的记忆系统就是融合的，即使是采用同一套底层数据不变，调用时可以采取不同机制。逐层检索类似 Mem0，文档维护类似Anthropic，Wiki网络推理类似Neo4j。可以根据不同场景混合使用。

把记忆连起来

虽然说，记忆也是文件，比如一个简单的md文件，但不同的形态，所产生的价值差异巨大。比如说，以我们常见的几种形态而言，三个递进的层次。记下来是记忆（Memory），写下来是技能（Skill），连起来是知识工程（Knowledge Engineering）。为什么“连起来”值得单独说。因为哪怕是目前各种龙虾的核心，单个Skill 最终会随着模型能力提升变得不那么重要，模型自己会了。错误记录和交互日志的价值也会被更好的记忆系统吸收。但知识网络，特别是很多特定领域知识，其错误识别的模式、概念之间的因果关系、领域特定的判断框架，大模型学不会。因为它们不是通用的。与人的认知过程类似，在提升判断力的过程中，有必要刻意的保留一些摩擦，来加强概念之间的链接



图：腾讯研究院AGI路线图系列专题研究三

图片来源：https://mp.weixin.qq.com/s/a2VUWB5IIIHSMNhM7ZY4Bw↩

这些知识工程的过程是人生经验的记录，最终会形成每个人不可复制的竞争壁垒。以前个人知识，一般还达不到知识图谱的规模门槛。借助 AI，每个人都可以把知识积累上升到需要结构化管理的程度。我们自己就是案例，研究院不少同学，通过日常前沿论文、产品热点、专家访谈等，多套知识编译系统，成千上万个节点的图谱，形成我们独特的研究脉络。当知识积累到一定规模时，没有结构化的网络，记忆就只是碎片。连起来之后，它变成可以推理、可以发现、可以迁移的东西，继而产生新的洞见。2024年我们的《AGI之路专题分析》专栏，就是采取这样的研究方式来推进的。关于知识工程，我们后面会通过另外一个关键词来展开讨论。

从这里开始

在项目根目录加一个规则文件，作为记忆的起点。不超过200行。声明核心约定、常见错误、偏好设置。比大多数人正在做的好一个数量级。每周看一眼文件大小。超过200行，该整理了。整理的方法可以借助AI，比如，把这篇文章发给它，让它根据你遇到的问题，推荐相应的工程项目。对于普通人来说，不需要一步到位上最复杂的系统。每一步都应该是“现有方案解决不了的问题”驱动的。边用边改。

参考文献：

1. Mem0, "State of AI Agent Memory 2026", 2026-04-01. https://mem0.ai/blog/state-of-ai-agent-memory-2026 ↩ ↩
2. Towards Data Science, "A Practical Guide to Memory for Autonomous LLM Agents", 2026-04-17. https://towardsdatascience.com/a-practical-guide-to-memory-for-autonomous-llm-agents/ ↩
3. FadeMem, "Biologically-Inspired Forgetting for LLM Agents", arXiv, 2026-02-06. https://arxiv.org/html/2601.18642v2 ↩ ↩ ↩
4. Mem0, "The Modal Model of Memory: What AI Agents Can Learn from Cognitive Science", 2026-04-04. https://mem0.ai/blog/the-modal-model-of-memory-what-ai-agents-can-learn-from-cognitive-science ↩
5. Plastic Labs, "Honcho — Dialectic Memory Library". https://github.com/plastic-labs/honcho ↩
6. Anthropic, "Using Agent Memory", 2026-04. https://platform.claude.com/docs/en/managed-agents/memory ↩
7. Neo4j, "Agent Memory", 2026-04. https://neo4j.com/labs/agent-memory/ ↩
8. Mündler et al. (ETH Zurich), "Are Repository-Level Context Files Helpful?", arXiv, 2026-02-12. https://arxiv.org/html/2602.11988v1 ↩

关键词三

技能：最好用的，是自己持续维护的skills。





曹士圮

腾讯研究院AI原生联合负责人

> 我们最近又删掉了一批Skill。不是写得不好，是时间一长反而成了负担。一个成熟的Skill会越来越短。最终留下的就三样：精准描述、领域知识、踩坑记录。

我们最近又删掉了一批 Skill。不是因为哪个写得不好。这些不断累积的 Skill 就像缓存，曾经加速了一切，但时间一长反而成了负担，占上下文窗口，偶尔和新能力打架。得重新整理，系统才能跑得更顺，也才装得下新的技能。

第三条路

让 Agent 具备专业能力，有三种路径。微调。把知识训练进权重。贵，脆弱，还不透明，你不知道它学到了什么，也不知道它什么时候会忘； 检索增强生成（RAG）。把知识放在外部文档里，用时检索。信息是找回来了，但 RAG 只告诉 Agent “知道什么”，不告诉它怎么做； Skill 是第三条路。不改权重，不靠搜索，而是直接把“遇到这种情况应该怎么做”写成可复用的指令模块，运行时按需加载。换个说法：RAG 是图书馆，帮你找事实；Skill 是手，帮你做事情。Work3 的 Matteo Cellini 管这叫“程序性知识的数字表达”。政策变成决策规则，流程变成工作流脚本，指南变成 Agent 可遵循的指令集。从“知道什么”到“知道怎么做”，中间差一个 Skill。更深一层来看，认知科学家 Dreyfus 兄弟在上世纪 80 年代就区分过“知道什么”和“知道怎么做“。真正的专业能力存在于后者，内隐的，不容易用语言表达的。Skill 的工程意义就在这里：把专家级别的操作经验编码成一份文档，让一个能力还在胜任级的 Agent 也能按专家的路径执行。

上下文是稀缺资源

skills的广泛应用，其实，还考虑到一个工程问题：上下文永远是稀缺资源。最开始标准 MCP 配置消耗约 72% 的上下文窗口，3 个 server 在 200K 模型上占掉 143K tokens。工具越多，Agent 的工具选择准确率反而越低，从 43% 降到 14%。更多能力反而

更差表现。不是开发者的 bug，是架构的硬约束。Skill 用渐进式加载（Progressive Disclosure）解决这个问题。三层按需展开：第一层只暴露约 100 tokens 的元数据，名称和描述，始终在线；第二层在模型判断匹配后才加载 SKILL.md 全文；第三层按需访问脚本和参考资料。关键不是“有没有这些知识”，是“什么时候让模型看到”。这也解释了为什么删 Skill 反而能提升效果。上下文窗口就那么大，每一个 Skill 占窗口的同时，就在挤压其他 Skill 和用户指令的空间。Skill 管理是一个注意力分配问题。

```
Skill Folder
|
├─ SKILL.md            ← Core file: defines what the skill does, when to
|     • name: "Quarterly Report Generator"
|     • description: "Parses financial data and produces a structured sum
|     • usage: "When asked to prepare or analyze quarterly results."
|
├─ /resources/         ← Documents, templates, or reference data.
|     └─ reporting_template.xlsx
|
├─ /scripts/           ← Optional code or tools the agent can execute.
|     └─ summarize_financials.py
|
└─ /docs/              ← Extended documentation or examples.
      └─ usage_notes.md
```

图：Skills 技能包的文件结构

图片来源：https://wrk3.substack.com/p/agent-skills-new-currency-of-work



图：收集常见错误或成为最有价值的一节

图片来源：https://medium.com/@tort_mario/skills-for-claude-code-the-ultimate-guide-from-an-anthropic-engineer-bcd66faaa2d6

指令漂移：为什么加法会有反作用

Skills虽好，但太多的话一样会造成负担。开源 Skill 生态在爆发。GitHub 上 awesome-openclaw-skills 收录了 5400 多个 Skill，VoltAgent 的目录超过 1000 个，横跨 Claude Code、Codex、Gemini CLI、Cursor 多平台。Vercel 推出 npx skills 安装指引，有人管这叫 Skill 的“npm 时刻”。确实方便，但生态越大，问题越尖锐。大型规则文件导致指令漂移（instruction drift），AI 跟随最响亮、最新的指令，忽略中间的关键防护栏。指令集一旦冲突，模型性能骤降，比没给指令还差。50 个 Skill 以上，光是元数据层就成了上下文管理问题。不是管理不善，是结构性的。新 Skill 干扰旧 Skill 的触发。你什么都没改，只是多装了一个，原来好好工作的那个突然不触发了。Dreyfus 模型提供了一个理解角度。新手靠规则，专家靠直觉，最高水平的表现出现在结构消退的时候，要留有足够的灵光一闪的空间。一个 Skill 系统如果堆满了面面俱到的指令，实际上是把模型锁在“新手模式”里。成熟的 Skill 越来越短，不是退步，是在向专家级靠拢。

数百个生产级 Skill 之后

没人因为删掉一个 Skill 而获得奖励，但每个人都因为创建一个新 Skill 而获得认可。激励机制不对，减法永远不会发生。

那么，除了直接删skills外，我们应该如何从一开始就学会如何精简一个skill？以及，如何将下载到的各类skills及其精华思想，提炼成自己的一个skill？Anthropic 内部跑着数百个生产级 Skill，工程师总结的经验比理论框架直接，有几个关键要点。描述（Description）要写对，它不是摘要，是触发条件集。是写给模型看的，不是给人看的。模糊了不触发，太宽了误触发。不要写显而易见的东西。Agent 自己能做到的不需要你教。聚焦在那些能推动它超越默认思路的信息上。踩坑记录（Gotchas）是最有价值的部分。不是事先预想的，是基于真实失败积累的。每一条对应一次 Agent 犯过的错。这些才是 Skill 里真正不可替代的内容。其余的，信任模型自己搞定。一个成熟的 Skill 会越来越短。因为模型能力在涨，很多旧的信息留在Skill里反而可能降低性能，该合并的合并，该清理的清理。半年前需要 20 行指令才能可靠执行的任务，现在裸跑就行。最后留下来的老三样：精准的描述、不可替代的领域知识、从失败中学到的教训。而且，这些内容应该是在你自己不断的实践中去动态维护，每个人都有不同的业务场景，即使下载的是同一个Skills，可以在使用的过程中不断的精进，最终远远超越最初的版本。

从这里开始

试着自己创建skill，最初只包含一个 SKILL.md。不超过 50 行。直接放进生产环境用。收集执行情况、错误、失败模式，然后完善。每周看一眼。某些指令模型已经不需要了，删掉。某个边界情况反复出现，加一条踩坑记录。Skill 变短了而表现没下下降，这就是在做减法，做优化。

参考文献：

1. Matteo Cellini (Work3), "Agent Skills as New Currency of Work", 2025-10-22. https://wrk3.substack.com/p/agent-skills-new-currency-of-work ↩ ↩ ↩
2. Muhammad Shafat, "Stop Engineering Prompts, Start Engineering Context", 2026-03-19. https://medium.com/@muhammad.shafat/stop-engineering-prompts-start-engineering-context-a-guide-to-the-agent-skills-standard-bc8e2056f40a ↩ ↩ ↩ ↩ ↩ ↩ ↩ ↩
3. Leading Sapiens, "The Dreyfus Model of Skill Acquisition — A Deep Dive", 2026-01-18. https://www.leadingsapiens.com/dreyfus-model/ ↩ ↩ ↩
4. Milvus, "Is MCP Dead? CLI and Skills for AI Agents", 2026-04-01. https://milvus.io/blog/is-mcp-dead-cli-and-skills-for-ai-agents.md ↩ ↩
5. Mario (Anthropic Engineer), "Skills for Claude Code: The Ultimate Guide", 2026-04-02. https://medium.com/@tort_mario/skills-for-claude-code-the-ultimate-guide-from-an-anthropic-engineer-bcd66faaa2d6 ↩

关键词四

评估：可靠的评估与反馈是方向盘，是质量的前提。





余一

腾讯研究院AI原生联合负责人

> 你刚写完一篇文章马上校对，大概率漏掉很多错误。看到的是你想写的，不是你写的。AI的自评比人更严重。别等它学会自省，给它一个独立的裁判。

一个开发者用同一个模型生成了107个训练样本。Agent 自评：100%通过率，每个样本满分。然后他拿完全相同的模型，开一个全新的上下文独立审查。5.5/10。5个严重缺陷。同一个模型，同一个任务。唯一的区别：换了一个上下文。Anthropic 工程团队给出一个经验，工程化一个严格的评估者，比教一个生成者自省容易得多。别指望 AI 学会自我反省，给它一个独立的评判系统。

技术根因

上下文污染（Context Contamination）。Agent 在同一个推理上下文中既生成又评估，自动补全机制会复用相同的推理路径。类比一下：你刚写完一篇文章，马上校对，大概率漏掉很多错误。并非能力不够，是大脑还在写作模式，看到的是你想写的，不是你写的。在这方面，LLM 比人严重得多。生成就是自动补全，评估时会复现同样的推理错误。好消息是，修复比想象中简单。不需要换更强的模型，只需要换一个新上下文。简单任务这就够了。复杂任务上，评估的设计需要再多花点心思。

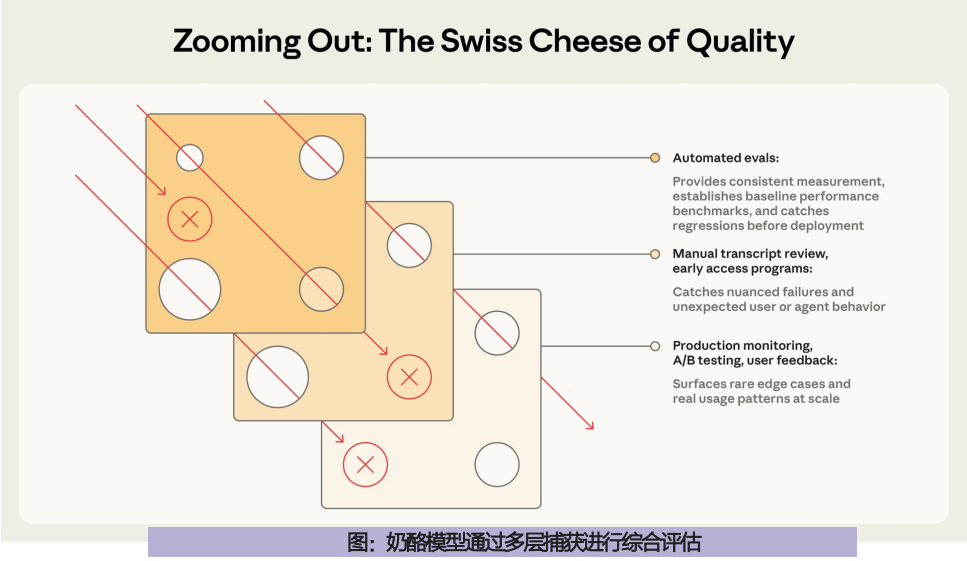
分离生成者和评判者

Anthropic 工程负责人认为其中有一个一个关键：把做工作的 Agent 和判断工作的 Agent 分开，这是最有效的一招。受生成对抗网络（GAN）对抗反馈的启发，工业界形成了一套叫 PGE 的三角色架构，规划者（Planner）负责把任务分解为结构化规格说明，生成者（Generator）按规格产出内容，评估者（Evaluator）对照规格审查产出，输出结构化批评。Anthropic 在实战中用了这套思路。生成者和评估者各自跑在独立的上下文窗口里，靠文件系统通信。一个写文件，另一个读文件并回应。评估者配了浏览器自动化测试工具，能独立导航到实时页面、点按钮、截图、检查交互，然后把评分和 Bug 报告写回文件。每轮跑5到15次迭代，有时长达4小时。评估不是“看一眼输出”，而是像真实用户一样测，直接看结果。

有意思的是，随着模型能力提升，Anthropic 已经移除了早期工程脚手架中的上下文重置和任务分解机制，更强的模型不再需要这些拐杖。但在评估一节，生成者和评估者的分离始终是保留的。这说明自评偏差不是模型能力够不够的暂时问题，而是在根源上需要长期补全的一个有效机制。

评估本身需要被评估

分离解决了“谁来评”。但评估者自己靠谱吗？拿 AI 当裁判（LLM-as-Judge）来评 AI。在受控实验中，GPT-4 和人类评估者的一致率超过80%，跟人和人之间差不多。听起来不错。但换到生产环境常见的扰动条件下，前沿模型错误率超过50%。四种系统性偏见已被证实。可能出现的情况：位置偏见：两个回答质量一样，谁排在前面谁得高分。冗长偏见：内容相同，长的那个分更高，AI 把“写得多”当成了“写得好”的代理指标。自偏好：模型给自己生成的内容打高分，尤其不愿惩罚自己犯的错。家族偏见：同一家公司出的模型有可能互相打高分，比如用 GPT 评 GPT、用 Claude 评 Claude，偏差可能比单个模型自偏好还大。兰德公司2026年的开源压力测试覆盖了安全、说服力等多个基准，最终得到了一个结论：没有任何一个 AI 裁判能跨基准保持一致可靠。那用基准测试来评呢？基准也可以作弊。包括我们现在事实上也越来越难相信各类基准评测。那用人来评估呢？AI生成的速度太快了，人类评估会成为任务瓶颈。另外，人也会出错。Anthropic提出了一个三层信任梯度。确定性检查是地基，最可信但覆盖面最窄。AI 审查是楼层，能看到语义问题但有偏见。人类判断是屋顶，覆盖面最广但最慢最贵。单独哪一层都不够。组合起来，一层的盲区被另一层覆盖，类似瑞士奶酪模型（Swiss Cheese Model）的逻辑。



图：奶酪模型通过多层捕获进行综合评估

图片来源：https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents

那么在我们的日常工作中，怎样让自己这一层产生乘数效应？

第一，AI 产出不满意时，要克制，别改产出，改产生产出的系统，规则有没有遗漏、约束有没有缺失、上下文有没有给错。人在顶层评估的频次低，但每一次改的是系统而非产出物，影响是乘数级的。

第二，检查机制发现错误时别只说“错了”，反馈要直接包含修复方法，或者跟AI一起讨论给出修复方法并记录。约束是事前预防，评估是事后检测，正向提示注入（Positive Prompt Injection）是桥梁，把检测结果转化为下一轮预防。

具体应该怎么做

关于评估的搭建，听起来是一些很复杂、让人头大的事儿。但事实上，可以从简单开始。Anthropic 基于内部数百个生产级 Agent 的经验，提炼了一套路线图；Red Hat 和 Amazon 在各自的实践中也独立走到了类似的结论。几个要点：

一、先跑起来。20到50个评估任务就够起步。Red Hat 更直接：把上周的 bug、用户反馈过的问题直接转成测试用例，这就是第一版。别等到题库完善了才开始。

二、正反都要考。只考“该做的事有没有做”，结果就是训练出一个什么都做的 Agent，该出手时出手，不该出手时也出手。还得考“不该做的事能不能忍住”。单向考试导致单向优化。Red Hat 在实践中专门维护了一组"失败对话"，每次改规则都拿出来验一遍，确保这些错误还能被抓住。

三、只看结果，不管过程。别规定 Agent 必须按特定步骤执行，不然会惩罚创造性解法。只看最终结果达没达标。不过说实话，“怎么判断最终结果是对的”本身就很难，Martin Fowler 团队把它叫“房间里的大象”，目前没有特别成熟方案。

四、先守住底线，再逐步加码。Agent 上周能做对的事，这周别别做错，这是底线。稳住之后加难度，测它现在还做不到、但将来应该做到的事。做到了就纳入底线。不需要一开始就知道满分长什么样。Amazon 在生产环境也采取了这一思路：持续监控，性能一掉就触发审查。

这一系列的做法，在开发领域已经显现出极大的效率提升：没有评估体系的团队，新模型发布后需要数周人工验证。有评估体系的团队，几天就能完成升级。在非编程环境，比如知识工作等目前并没有特别成熟的评估方案，但是相关的评估思路可以借鉴。哪怕是把错误记录下来，反馈给AI。

从这里开始

在日常工作中，给 Agent 加一个独立审查者。同一模型、新上下文就够。不需要搭复杂系统。把上周修的3个 bug 转化为评估任务。这就是回归测试套件的 v0.1。不要追求完美评估。追求多层不完美评估的组合，确定性检查（硬性条件）兜底，推理 AI 审查扩面，人类判断改系统。

参考文献：

1. Victor, "Why Your AI Agent Can't Check Its Own Work (and How to Fix It)", 2026-03-27. https://dev.to/victorvvedtion/why-your-ai-agent-cant-check-its-own-work-and-how-to-fix-it-3kjc ↩
2. MindStudio, "Planner-Generator-Evaluator Pattern: GAN-Inspired AI Coding Architecture", 2026-04-01. https://www.mindstudio.ai/blog/planner-generator-evaluator-pattern-gan-inspired-ai-coding/ ↩ ↩
3. Birgitta Boeckeler (ThoughtWorks), "Harness Engineering for Coding Agent Users", 2026-04. https://martinfowler.com/articles/harness-engineering.html ↩ ↩ ↩
4. InfoQ, "Anthropic Designs Three-Agent Harness", 2026-04. https://www.infoq.com/news/2026/04/anthropic-three-agent-harness-ai/ ↩
5. Adaline, "LLM-As-A-Judge: Reliability, Bias, And What The Research Says", 2026-04-08. https://www.adaline.ai/blog/llm-as-a-judge-reliability-bias ↩ ↩
6. Anthropic, "Demystifying Evals for AI Agents", 2026. https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents ↩ ↩ ↩ ↩
7. Red Hat Developer, "Eval-Driven Development: Build and Evaluate Reliable AI Agents", 2026-03. https://developers.redhat.com/articles/2026/03/23/eval-driven-development-build-evaluate-ai-agents ↩ ↩ ↩
8. AWS, "Evaluating AI Agents: Real-World Lessons from Building Agentic Systems at Amazon", 2026-02. https://aws.amazon.com/blogs/machine-learning/evaluating-ai-agents-real-world-lessons-from-building-agentic-systems-at-amazon/ ↩ ↩

关键词五

上下文管理：停止把你的人生故事放进去。





王鹏

腾讯研究院资深专家

> 信息太少让AI发散，信息太多也让AI犯蠢。它的问题不是不听话，恰恰是太听话了，把所有指令都当回事，包括不必要的那些。

信息太少让 AI 太过发散，信息太多也会让 AI 犯蠢。AI 问题不是不听话，恰恰是太听话。它把所有指令都当回事，包括不必要的。

n^2 的代价

Transformer 架构有个特性。每个 token 要跟所有其他 token 建立注意力关系。10个 token 是100对，100个是10,000对，10,000个是一亿对。上下文膨胀的成本不是线性的。这是所有“少即是多”（less is more）现象的数学根源。Anthropic 的定义值得逐字品味：好的上下文工程（Context Engineering）意味着找到最小的高信号 token 集合，最大化某个期望结果的概率。注意措辞，划重点：最小集合（Smallest possible set）。Karpathy 的类比：LLM 是 CPU，上下文窗口是内存（RAM），外部存储是磁盘。你不会把整个硬盘塞进内存。哪怕装得下，CPU 也处理不过来。

从第一个 Token 就在退化

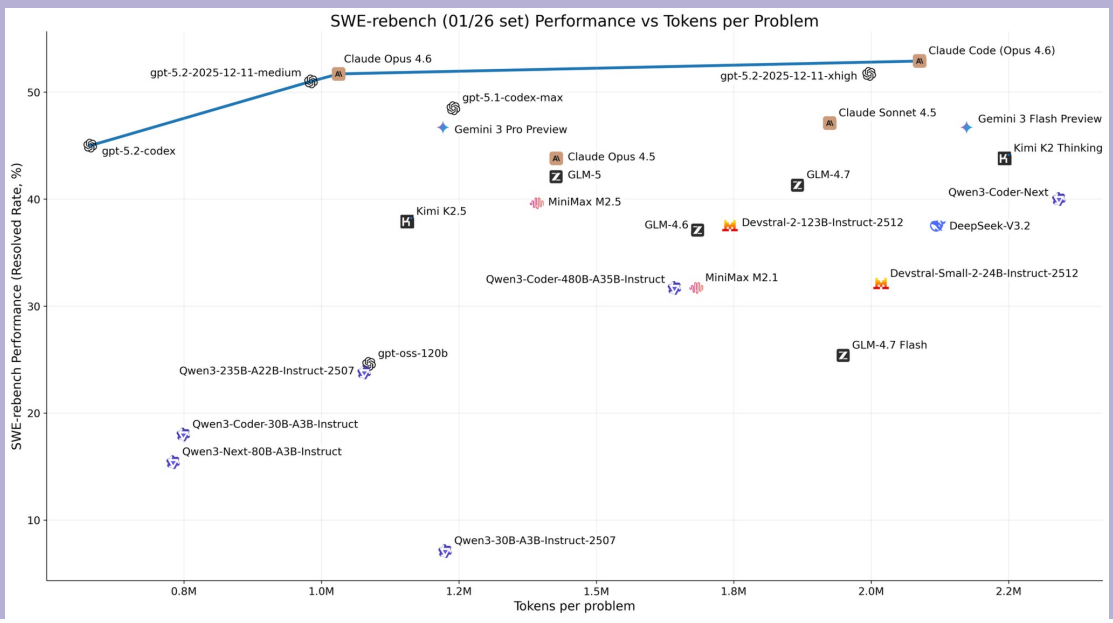
向量数据库公司 Chroma 拿18个前沿模型做了系统化长上下文评测。固定任务难度，只变输入长度。所有18个模型，在所有实验中，性能都随输入长度退化。没有例外。不用到了窗口极限才掉，从输出的第一个 token 就在掉。其中，更反直觉的发现。结构化的内容反而让模型更容易犯错。打乱句子顺序后所有模型表现更好。当内容有逻辑连贯性时，模型被逻辑流吸引，把注意力分散到理解结构上，而不是集中在找答案。结构化的指令是好的。结构化的填充内容是坏的。比如让 AI 写一封邮件，给它一句“用正式商务语气”可能就够了；但如果你塞进去一份两页的格式规范，字号、行距、段间距、称谓规则、落款格式，模型反而会把注意力花在对齐格式上，忽略了邮件本身要说什么。给多少信息没那么重要，怎么呈现才是关键。并不是越多越好，普通人在日常使用过程中，几乎没有注意到这一点，几乎都是全部复制粘贴。

百万 Token 天花板

2026年窗口军备竞赛确实在继续，不少前沿模型的窗口已经超过100万。但上下文并不是万能的，更大的窗口并不意味着更好的表现。SWE-rebench 的维护者报告：模型在约100万 token 处撞上了明确的性能天花板。RULER 基准测试的数据更直观：大多数模型从4K到128K，性能下降15到60分不等。你在不同上下文长度下用的，其实不是同一个模型。日常工作中的体感也一样：即便工具提供20万 token 上下文窗口，真正高效利用的部分通常不超过2-3万。正确路径不是更大窗口，而是更精准的输入。MCP提出之初，工具是直接全量加载的，随着MCP安装数据的增加，上下文占用问题严重，每次打开对话上下文已经被大幅占用，导致性能下降。随后，Claude Code 2.1.7 引入了 MCP 工具懒加载（Tool Search），不再在会话开始时加载所有工具定义，而是按需搜索和加载。效果：工具定义的 token 消耗从约7.7万降到约8700，减少85%；模型准确率也跟着涨了。

四种坏法

毒化（Poisoning）。一个幻觉或错误进入上下文后，被反复引用，错误滚雪球放大。分心（Distraction）。上下文过长后，模型开始重复历史模式而不是执行当前指令。混淆（Confusion）。工具和选项太多时性能下降。不是长度问题，是复杂度问题。冲突（Conflict）。累积的信息包含矛盾时，推理直接脱轨。AI一旦在对话中走错方向，就回不来了。上下文工程不只是精简的问题，还有治理的问题。



图：基础模型或在百万tokens遇到明显的性能上限

图片来源：https://x.com/Shevan05/status/2022364902090764308

清空比压缩更有效

研究中最意外的发现之一。来自长任务场景的经验。直觉上，原地压缩应该优于清空重启，因为我们往往希望模型能记住之前讨论的话题，从而引导后续的任务完成。但实践数据反复显示：重启在很多场景下更有效。认知科学提供了解释。持续注意力下降不是因为资源用完了，是控制资源分配的能力退化了，还有燃料，但导航系统开始出错，把燃料送到了错误的地方。举一个更贴近生活例子，也是数十年验证的最有效对策。当我们工作时间已经非常长的时候，最有效的方法，是休息。不是更努力集中注意力，不是把之前的工作，总结下来，而是可以换一个工作内容，换一个上下文，中断再恢复。上下文重置（Context Reset）有效不是偶然的工程巧合。它对应了注意力系统的一个基本特性。不要害怕清空对话重新开始。只要进度持久化在了文件里，清空往往比死撑着压缩更高效。但这里面有一个关键前提是持久化，在“记忆”关键词中，我们讨论的写入-管理-读取（Write-Manage-Read）框架。没有持久化的重置是失忆。有持久化的重置是睡一觉醒来精神抖擞。

一开始就不加载

关于上下文，大多数人还在纠结怎么压缩。真正的杠杆在于一开始就不加载。渐进式披露（Progressive Disclosure）。从用户体验设计借来的概念。不是一次性给所有信息，按需分层揭示。先给索引，需要什么再读什么。轻量预筛步骤先评估需要什么。研究/规划/执行/审查，每阶段不同上下文。这样的思路现在被普遍使用，除了上述的MCP 工具懒加载外；还有，记忆先读索引，不读全量；Skill 描述"做什么"，不包含做它所需的材料。距离核心任务越远的信息，加载时机应该越晚。始终加载的内容必须极度精简。按需加载的内容可以详尽一些。

从这里开始

大模型读取的第一轮文件（不管是记忆还是skill，或者是其它），只需要三样东西。一句话描述、关键约定、非标准流程，其余一切在别处。前沿 LLM 能较好的一致遵循150-200条指令，以此作为基础指令预算。如果更多 token 没有带来更好的结果，需要精简不是扩展。最大的坑是放任过时信息留在上下文，它们不是无害的背景信息，它们在主动毒化 Agent。定期审查，删掉比添加更重要。

参考文献：

1. Mündler et al. (ETH Zurich), "Evaluating AGENTS.md: Are Repository-Level Context Files Helpful for Coding Agents?", https://arxiv.org/html/2602.11988v1 ↩
2. Anthropic, "Effective Context Engineering for AI Agents", https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents ↩
3. Galileo AI, "Deep Dive into Context Engineering for Agents", https://galileo.ai/blog/context-engineering-for-agents ↩
4. Chroma, "Context Rot Research", https://www.trychroma.com/research/context-rot ↩ ↩
5. MorphLLM, "1M Token Wall", https://www.morphllm.com/context-engineering ↩ ↩
6. Software Thug, "Claude Code MCP Tool Search: How Lazy Loading Cut Token Usage by 85%", https://www.softwarethug.com/posts/claude-code-mcp-tool-search-lazy-loading/ ↩
7. Atlan, "Working Memory in LLMs", https://atlan.com/know/working-memory-llms/ ↩ ↩
8. MindStudio, "Progressive Disclosure in AI Agents", https://www.mindstudio.ai/blog/progressive-disclosure-ai-agents-context-management/ ↩ ↩
9. Matt Pocock (AI Hero), "A Complete Guide To AGENTS.md", https://www.aihero.dev/a-complete-guide-to-agents-md ↩ ↩

关键词六

工作流：人类工程师始终是这场演出的导演。





刘莫闲

腾讯研究院高级研究员

> 大多数人在用AI加速单个任务，整体工作流没有动。有工具不等于有工作流。真正卡住大家的，是重新审视整条流程的那一步。

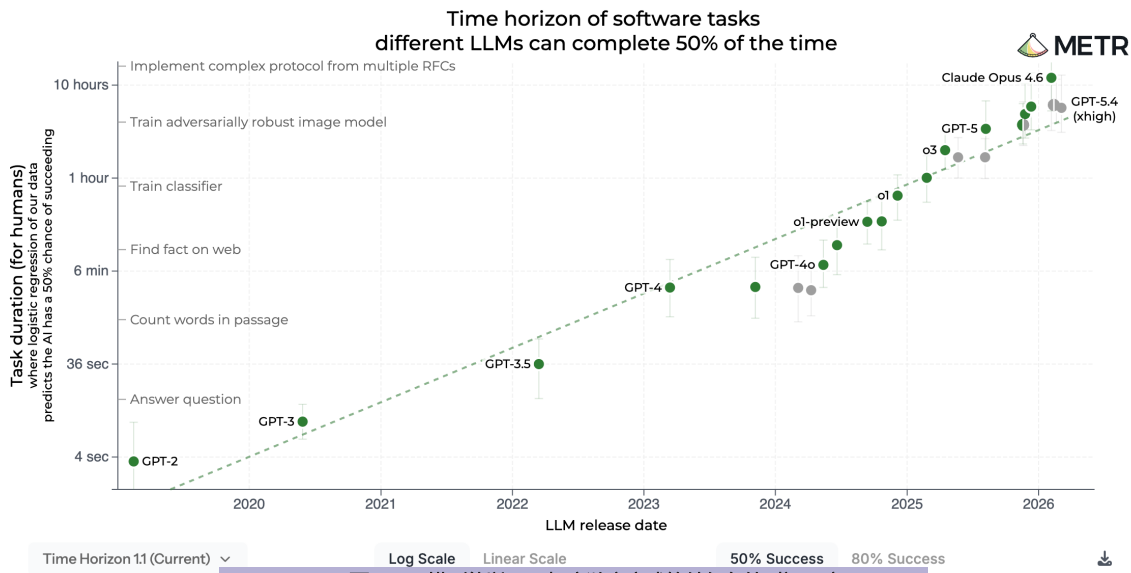
84%的团队在用 AI。只有21%认为自己的工作流为高效。甚至有人认为，使用 AI 反而降低效率。差距不是技术问题。工具到位了，模型够强了。大多数人依然在用 AI 做一件事，加速单个任务。整体工作流没有改变，提效必须是有限的。比如，一个10步工作流的简单数学。每步各占10%，AI 让其中一步速度翻倍，整体从100%降到95%，翻倍了单项任务效率，整体只得到5%的系统改善。拥有 AI 工具不等于拥有 AI 工作流。

认知搜索空间的差距

INSEAD 的 Hyunjin Kim 和哈佛商学院的 Rembrand Koning 做了一个515家全球初创公司的随机对照实验（RCT）。实验周期10周。两组拥有完全相同的资源：API 额度、前沿模型、每周3小时技术培训（MIT 和哈佛专家授课，内容涵盖 RAG、Agent 工作流、上下文工程）。唯一差异在每周60分钟的工作坊：对照组学的是标准创业方法论，客户画像、假设验证、精益迭代等等；实验组看到的是 AI 原生企业如何围绕 AI 重组整个生产流程的前后对比案例。结果：实验组发现的 AI 用例比对照组多44%，获得付费客户的概率高出18%，总收入是对照组的1.9倍。资本需求降低了39.5%，同样的人，不增加人手，用更少的钱做到了更多。而且收入增长不是均匀分布的，在第90和95百分位出现了巨大跃升，工作流重设计让最好的创意突破天花板。没有技术差距，没有资源差距。研究者叫它映射问题（Mapping Problem），发现 AI 在生产流程中具体在哪里、如何创造价值，本质上是一个搜索问题。对照组失败是因为局部搜索，在熟悉的地方找应用。这与我们当下不少企业用AI的方式是类似的。实验组被迫扩大了认知搜索空间。实验最大的价值不是最终的数据，不是那个1.9倍。是它用因果证据证明了阻碍 AI 盈利的不是工具、不是技能、不是预算，而是重新看待整个长任务工作流、端到端AI提效的的能力。

时间维度

工作流设计首先要克服长任务的限制。AI 安全评估机构 METR 发现，AI 能独立完成的任务时长过去六年一直在指数增长，大概7个月翻一倍。当前最强模型在4分钟以内的任务上成功率接近100%，但超过4小时就掉到10%以下。能力在指数增长，但长任务依然是一堵墙。7个月翻一倍说明这堵墙在快速变矮，但现



图：不同模型能以50%概率独立完成软件任务的时间尺度

图片来源：https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/

它还很高。长任务的核心不仅仅是让 AI 跑更久，还要设计完美的交班协议。三个完全独立的项目，在互不知情的情况下，收敛到了同一个解法：用文件管理来实现长任务的管理。Manus 用的是 task_plan.md 加 notes.md。OpenClaw 用是 MEMORY.md 做长期记忆。Claude Code 用的是 CLAUDE.md 层级加 Skills 系统。这三个项目收敛到文件方案，方案虽然简单，但这些方案已经较好的精确匹配了长任务的需求。持久，进程崩了数据不丢。透明，任何编辑器能看能改。版本可控，出问题可以回滚。全局上下文，不像片段检索，一个写好的文件给你整张地图。可移植，不锁定任何供应商。基于简单的文件管理系统方案，足够我们结合业务场景搭建出可以持续运行数小时的工作流任务，关键是融入自己对流程的理解。

控制权

工作流设计有一个被技术社区忽视的维度。控制权怎么分配。管理学几十年来的核心问题之一就是委托（Delegation）。多少决策权交给别人，自己保留多少。人机协作中也完全适用，其中，可恢复执行（Recoverable Execution）是工程化的关键理解。仔细想，人工审批暂停和系统崩溃恢复其实是同一件事，都是保存状态、暂停、恢复状态、继续。一套基础设施同时解决了“人要审批”和“机器会挂”两个问题。通过上述文件系统可以实现这一点，我们可以结合业务流程理解，来设置自己的干预点，但不会直接导致整个工作流程崩溃。半人马模型（Centaur Model）提供了更深的思考。人和 AI 不是各做各的，是合并。哈佛和梅奥诊所（Mayo Clinic）的实验中，半人马组合优于纯算法，纯算法优于纯人类专家。人类直觉不如算法，但能提升算法。人类在工作流中的价值不仅是审批者。是通过共生学习持续优化 AI 决策质量的合并伙伴。工作流中需要保持人类“通过做来思考”的空间。阅读提升认知、执行输出、批注决策，如此循环。我们日常研究工作也是类似的，需要通过一系列深度协同的工作来提升自己的认知，人为的制造一些认知提升必要的摩擦。

飞轮

好的工作流不是静态设计，是不断积累的飞轮。如果一个工作流很长时间没有得到反馈更新，那么它大概不是一个好的工作流。每次执行产生的反馈：哪些步骤 Agent 做得好、哪些需要人类介入、哪些约束需要调整，应该系统化捕获并回注到工作流设计中。鼓励积累型而非消耗型使用。比如，纯用 AI 生成报告然后丢掉是消耗型。过程中沉淀了新 Skill、新评估标准、新约束规则是积累型。后者每次使用都让系统变好。值得关注的是，微软研究院（Microsoft Research）指出人机协作中的一个代价。从“通过做来思考”到“从输出中选”的范式转变中，如果人类习惯了只选不做，判断力依然可能退化。当前工作流、方法论假设稳定的人类判断力供给。今天设计的工作流，最好也为判断力传承留一个接口，避免去技能化。这个我们会在后续的关键词中讨论到。

局部搜索

回到 INSEAD 实验最深层的发现。对照组失败不是因为技术不行。完全相同的资源。失败是因为局部搜索（Local Search）。在熟悉的地方找应用。设计师用 AI 生成占位文字，开发者用 AI 调试脚本，客服用 AI 写回复。每个人优化自己的节点。局部优化保留了瓶颈，某环节快了300%，下游积压只会更长。可用性之父 Nielsen 提出：自动化必须是端到端的，或者至少要覆盖关键路径上的所有环节。Nielsen 给出了四个动作判断你是不是在做真正的工作流重设计：移除交接（两个环节之间不再需要人搬运数据）、并行化变体（同时跑多个方案而非串行）、把人类移到异常处理（常规路径全自动，人只处理出错的）、增加评估循环（AI 加速了生成，瓶颈转移到评估）。如果一个 AI 改造项目不触及其中任何一条，大概率只是局部优化，卡点大概率会在别处出现。

从这里开始

拿出最关键的工作流。画出来，找到人类作为胶水的位置，比如，手动传递上下文、数据、状态的时刻。问：AI 能成为这里的桥梁吗，如果可以，怎么实现？请AI帮忙一起突破这一部分。要尽量避免：局部搜索，单任务提效，只在熟悉的地方找 AI 应用。当下使用AI，更好的解法是AI在跨职能长流程的映射，而非单单研究更好的提示词。

参考文献：

1. Telerik, "Workflows in the Age of AI", 2025. https://www.telerik.com/ai-design-development-workflows-report-2025 ↩

2. Jakob Nielsen, "Redesigning Workflows for AI (INSEAD/Harvard RCT)", 2026. https://jakobnielsenphd.substack.com/p/workflow-redesign ↩ ↩ ↩ ↩ ↩ ↩

3. Harvard D^3, "Everyone Has AI. Which Firms Win?", 2026. https://d3.harvard.edu/everyone-has-ai-which-firms-are-going-to-win/ ↩ ↩ ↩ ↩ ↩

4. METR, "Measuring AI Ability to Complete Long Tasks", 2025-03-19. https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/ ↩

5. ImagineX, "AI Agent Memory Management: When Markdown Files Are All You Need", 2026. https://dev.to/imaginex/ai-agent-memory-management-when-markdown-files-are-all-you-need-5ekk ↩

6. Rittika Jindal, "Building Retries in Agents: How to Build AI Agents That Survive Failures", 2026. https://rittikajindal.medium.com/building-retries-in-agents-how-to-build-ai-agents-that-survive-failures-32eedd2623f0 ↩

7. Harvard HDSR, "The Human-Algorithm Centaur", 2026. https://hdsr.mitpress.mit.edu/pub/3rvlzjtw ↩ ↩

8. Microsoft Research, "New Future of Work: AI Is Driving Rapid Change, Uneven Benefits", 2026. https://www.microsoft.com/en-us/research/blog/new-future-of-work-ai-is-driving-rapid-change-uneven-benefits/ ↩ ↩

关键词七

多智能体：用好一个再用多个，可以打开更大的世界。





白惠天

腾讯研究院高级研究员

> 多Agent确实更强大，但它的门槛不在模型，在人。绝大多数失败来自规则没说清楚、分工不够细。先把一个Agent用好，再考虑加人。

一棵决策树

多智能体（multi-agent）很好，有着非常广阔的应用场景，但也有复杂的应用边界，不是所有场景都适用。对于普通人，单 Agent 加好的提示词（prompt）加合适工具能搞定吗。能，停。暂时不需要多 Agent。加Agent的理由有四种。上下文放不下了。不同子任务需要不同提示词/工具/模型。有独立子任务可以并行跑。需要故障隔离（Fault Isolation）。四条一条都不中。暂时不需要多 Agent。先用好一个 Agent，工具加够，再考虑加 Agent，这是迈入多智能体协作的基础。

Token 扩展

多 Agent 确实能带来显著收益，但前提是入场时机对。Anthropic 构建 Claude Research 系统后发现，升级模型可能比堆更多 token 更高效。如果还在用上一代模型，先升级。用上最强模型仍然不够，才是多 Agent 真正发挥价值的时候。其在后续的一项专题研究中进一步提醒：许多团队花费数月构建复杂多 Agent 架构，最终发现改进单一 Agent 的提示词就能达到同等效果。不是多 Agent 不好，是基础没打牢就上架构，浪费了它的优势。而且，评估多 Agent 收益是一个复杂的问题。我们看到一些标杆案例的多 Agent 系统之所以有效，主要是因为它们消耗了足够多的 token 来解决问题。比如，OpenAI 网页浏览评测 BrowseComp 中，token 用量单独解释了80%的性能方差。工具调用次数和模型选择加在一起才解释15%。也就是说，多 Agent 的核心价值不是“更聪明”，而是给 AI 更多并行计算容量。每个子 Agent 有自己的上下文窗口，独立探索。本质上是用更多 token 覆盖更大的信息空间。在需要广度的任务上，这正是它的杀手锏。换一个任务，不像搜索这种需要超大信息空间的任务，多 Agent 也不一定比单 Agent 更有效。

四大陷阱

加州大学伯克利分校（UC Berkeley）分析了1642个多 Agent 执行轨迹。41-86.7%在生产中失败。79%的失败来自规范问题和协调失败。并非模型不够强，是人没把规则说清楚。一、过度委托（Over-delegation）。一个配了好工具的单 Agent 能处理绝大多数场景。确保第一个稳定之后，才逐步引入子 Agent。比如，写一个大报告分为若干章节，我们至少需要让 Agent 先把第一章内容打磨好，再推进到其它部分。包括如何搜索、整理数据、研讨、判别、撰写与修改等等。二、规格不足（Under-specification）。给执行者（Worker）的任务描述要像给第一天入职的初级工程师写工单一样详细。Anthropic 早期让主 Agent 给简短指令如“research the semiconductor shortage”。结果多个子 Agent 重复调查同一个方向。分工不够 MECE（麦肯锡等咨询公司的经典分析原则，互不重叠，完全穷尽）。三、协调开销（Coordination Overhead）。一个任务经过分发、执行、合成，可能5-10次 API 调用。单 Agent 只需1-2次。协调成本超过收益，该简化。四、电话游戏效应（Telephone Game Effect）。信息在每次传递中衰减。这个时候，更建议的做法，是让子 Agent 直接写文件系统，绕过主 Agent 转述。但还是不可避免的有一些信息衰减。

微服务思维

怎么决定 Agent的分工，或者某单一 Agent 能处理分给它的任务。一个判断原则，就是是否共享同一个上下文。如果某项任务，所有的执行内容，都需要共享同一个上下文，那么，最好是直接使用主Agent来执行。如果一些任务可以并行处理，也必须确保每个 Agent 使用自己独立的上下文窗口是可以把任务完成的，因为 Agent 之间一般不共享上下文。而且，我们用多



图：众多挑战下多智能体系统仍具广阔的研究价值前景

图片来源：https://www.anthropic.com/engineering/multi-agent-research-system

Agent 就是为了让每个 Agent 聚焦一个切片。如果每个都看全部上下文，只是把过载信息复制 N 份还多付 N 倍 token。Agent 间通信天然有损。必须接受这一点，然后设计系统来管理信息衰减。Worker 返回结构化结果。子 Agent 直接写文件系统。任务描述详细到不依赖上下文就能执行。这里需要澄清一下，根据 LangChain 的数据。在特定任务情况下，子 Agent 模式（Subagents）比上下文累积模式少处理67%的 token。这是好事，前提是这个工作流，这些 子Agent 可以接受信息不对称。

管理学的老酒

多 Agent 系统正在重新发现组织理论研究了几十年的问题。管理幅度（Span of Control）。一个编排者（Orchestrator）管多少执行者。目前，Anthropic 提出的最佳点是3-5个，适用于研究任务。编码任务可能1-2个就够。分工。太细，协调成本爆炸。太粗，多 Agent 没意义。遵循权威梯度（Authority Gradient）

的规律。主 Agent 到子 Agent 的层级关系。太陡，产生谄媚效应（Sycophancy），下属只会迎合上级。太平，协调失败。当然，这些组织理论是启发框架，不是工程规范。人类组织的管理幅度受限于人脑带宽，Agent 的管理带宽遵循不同规律。但是，我们仍然可以把组织理论当检查清单：管理幅度合理吗，分工边界清晰吗，信息不对称可控吗。这是我们组织多 Agent 任务前需要明确的问题。不管是我们自己思考，还是与AI一起讨论。

从这里开始

单 Agent + 工具是起点，90%的场景到这一步就够了。当单 Agent 确实到了瓶颈，从编排者+执行者（Orchestrator + Worker）开始。一个主 Agent 做规划，先把任务聊清楚，再交给2-3个 Worker 做执行。前期复杂任务使用多 Agent 的情况下，先多参与，避免过度委托。团队在还没榨干单 Agent 能力的时候，就急着上多 Agent 架构，有可能更慢。

参考文献：

1. LangChain, "Choosing the Right Multi-Agent Architecture", 2026-01-14. https://blog.langchain.com/choosing-the-right-multi-agent-architecture/ ↩ ↩
2. Anthropic, "How We Built Our Multi-Agent Research System", 2025-06-13. https://www.anthropic.com/engineering/built-multi-agent-research-system ↩ ↩ ↩ ↩ ↩ ↩ ↩
3. Anthropic, "When to Use Multi-Agent Systems (and When Not To)", 2026-01-23. https://claude.com/blog/building-multi-agent-systems-when-and-how-to-use-them ↩
4. Augment Code, "Multi-Agent AI Systems: Why They Fail" (citing MAST/NeurIPS 2025), 2026. https://www.augmentcode.com/guides/why-multi-agent-llm-systems-fail-and-how-to-fix-them ↩ ↩
5. dev.to, "When to Use Multi-Agent Systems (And When Not To)", 2026-03-25. https://dev.to/diven_rastdus_c5af27d68f3/when-to-use-multi-agent-systems-and-when-not-to-5ah1 ↩ ↩ ↩ ↩
6. arXiv 2601.13671, "The Orchestration of Multi-Agent Systems", 2026-01-20. https://arxiv.org/html/2601.13671v1 ↩

关键词八

加法偏见：加法是能力，减法是智慧。





王强

腾讯研究院资深专家

> 三岁小孩从高柱上拿掉一块积木，桥就搭好了。大人的本能是去找一块新的加上。一系列研究发现，AI从人类语言中训练，把这个加法偏见进行了放大。

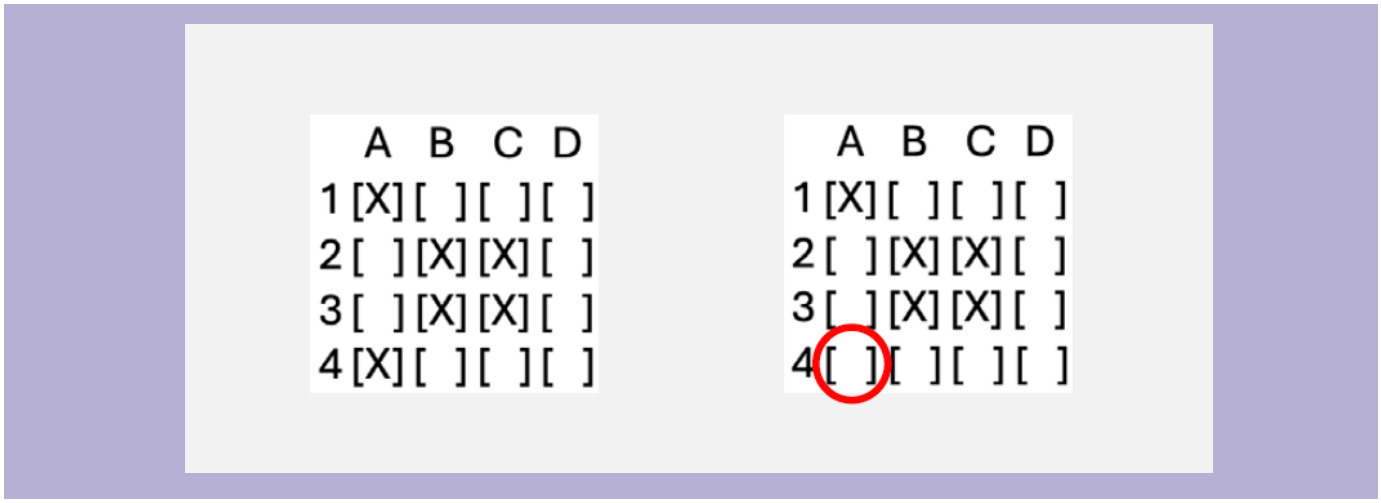
一个故事，弗吉尼亚大学行为科学教授 Leidy Klotz 和他三岁的儿子搭 Lego 桥。两根支柱高度不同。Klotz 本能转身去拿一块积木，加高矮柱。等他转回来，儿子已经从高柱上拿掉了一块。桥搭好了。后来变成了一篇 Nature 论文。通过8个实验证明，当要求改进某个对象（乐高结构、文章、行程路线、网格图案等）时，人们默认倾向于“添加”元素，即使“移除”元素是更简单、更高效的解。对照组：无提示，只有 41% 想到减法；实验组：仅加一句提示，8个单词“removing pieces is free and costs nothing”，减法使用率升至 61%。不是不会减。是想不到。

默认设置

加法偏见（Addition Bias）。人类面对“怎么改变现状”时，系统性倾向于“加点什么”，忽视“减掉什么”。哪怕减法是更优解。生活里到处是加法偏见。身体不舒服，第一反应是加点什么，补维生素、吃药、挂号。但有时候什么都不做，减少消化负担，身体自己就能更好的修复。加法偏见，与一系列的相关理论，包括：沉没成本谬误（Sunk Cost Fallacy）、损失厌恶（Loss Aversion）、浪费厌恶（Waste Aversion）是一伙的。2023年伯明翰大学一项研究揭示了更底层的原因。语言本身偏向加法。英语中 add、more 相关词的使用频率显著高于 subtract、less。但还好，加法偏见不是铁律。是可干预的倾向，8个词就能松动它，我们可以想办法去避免其副作用。

双重放大

开篇提到的是 2021 年发表在 Nature 的实验，主要研究对象是人类，证明人类有加法偏见。五年后，另一项研究追问：AI 呢？2026年，Uhler 团队在 Nature 子刊 Communications Psychology 发表了一项人机对比实验。人类和 GPT-4/GPT-4o 做同一套任务，编辑摘要、调整网格图案。实验通过巧妙设计，让有些题目减法明显更省力：比如摘要任务中，减掉6个词就能达标，加法则需要加18个词。结果：人类加法策略使用率约 60%。GPT-4o 是88%-100%。更关键的差异：面对减法更高效的题目，人类会切换到减法。GPT-4 不仅不切换，加法比例反而更高，效率信号对它产生了反向作用。LLM 从人类语言中训



图：为使图案对称不管是人或AI均倾向于增加元素

图片来源：https://www.nature.com/articles/s44271-026-00403-0

Table 1
Target words used for the frequency analysis; word frequencies from the Corpus of Contemporary American English (grand total over all registers)

Pair number	Word	Type	Frequency
1	add	+	361,246
	subtract	−	1802
2	addition	+	78,032
	subtraction	−	313
3	plus	+	110,178
	minus	−	14,078
4	more	+	1,051,783
	less	−	435,504
5	most	+	596,854
	least	−	139,502
6	many	+	388,983
	few	−	230,946
7	increase	+	35,247
	decrease	−	4791

图：英语训练语料中 add、more 相关词的使用频率显著更高

图片来源：https://onlinelibrary.wiley.com/doi/epdf/10.1111/cogs.13254

练。语言本身偏向加法。放大后变成比人类更极端的加法机器。我们面对的不是一重偏见，是双重偏见，我们自己的加法偏见加上 AI 放大后的加法偏见。做减法在 AI 时代不是可选的好习惯。是必须被设计进系统的纪律。

55,315 个 Skill

arXiv 论文 SkillReducer 拆了 55,315 个公开 Skill。Skill 正文中仅 38.5% 是 Agent 需要遵循的可操作规则，40.7% 是背景解释，12.9% 是示例，剩下的是模板和冗余。超过 60% 的内容每次调用都被注入上下文窗口，但多数时候不产生价值。研究者把这些内容压掉了，描述压缩 48%，正文压缩 39%。然后测试功能质量。压缩版得分 0.742，原版 0.722，反而提升了 2.8%。回到此前提到的 n^2 成本。上下文窗口里每个 token 都在争夺注意力。60% 是噪声时，它们在分散 Agent 对核心规则的注意力。每减少一个 token，边际收益是递增的。当工具也是类似。伯克利函数调用基准（Berkeley Function Calling Leaderboard）的数据：4 个工具时，模型选对工具的准确率 43%；扩展到 51 个跨域工具，跌到 2%。甚至做不到优雅退化，而是断崖式崩溃。

减法不是简单的删除

减法难在哪？不是动手难，是判断难。每条规则被写下来的时候都有道理。难的是判断它今天还有没有道理。AI 研究者 Carlos E. Perez 在几个不同领域里找到了同样的模式。黄石公园生态退化了几十年。解法不是投更多钱、派更多护林员。1995 年重新引入了狼——几十年前因为政策原因被猎杀掉的。当年施加猎狼政策有当年的理由，但生态变了，这个旧决策变成了负担。狼回来后，鹿群不再过度啃食，植被恢复，河岸稳定，鸟类回来了。Steve Jobs 回归苹果后砍掉了 70% 的产品线。功能机时代堆按键堆功能是行业惯例，到了触屏时代这些惯例变成了累赘。iPhone 不一定是功能最多的手机，但肯定是去掉大量过时惯例的手机。研究者Perez 称其约束流动性（Constraint Fluidity）：区分哪些规则是真实的物理边界，哪些只是历史惯性，某个时期合理、但今天已经过时的做法。然后只动后者。这个判断 AI 目前做不了。前面的实验已经说明，AI 连“减法比加法省力”都感知不到，更不可能判断一条沿用三年的规则今天还该不该留。识别什么该减，是人在 AI 工作流中绑不开的职责。

策展

也许，做减法这个词可能会引起误会，会被认为是简单粗暴的删除。更好的词是策展。删除是破坏性的。减法是数学维度的。策展是判断性的，决定什么值得留下、什么需要生长、什么应该归档。Skill 库、规则文件、记忆系统就是数字花园（Digital Garden）。需要持续策展。不只种植，更需要修剪、除草、评估。当加法的边际复杂度成本超过边际价值时，该做减法。Skill 库从0到10，每个新 Skill 价值远大于复杂度。从100到101，新

Skill 带来的冲突、路由混乱、注意力消耗很可能已超过价值。

默认姿态，先问能不能减，再问需不需要加。

从这里开始

审计你的 Skill 库。几个月没触发过的标记废弃。单个 Skill 超过500字，大概率60%不必要。可以看一下，有什么内容可以移往下一层，放到子文件夹中，按需加载。每次想加一个新 Skill，可以先想想：边际复杂度成本是多少，能不能修改现有的来覆盖。减法是反人性的，但纪律可以练。

参考文献：

1. Leidy Klotz, "Subtract: Why Getting to Less Can Mean Thinking More", Behavioral Scientist, 2021. https://behavioralscientist.org/subtract-why-getting-to-less-can-mean-thinking-more/ ↩
2. Adams G S et al., "People systematically overlook subtractive changes", Nature 592, 258 – 261, 2021. https://www.nature.com/articles/s41586-021-03380-y ↩ ↩
3. Uhler L et al., "Influence of solution efficiency and valence of instruction on additive and subtractive solution strategies in humans, GPT-4, and GPT-4o", Commun Psychol 4, 41, 2026. https://www.nature.com/articles/s44271-026-00403-0 ↩ ↩ ↩
4. Winter B et al., "More is Better: English Language Statistics are Biased Towards Addition", Cognitive Science 47(4), 2023. https://onlinelibrary.wiley.com/doi/10.1111/cogs.13254 ↩
5. SkillReducer, "Optimizing LLM Agent Skills for Token Efficiency", arXiv, 2026-03. https://arxiv.org/html/2603.29919v1 ↩ ↩
6. Tian Pan, "The Tool Selection Problem", 2026-04. https://tianpan.co/blog/2026-04-09-tool-selection-problem-agent-tool-routing-at-scale ↩
7. Carlos E. Perez, "The Paradox of Subtractive Enhancement", Intuition Machine, 2025-06. https://medium.com/intuitionmachine/the-paradox-of-subtractive-enhancement-from-mathematical-revolution-to-ecological-regeneration-f0b7029635c6 ↩ ↩ ↩
8. Maggie Appleton, "A Brief History & Ethos of the Digital Garden". https://maggieappleton.com/garden-history ↩

关键词九

去技能化：如果有一种技能我们输不起。





吴朋阳

腾讯研究院资深专家

> 那种笃定、手感、不需要想就知道下一步该怎么走的流畅，消失了。AI未必让你失业，但可能先让你失去"本来会做这件事"的能力。而你可能根本不会注意到。

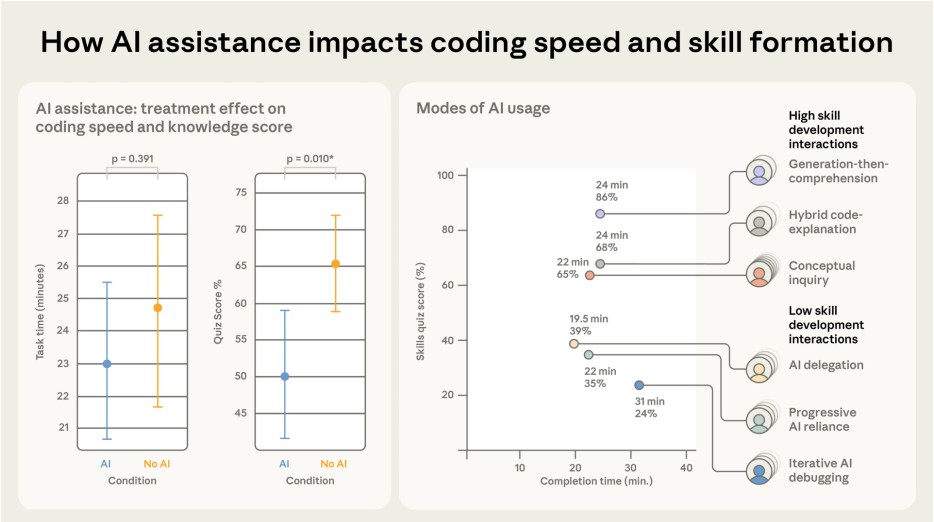
引用一个案例，重度使用AI的用户应该深有感触：一位做了25年软件开发的顾问提到，让 AI 帮自己全程开发一款应用。最开始近乎魔法。代码膨胀到10万行，事情开始不对。但真正让他不安的，是准备重新亲自上手的那一刻。“我的动作不对了。”不是不会了，但那种笃定、手感、不需要想就知道下一步该怎么走的流畅，消失了一点。这就是去技能化（Deskilling）。AI 未必立刻让你失业。但它可能先让你失去“本来会做这件事”的能力。更可怕的是，你可能根本不会注意到。

三种损失

纽约大学哲学家 Appiah 在 The Atlantic 给了一把三分法的刀，不是所有去技能化都一样疼：替代性（Substitutive）。旧技能被新工具替代。手写到打字，心算到计算器。不太需要担心。侵蚀性（Erosive）。技能因不使用而萎缩。飞行员的手动飞行能力退化。需要关注。构成性（Constitutive）。改变人之为人的核心能力。判断力、想象力、共情力。必须警惕。有些损失无关紧要，有些代价高昂，有些“具有生成性”。写作的发明消灭了记忆术，却催生了法学和科学，并不是每一项技能的消失都值得哀悼。但构成性去技能化不在这个范畴。把判断力外包出出去，外包掉的不是一项技能，是你这个人。我们在日常AI应用的过程中，很多同学都反馈，会有一种虚无感，恐怖在于不是说工作被AI替代了，因为AI大概率会催生更多的工程，而是慢慢发现，自己的判断力、想你力、共情力在减弱。

52个人，17%

Anthropic 在2026年1月的随机对照实验（RCT）。52个开发者，学一个新 Python 库。一半有 AI 辅助，一半没有。AI 组得分平均50%，对照组67%。差了17%。AI 组并没有显著更快，有人花了30%的时间在想怎么提问。其中最有意思的是分布，有6种使用模式。低分模式。AI 委托（AI Delegation），全交给 AI。渐进依赖（Progressive Reliance），逐步交出一切。迭代调试（Iterative Debugging），粘贴报错不问为什么。高分模式。概念查询（Concept Querying），只问概念自己写。混合代码-解释（Mixed Code-Explanation），要代码也要解释。生成后理解（Post-Generation Understanding），先让 AI 生成再追问到懂。得分最低的正是最高效的这组。高分模式的共同点就一个词，认知参与（Cognitive Engagement）。同样在使用AI，但主要是用 AI 支撑思考而非替代思考。这与我们此前提到的，在



图：AI 助手如何影响编码技能的形成

图片来源：https://www.anthropic.com/research/AI-assistance-coding-skills

日常研究工作中，故意设计了一些为了提升认知而设的学习摩擦。

犯错

认知心理学家 Bjork 提出过“可取困难”（Desirable Difficulties），短期内让学习更困难的条件，反而增强长期记忆和迁移能力。Anthropic 的实验直接印证了这一点，理论做出了解释。更隐蔽的问题是认知替代。让 AI 实现一个函数，它不只写了样板代码，还替你做了核心设计决策，用什么数据结构、什么处理顺序、如何处理边界。这些判断恰好是构建心智模型的关联负荷（Germane Load），是让你长脑子的部分。回看我们此前提到的 METR 实验：经验丰富的开发者用 AI 后实际慢了19%，却坚信自己快了20%。感知与现实的裂缝，部分原因可能就在于此，AI 替代了费力但有价值的认知过程，而你甚至没察觉它发生了。犯错不是意外。是学习的必要条件。正如让AI快速迭代进步的捷径，就是不断的记录错别，并反馈给自身，然后获得进步。人也是一样的，如果一直不犯错，也许我们的认知就会原地踏步，不会再进步。要珍惜AI协作过程中出现的报错。

专业领域数据

医学。顶级医学期刊《柳叶刀》（Lancet）2025年多中心研究。约1500例肠镜，19名内镜师。AI 辅助3个月后移除 AI，标准检出率从28.4%降至22.4%。仅仅三个月，而且是有经验的专业医生。法律。伊利诺伊法学院（Illinois Law School）发现使用生成式 AI 的法学生更容易犯关键错误。知识工作全域。微软研究院（Microsoft Research）和卡内基梅隆大学（CMU）发现知识工作者用 AI 后觉得任务更简单了。与此同时把问题解决能力让渡给了系统。更吊诡的是，对 AI 的信心反而提升了。

悖论

生产力的最优策略不等于学习的最优策略。Anthropic 自己的观察性研究显示，AI 可能同时加速已有能力，又阻碍新能力的形成。也就是说，你得先不靠 AI 学会一件事，才能有效地用 AI 加速它。能最好地用 AI 的人，恰恰是不用 AI 也能判断产出质量的人。那新人怎么办？加州大学圣巴巴拉分校（UCSB）的 Matt Beane 在 ACM 通讯的访谈中说，资深工程师用 AI 加速的是已有生产力，但同样的系统破坏了年轻人本该通过实践积累经验的路径。微软研究院和卡内基梅隆大学（CMU）的调查还发现一个更麻烦的现象：知识工作者用了 AI 之后觉得任务变简单了，对 AI 的信心也变高了，但实际上是把解决问题的能力让渡给了系统。信心涨了，能力没跟上。产出漂亮，内里是空的。ACM 文章给了一个准确的说法：核心专业能力的空心化（hollowing-out of core expertise）。更麻烦的是，空心化的后果往往事后才看得见。第六章的工作流飞轮需要稳定的人类判断力供给。判断力一退化，飞轮就空转。短期看不出来，当真正发现的时候再恢复可能会有点吃力。

从这里开始

每隔一两周找一个小任务。完全不用 AI 做。手感在下降、犹豫在增加，那就是信号。少吃数字垃圾食品。保留不被打断的深度思考时间。重新练习深度阅读。不是 AI 不好用。是用得太顺手了以至于忘了检查基线还在不在。外骨骼穿久了，不脱下来试试，永远不知道里面的肌肉是不是已经萎缩了。

参考文献：

1. Kwame Anthony Appiah, "The Age of De-Skilling", The Atlantic, 2025-10-26. https://www.theatlantic.com/ideas/archive/2025/10/ai-deskilling-automation-technology/684669/ ↩ ↩ ↩ ↩
2. Shen & Tamkin, "How AI Impacts Skill Formation", arXiv, 2026-01-29. https://arxiv.org/html/2601.20245v2 ↩ ↩ ↩
3. Anthropic Research, "How AI Assistance Impacts the Formation of Coding Skills", 2026-01-29. https://www.anthropic.com/research/AI-assistance-coding-skills ↩ ↩
4. Evan Moon, "Developers Who Stopped Growing", 2026-04-18. https://evan-moon.github.io/2026/04/18/developers-who-stopped-growing-in-ai-era/en/ ↩ ↩

5. Psychology Today, "AI De-Skilling: Will Chatbot Use Corrode Our Humanness?", 2025-12-16. https://www.psychologytoday.com/us/blog/virtue-in-the-media-world/202512/ai-de-skilling-will-chatbot-use-corrode-our-humanness ↩

6. ACM Communications, "The AI Deskilling Paradox", 2026. https://cacm.acm.org/news/the-ai-deskilling-paradox/ ↩ ↩ ↩ ↩ ↩ ↩

7. 不懂经©, "第一批深度使用AI的人，正在被AI去技能化", 虎嗅网, 2026-03-30. https://www.huxiu.com/article/4846729.html ↩

8. Dr Sam Illingworth, "Three Ways to Learn With AI", Slow AI (Substack), 2026. https://theslowai.substack.com/p/three-ways-to-learn-with-ai ↩

关键词十

知识工程：方法会过时，工具会迭代，最终留下来的是什么。





陈东明

腾讯研究院高级研究员

> AI越来越强，人类还能留下什么？方法会过时，工具会迭代。但你积累的判断框架、因果链条、领域直觉，这些连起来，才是别人拿不走的东西。

回到一个贯穿全篇的问题。如果 AI 越来越强，人类还能留下什么。我们前面讨论了一系列关系词，Harness、记忆、技能、评估、上下文、工作流、多Agent、加法惯性、去技能化。这些都是很重要的方法，怎么用好 AI 的方法。但方法是会过时的。Harness 空间不缩小，只移动。今天有用的设计，明天可能是障碍。那什么不会过时？我们觉得是你的知识网络，包括错误识别的模式、概念之间的因果关系、领域特定的判断框架等等，这些是帮助我们把事情做到极致的重要数据或隐性知识，它们是独特的，是从每个人的真身实践中长出来的。

从消耗到积累

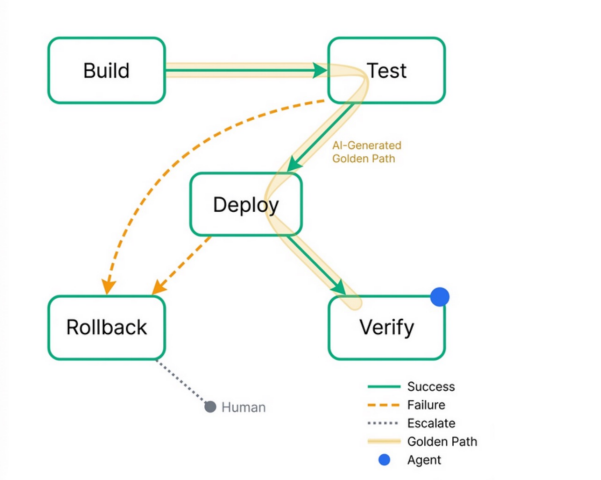
我们经常会有一种感觉，使用 AI 工具的时候，经常在做一件事：重新教它你已经教过它的东西。这个循环叫消耗型学习。你在消耗时间、消耗 token、消耗注意力，但什么都没有留下来。对话窗口一关，经验归零。并非懒，是因为消耗有即时满足感，积累只有延迟回报。打开对话窗口、输入 prompt、拿到结果，30 秒完成。而“把这个 prompt 结构化成可复用模板”至少要 20 分钟，这甚至是很多人不想用AI的重要原因，这是行为经济学的问题。基本的积累至少有三个维度，我们在前面的关键词中已经提到过：写下来，把你教过 AI 的做事方式变成 Skill。解决“重复教”。记下来，让交互留下痕迹。决策、偏好、失败教训。解决“反复忘”。连起来，让积累的东西之间产生关系。“这个部署流程依赖那个安全检查”、“上周的 bug 和三个月前的架构决策有因果关系”，这些连接才是真正的价值。三个维度是自然递进的螺旋。先写下来，积累够了再记下来，记得多了连接就会涌现。

螺旋模型

写下来、记下来、连起来，往往不需要同时，不同规模的用户，重心不同。个人先写 Skill：核心痛点是重复劳动。团队先建记忆：核心痛点是知识不传递。组织先理知识：核心痛点是一致性，三个团队三种“收入”定义。不要让组织级的框架吓退个人级的用户。LinkedIn 实施知识库后工单分类时间降 70%：那是组织级投资。对一个独立开发者来说，同样量级的回报来自写好 3 个 Skill。怎么判断自己在哪个阶段。三个问题：你有没有至少一个自己写好的 Skill？你的 AI 知道你上周做了什么决策吗？你能说清下载的N个 Skill 之间的依赖关系吗？如果连第一个回答都是没有，先写一个，写好一个。在我们观察大家AI 学习的过程中，似乎还有一个误区，“只要我学得够慢就不用学”。这可能是一条捷径，但我们很不支持团队成员这样做。犯错的过程可以帮助我们更好的了解 AI 的能力边界，也能够帮助我们了解各项能力之间的相互作用关系。常见的阶段跳跃错误：Skill 没写过，就直接下载了一堆，就搞知识图谱（过度工程）。记忆没做，就买了向量数据库（过早复杂化）。连接没做就想搞“组织级治理框架”（框架比被治理的东西还多）。

知识工程：为什么加起来还不够

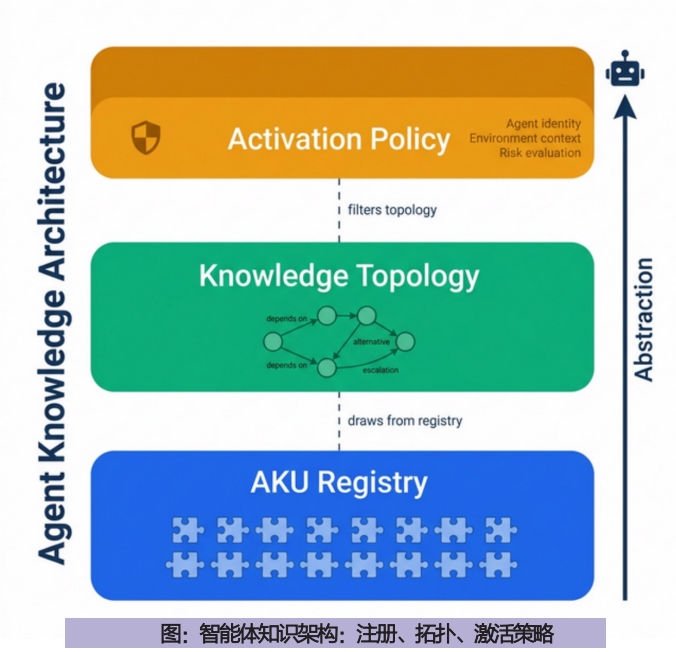
如果你只做写下来和记下来，你会得到一堆好用的 Skill 加一堆有用的记忆。已经比大多数人强了。但你的资产之间没有关系，散落在桌上的拼图碎片。一篇 arXiv 论文（Gal Bakal, Knowledge Activation）给了精准诊断：企业敏捷软件开发的瓶颈不是模型能力，是知识架构。Agent 的通用训练知识和你组织的特定做事方式之间存在结构性断裂，它不知道你们的命名规范、部署流程、那些从三次事故中总结出来的操作禁区。于是它猜测、失败、你纠正、它重试，每一轮都在消耗上下文窗口。组织部署 AI 以提高生产力，但缺乏机构知识的 AI 反而降低了生产力。解药，是把散落的资产连成结构。毕马威（KPMG）2026 年的白皮书：大数据时代已结束，大知识时代



图：可互操作技能网络：失败不迷路，试错有章法

图片来源：https://arxiv.org/pdf/2603.14805

正开始。竞争优势不来自拥有数据，而来自赋予数据以意义。他们把知识工程拆成三根柱子：语义层（业务翻译器）、本体（概念之间的关系规则）、知识图谱（让前两者活起来的可遍历网络）。OpenKG 用数据验证过这一点，结构化知识图谱将 DeepSeek 准确率从 80.7% 提升到 86.1%，只是给同一个模型提供了更好的结构化知识。结构即力量，严谨的本体结构比单纯的语料堆砌更有效，但别被组织级框架吓到。对大多数个人用户，知识图谱的投入产出比只有在连接数超过人脑能维护的阈值（大概几百个节点）时才为正。暂时不需要图数据库，给 Skill 加依赖关系、给记忆加标签、给上下文文件加版本号，连接会自然涌现。企业级知识库的典型实现也没多复杂，其实就是 YAML 配置文件 + 版本化的 markdown + 通过 API 暴露的结构化目录表。工具形式不限，但要能体现机构知识的过程链接。



图：智能体知识架构：注册、拓扑、激活策略

图片来源：https://arxiv.org/pdf/2603.14805

资产会腐烂

一个只进不出的库，最终会变成技术债的垃圾填埋场。Addy Osmani（Google Chrome 工程主管）说过：大多数工程组织擅长构建东西，很少有组织擅长移除它们。关于减法的智慧，在知识工程领域同样适用。AI 时代，腐烂速度完全不同。两种新型模式：主动腐烂（Active Rot），AI 代码不断堆积，看起来在前进实际在碎片化。休眠腐烂（Dormant Rot），遗留模块静默偏离现实，没人觉得自己负责。GitClear 数据显示 AI 生成代码重复率增长 8 倍。这个腐烂速度并不会在一开始就表现出来，但数据显示，如果不加以治理，大多数团队 3-6 个月后 AI 输出质量开始下降，原因不是模型变差，是上下文文件还在描述三个月前的代码结构，或者引用一些过时的观点。怎么对抗？斯坦福 ACE 论文：增量更新（包括新增、删除与替换）比整体重建减少 86% 的漂移。每次碰到一个旧 Skill 发现有一行过时说明，花 30 秒改掉。这 30 秒比三个月后的大规模重构有效得多。好的技能库不是大的技能库，而是小的、精确的资产库。一个人需要的 Skill 在 10-15 个，不是 89 个。一个好的知识库或者也不是越大越好，比如说我们做AI 趋势的研究研判，半年前的数据基本上属于不可用的状态，这一领域的知识变化太快了。

飞轮

到这里你可能会想：这么多要管理的事情，谁有时间。积累不需要是额外的工作，它应该是工作的副产品。Martin Fowler 观察到一个现象。团队采用 AI 后，效果要么持续提升，要么六个月后停滞。区别不在工具，在于是否有机制把每次交互的学习喂回共享工件（Shared Artifacts）。四类信号。上下文信号喂入知识库。指令信号喂入 Skill。工作流信号喂入操作手册。失败信号喂入防护栏。一个具体场景为例，开发者用 Agent 生成一个新的服务端点。代码审查发现缺少授权检查。开发者修复后，在学习日志里记下一行：“生成指令未强制执行新端点的授权检查”。这行记录已经是后续会话上下文的一部分。下一个开发者实现端点时，Agent 从第一轮就会验证授权检查。生成指令没变，上下文变了，系统学会了。

我们需要寻找的是一种极轻的实践节奏。每次会话结束问自己“有什么应该改”，Agent执行过程中各类bug与修复过程（秒级）。今日阅读的一篇精彩论文笔记，如何与原有知识库织成网络（日级）。工作过程中的复盘会议记录，如何更新后续的工作计划（迭代级）等等。Fowler 有一句话值得我们深思：区分一个“在用 AI”的团队和一个“越来越善于用 AI”的团队的，不是用多前沿的模型、skills和插件工具，而是这个团队是否有办法把每次交互变成可以共享的微小改进。

从这里开始

每次 AI 出错时花 10 秒把纠正写进上下文文件。飞轮的第一圈，不需要系统，不需要数据库，最开始一个 markdown 文件就够。文件多了之后，再让你的记忆系统，建立一个专项记忆来处理，记下来，连起来。

参考文献：

1. Tiago Forte, "Building a Second Brain Overview", 2023-05-01. https://fortelabs.com/blog/basboverview/ ↩
2. Gal Bakal, "AI Skills as the Institutional Knowledge Primitive for Agentic Software Development" (含 AKU 形式化), arXiv:2603.14805, 2026-03. https://arxiv.org/abs/2603.14805 ↩
3. Gal Bakal, "Knowledge Activation: AI Skills as the Institutional Knowledge Bridge", arXiv:2603.14805, 2026-03. https://arxiv.org/abs/2603.14805 ↩ ↩ ↩
4. KPMG, "The Big Knowledge Era", 2026. https://kpmg.com/xx/en/our-insights/ai-and-technology/big-knowledge.html ↩
5. 王昊奋 等, "知识图谱激活DeepSeek智能体，图模互补重构企业专业知识管理", 2025. https://www.53ai.com/news/knowledgegraph/2025052854289 ↩ ↩
6. GitClear, "AI Copilot Code Quality: 2025 Data Suggests 4x Growth in Duplicated Code", 2025. https://www.gitclear.com/ai_assistant_code_quality_2025_research ↩ ↩ ↩
7. Packmind, "Feedback Flywheel & Context Engineering Best Practices", 2026-04-03. https://packmind.com/context-engineering-ai-coding/context-engineering-best-practices/ ↩ ↩



《AI原生工作报告2026》

腾讯研究院