

Harness 决定你是什么人

模型决定你是人——工具与上下文，决定你能成什么事

竹小竹8167

费曼式写法 · 由多个 AI 代理撰写与互相审校

导读

模型决定你是人——工具与上下文，决定你能成什么事

先别管 AI 了，先想想你楼下的咖啡店。

两家咖啡店，买的咖啡豆是同一个供应商的，咖啡机是同一个型号。一家排队到街角，一家门可罗雀。为什么？

答案你知道：豆子只是原料。店员的训练、动线的设计、出杯的品控、熟客的口味记录——这些「豆子之外的东西」才决定了这是什么店。没有人会说「这家店的竞争力是咖啡豆」。

现在把咖啡豆换成大模型。

GPT、Claude、DeepSeek——这些模型任何人花钱都能买到，就像任何人都能买到那批咖啡豆。但同一个模型，有人用出来是个聊天玩具，有人用出来是个能自己检查作业、主动汇报进度的员工。**差别从来不在豆子里。**

这本书讲的是豆子之外的那套东西。它有个名字，叫 Harness——马具，套在马身上让它能拉车的那套装备。套在 AI 上，指的是模型之外、让模型能干活的一切：它手里有什么工具、眼前摆着什么资料、跑在什么环境里、做完的事谁来检查。

全书的论点就一句话：**模型决定你是人，Harness 决定你是什么人。**

围绕这句话，我们会拆几个反直觉的问题：为什么给 AI 配 11T 磁盘不算浪费？为什么让 AI 「自己检查作业」比换个更聪明的模型更管用？大模型玩超级马里奥为什么要暂停画面？为什么市面上绝大多数「数字员工」其实是电动算盘？为什么有的 AI 公司换个模型就死，有的反而越来越强？

最后一个问题留给自己：**就算 AI 的智商从此一分不涨，你把它用好的空间，真的用尽了吗？**

如果没有——那我们开始吧。

第一章 · 同一个大脑，两种命运

两个人，买的是同一个 AI 模型，付的是一样的订阅费。

第一个人打开聊天框，问：「帮我写个周报。」AI 写了一份，他改了改，交了。他觉得这东西不错，像个聪明的实习生。

第二个人的 AI 在帮他看一个数据仓库。它自己连上数据库，自己跑 SQL，发现数字对不上的时候自己写脚本核对，核对完把结论整理成一张表，还在表后面附上一句：「上个月的口径和这个月不一致，建议你确认一下用哪个。」这个人不觉得自己在用工具，他觉得自己雇了个员工。

同一个大脑，两种命运。差别不在模型——模型是同一个，智商一模一样。差别在模型外面包着的那层东西。

先记住这句话

我一直说一句话：**模型决定了你是人，Harness 决定了你是什么人。**

Harness 这个词直译是「马具」——套在马身上、让马能拉车的那套挽具。放在 AI 领域，它指的是**模型之外、让模型能真正干活的一切**：它能调用什么工具、能看到什么资料、跑在什么环境里、做完的事谁来检查、检查结果怎么喂回去。

大白话

大白话：模型是一颗大脑，Harness 是给这颗大脑配的身体、工具箱、工作台和办公室。大脑再聪明，没有手，连一杯水都端不起来。

这句话的类比其实非常直白。一个人是什么样的人——是流水线工人、外科医生还是探险家——不取决于他的大脑有多聪明（大多数人的大脑差不太多），取决于他手里有什么工具、见过什么世面、在什么体系里干活。给同一个人一把手术刀和一间无菌手术室，他是医生；给他同一双手但只给一把锤子，他是铁匠。

AI 也一样。GPT、Claude、DeepSeek 这些模型，就像刚从名校毕业、什么都会一点但什么都没摸过的毕业生。他聪明，但**聪明只是入场券**。他最终成为什么样的人，取决于你给他配的 Harness。

一个反直觉的推论

很多人以为，AI 好不好用，主要看追没追上最新最强的模型。新模型一发布，赶紧切换，觉得这样就能保持领先。

这本书想说服你的正好相反：**模型是大家都能买到的东西，Harness 才是你自己长出来的东西**。模型每几个月换一代，你追得再勤，和别人的差距也就几个月。但 Harness——你的工具链、你的数据、你的环境、你的流程——是复利，积累一年和积累一个月，完全是两个物种。

这就是为什么提升 Agent 的使用效率，早就过了「使用技巧」的阶段。两年前大家还在传「提示词魔法」，好像念对了咒语 AI 就变聪明。现在回头看，那只是给大脑递了张纸条。真正拉开差距的，是给大脑配了什么身体。

这本书要回答的问题

全书围绕一个问题推进：**同一个 AI 模型，为什么在不同人手里表现天差地别？**

我们会一层层往下拆：Harness 到底是什么（第二章）；为什么说工具和上下文是永远值得投资的两件事（第三章）；为什么要给 AI 配 11T 磁盘和一堆操作系统（第四章）；怎么让 AI 自己检查自己的作业（第五章）。然后看几个典型的误判：为什么大模型玩游戏必须暂停画面（第六章），为什么绝大多数「数字员工」是冒牌货（第七、八章）。再往下是最硬核的部分——真正的壁垒藏在 Agent 之下（第九到十一章）。最后聊聊人这一侧：为什么我们管理 AI 的能力远远跟不上 AI 的智商（第十二章），为什么该管的不再是会话而是结果（第十三章），以及 AI 离真正的 AGI 还缺什么（第十四章）。

读完这本书，下次再有人给你演示「我们的 AI 多厉害」，你应该能问出那个关键问题：**「模型之下的那层东西，是你们自己的吗？」**

第二章 · 给大脑配一副身体

上一章说了：模型是大脑，Harness 是大脑之外的一切。这一章把「一切」拆开看——它到底由哪几块组成，边界画在哪里。

先做个思想实验

想象一颗真正的大脑，泡在培养皿里，连着电极。你问它：「现在外面下雨吗？」

这颗大脑可能是全世界最聪明的大脑，逻辑严密、知识渊博。但它**永远答不对这个问题**。不是因为它笨，是因为它既没有眼睛，也没有窗户，甚至没有「现在」这个概念——它最后一次接触世界的信息，是培养它的时候灌进去的。

这就是裸模型。一个裸的大语言模型，本质上就是一个装在罐子里的大脑：它读过海量文本，但它看不到你的屏幕、打不开你的文件、连不上你的数据库，甚至不知道今天是几号。它唯一能做的，是对着一段文字，接龙下一段文字。

Harness 的四块拼图

把这颗大脑变成「能干活的东西」，需要四样东西。它们合起来，就是 Harness：

- **眼睛和手（工具）**。给大脑接上能感知和改变世界的接口：搜索网页、读写文件、执行代码、查数据库、发消息。没有工具的大脑只能「说」，有了工具才能「做」。
- **办公桌上的资料（上下文）**。大脑每干一件事，需要看到什么？你的项目文档、历史记录、公司的口径和规矩、上次做到哪了。上下文（context）就是「递给大脑看的那叠材料」——递什么、怎么递、递多少，直接决定它做出什么水平的判断。
- **房间（运行环境）**。大脑干活需要一个地方：一台能跑程序的机器、一个操作系统、够用的磁盘和内存、能折腾的沙箱。房间决定了它能折腾出多大的动静。
- **质检流程（闭环）**。大脑交出来的活，谁检查？怎么检查？检查不通过怎么打回去重做？没有这一条，大脑就是在「一次性许愿」；有了这一条，它才是在「迭代」。

大白话：Harness = 工具 + 上下文 + 环境 + 质检。模型负责想，Harness 负责让「想」落到地上，并且落对了。

边界画在哪

一个实用的判断标准：**换模型时，能带走的那部分，就是 Harness。**

你今天用 Claude，明天换成 DeepSeek 或者 GPT，模型换了，但你的工具接口、你的数据管道、你的验证脚本、你的环境——全都还在。这些东西不随模型贬值，反而随你的使用不断增值。这就是为什么第三章会说，Harness 里的两样东西是「永远值得投资」的。

反过来，提示词模板里那些「哄某个模型开心」的技巧，换模型就作废——那不属于 Harness，那属于对某个大脑的临时讨好。

一个容易踩的坑

很多人分不清「模型的能力」和「Harness 的能力」，于是做出错误的归因：AI 没干好，就觉得「这模型不行，等下一代」。

但请回想培养皿里的大脑：它答不出「外面下雨吗」，换一颗更聪明的大脑有用吗？没用，**换一万颗也没用，问题不在罐子里，在罐子外。**它缺的不是智商，是一扇窗。

这本书后面会看到大量这样的例子：Code Agent 几乎裸跑都能很强，因为代码世界自带窗户（第十一章）；而 Data Agent 不强，十有八九不是模型笨，是底下的分析引擎和数据管道没铺好（第九章）。学会把锅分清楚——是大脑的问题，还是身体的问题——是用好 AI 的第一个基本功。

下一章，我们讲身体里最值得花钱的两块肌肉。

第三章 · 永远值得投资的两件事

上一章把 Harness 拆成了四块拼图：工具、上下文、环境、质检。这一章回答一个更实际的问题：钱和精力有限，往哪儿砸，永远不亏？

我的答案很简单，就两个：**一个是工具，一个是 context 组织。**

为什么是「永远」

先想清楚一个问题：什么东西会贬值？

模型会贬值——GPT-4 发布时惊艳世界，两年后它只是及格线。提示词技巧会贬值——某个模型吃「深呼吸再回答」这套，下一代模型可能完全免疫。甚至你写的胶水代码也会贬值——框架一升级，全得重写。

但有两样东西不贬值：**一个好用的工具，和一套组织良好的上下文。**它们不受模型换代影响，反而模型越强，它们越值钱——因为更强的模型能把同一把好刀用出更高的水平。

大白话

大白话：给一个人配备好的工具，永远是值得投资的；给 Agent 配备好的工具，也永远是值得投资的。人会离职，电钻不会过时；模型会换代，但一个顺手的代码执行环境，换个大脑照样转。

工具：手的延伸

什么叫「给 Agent 配好工具」？不是给它一百个花里胡哨的接口，而是像给工匠配工具箱一样，每一件都**可靠、趁手、结果可预期**：

- 代码执行环境——让它能真正跑程序，而不只是「口述」程序。
- 数据库连接——让它能查真数据，而不是猜数据。
- 文件读写——让它能留下中间产物，下次接着干。
- 搜索与抓取——让它能看今天的世界，而不是去年的世界。

注意关键词是「可靠」。一个时灵时不灵的工具，比没有工具更糟——它会让模型学会「工具反正不靠谱，我瞎编一个答案吧」。这在人身上也一样：给工人一把打不准的卡尺，他很快就会放弃测量，改用眼睛估。

Context 组织：办公桌上的整洁度

第二件永远值得的事，更隐蔽，也更被低估：**组织上下文**——决定模型每次干活时「眼前摆着什么」。

打个比方。你请来一位顶级顾问，给他十分钟解决问题。方案 A：把公司三年的文件一股脑堆在他桌上，让他自己翻。方案 B：你提前整理好一页摘要、三份关键报表、一张组织架构图，按顺序摆好。

顾问是同一个顾问，聪明程度一样。但方案 B 的产出会碾压方案 A。**差距不在智商，在桌面。**

对 AI 来说，「桌面」就是它的上下文窗口。哪些资料该放进去、按什么顺序、以什么粒度、什么时候更新——这套组织工作，就是 context 组织。它包括：把公司的口径和规则写成模型能读的文档；把历史经验沉淀成可以检索的知识；把当前任务相关的信息在恰当的时刻递到恰当的位置。

这两件事为什么是复利

工具和组织好的上下文，有一个共同性质：**每用一次，就变好一点。**

工具每被踩一个坑，你就修一次，越修越可靠。上下文每服务一个任务，你就发现哪份材料多余、哪份缺失，越调越精准。而模型每升级一代，这些积累原封不动地继承下来，直接放大。

这就是为什么我说，Harness 在发展过程中，这两个是永远值得投入的「避雷针」。模型涨跌你控制不了，但工具和上下文是你自己的资产。下一章我们看一个把这条原则推到极致的真实例子——有人给 AI 配了 11T 的磁盘。

第四章 · 环境决定上限

前两章讲了道理。这一章讲一个真实的、有点极端的例子——我自己的做法。

我给我的 Agent 准备了大概 **11T 的磁盘空间、140 多 G 的内存、32G 的 GPU 显存**，还给它准备了 Windows、Mac、Linux、Docker、WSL 等各种可以折腾的环境。

第一次听到这个配置的人，反应通常是：「至于吗？不就一个 AI 吗？」

至于。而且这恰恰是「使用技巧阶段」和「堆资源阶段」的分水岭。

环境不是奢侈品，是工作台

想提升 Agent 的使用效率，早就已经过了研究「怎么提问」的阶段了。现在看的是**如何给 Agent 堆资源**。

为什么？回想第二章培养皿里的大脑。给它接一个「只能回答问题的聊天框」，和给它接一台「能装任何软件、能连任何系统、磁盘随使用」的机器，是两种完全不同的生物。前者只能动嘴，后者能动手——而且是随便怎么动手。

举个具体的场景。你让 AI 帮你分析一批数据，分析到一半它说：「我可以写个脚本并行处理，但需要 8G 内存跑起来。」如果你的环境只有 2G 内存，这个方案就胎死腹中——不是模型想不到，是环境接不住。再比如它想验证「这个软件在 Windows 上的行为和 Linux 上不一样」，如果你只有一台 Mac，它就只能猜。

大白话

大白话：模型负责「想到」，环境负责「做到」。想到做不到的次数多了，模型慢慢就只敢想那些做得到的事——它的想象力被你的硬盘大小阉割了。

为什么「能折腾」这么重要

有人会说：我又不做数据分析，要这么多系统干什么？

这里有个更深的东西。有了这些环境，Agent 才能充分发挥能力，去做各种折腾，玩出很多超乎你想象的创意出来。

注意「玩」和「折腾」这两个词。创意不是规划出来的，是试出来的。一个 Agent 想出个主意——「要不我把这些数据可视化成动画」——它需要装个库、跑个渲染、看看效果、不满意再来。这个过程里每一步都消耗资源：磁盘存中间文件，内存跑渲染，显存加速计算。环境够大，这条「想到→试→看→再想」的链就能走完；环境拮据，链条在第一步就断了，你甚至不知道它断过——AI 不会跟你抱怨，它只是默默地交出一个平庸的答案。

所以我说：环境决定了你这个 Agent 的上限有多高。模型决定下限——再差的环境，聪明模型也能给出像样的回答。但上限，那个「哇，它居然自己折腾出这个东西」的时刻，是环境给的。

一个务实的提醒

别误会成「人人都要买 11T 磁盘」。重点是思路：盘点你的 Agent 最常干的三件事，然后把这三件事需要的资源配到『不会成为瓶颈』。常处理大文件就把磁盘和内存给够；常跑代码就给个随便造不心疼的沙箱；常要跨平台验证就备好虚拟机和容器。

原则还是那个类比：你不会给一个厨师一间只有微波炉的厨房，然后抱怨他做不出满汉全席。厨师的厨艺（模型）和厨房（环境）是乘法关系——任何一项为零，结果就是零。

下一章讲 Harness 里最容易被误解的一块：闭环。很多人以为让 AI「自我迭代」就是让模型自己变聪明——不是，完全不是。

第五章 · 让 AI 自己检查作业

前面几章讲了 Harness 的静态部分：工具、上下文、环境。这一章讲它的动态部分——也是最容易被误解的部分：**闭环**。

从 Harness 角度，想提高使用 Agent 的效率，核心是为 Agent 构建一套可以**自我开发、自我验证、自我测试、自我迭代**的环境。

先别急，这里有个必须立刻澄清的误会。

不是模型在迭代，是工作在迭代

很多人一听「自我迭代」，以为说的是让 AI 自己训练自己、越用越聪明。不是。**模型的参数从头到尾没变过**——它今天和昨天一样聪明，既不更笨也不更聪明（为什么这很重要，第十四章会专门讲）。

迭代的是**它手里的那件工作**。写一版代码 → 跑一遍 → 报错 → 改 → 再跑 → 测试通过。模型还是那个模型，但代码从第一版烂稿变成了能跑的成品。就像同一个学生，把草稿改了三遍才交——他的智商没变，但分数变了。

大白话

大白话：闭环不是让 AI 变聪明，是让 AI 别交草稿。聪明程度不变，但允许它「写完自己检查一遍再交」，质量天差地别。

闭环的四个环节

一个完整的闭环长这样：

1. **自我开发**：Agent 做出一个东西——一段代码、一份分析、一个脚本。
2. **自我验证**：它自己检查这个东西对不对——代码能跑吗？数字对得上吗？
3. **自我测试**：它自己构造考验——边界情况行不行？换个输入还成立吗？
4. **自我迭代**：发现问题，回到第一步改，直到通过为止。

关键在最后一环往回跳的那个箭头。没有这个箭头，前面三步只是「做得比较仔细的许愿」。

闭环是基础设施，不是提示词

这是本章最想强调的一点：**你需要能够让 Agent 完成一整个闭环，并且提供一套基础设施，让它完成这么一个闭环。**

注意是「基础设施」，不是「指令」。你在提示词里写「请检查你的答案」，这不叫闭环——这只是让它「口头检查」。它说「检查过了没问题」，你信吗？它并没有真的跑一遍。

真正的闭环是物理的：Agent 写完代码，环境里真的有一个解释器能执行它，执行结果真的喂回给 Agent，失败了它真的能再改一版。每一个环节都是「真的发生」，而不是「嘴上发生」。这就是为什么闭环属于 Harness 而不属于模型——**执行的机器、回传的管道、重试的机制，全在模型之外。**

打个比方：你让员工「写完报告自己校对一遍」，和他桌上真的有一台能打印出校对稿的打印机、一支红笔、一套「改完才能归档」的流程——前者是许愿，后者是制度。闭环必须是制度。

闭环和多聪明没关系

一个反直觉的事实：闭环能补齐的，恰恰是不够聪明的部分。一个 80 分的模型配完整闭环，产出经常超过一个 95 分的模型裸跑——因为 95 分的模型也会犯 5 分的错，而闭环的作用就是在交付前把那 5 分抓回来。

这也是为什么本书后面会说：就算模型的智商从此冻结，我们的工作方式还有巨大提升空间。闭环就是最大的那个旋钮之一。

下一章换个角度，看一个 Harness 救场的极端案例：让反应以秒计的大模型，去玩反应以毫秒计的游戏。

第六章 · 会暂停的游戏

你可能刷到过这样的视频：AI 在玩超级马里奥，跳蘑菇、躲乌龟，玩得有模有样。弹幕一片「卧槽」。

我第一次听到用大模型玩游戏，反应和很多人一样：**这怎么可能？**游戏是实时的——乌龟每秒爬过来一格，你必须在一瞬间做出反应。而大模型思考一次要好几秒。等它想好「该跳了」，马里奥早就死三回了。

后来我才知道真相：**每一次画面都会先暂停，等大模型思考完成、做出决策以后，再重新把游戏恢复。**是这么来玩游戏的。

听起来像个作弊的小聪明。但我越想越觉得，这个「暂停」里藏着理解 Harness 的一把钥匙。

时间尺度的错配

问题本质上是：**世界的时钟和模型的时钟，走得不一样快。**

游戏的世界以毫秒计，模型的世界以秒计——差了上千倍。硬把它们接在一起，就像让一个人类棋手去参加「每秒必须落子十次」的比赛：不是棋力问题，是生理结构不允许。

而那个「暂停键」做的事，本质上就是：**在两个时钟之间当一个翻译官。**世界走一步，翻译官喊「停」，把当前局面拍成快照递给模型；模型想多久都行，想完说「往右跳」；翻译官执行这个动作，再喊「继续」。对模型来说，游戏从实时制变成了回合制；对游戏来说，玩家从「反应快」变成了「想得深」。

大白话

大白话：模型玩不了实时游戏，就像人类棋手下不了每秒十步的棋。但你可以改规则——变成回合制。改规则这件事不归棋手管，归比赛组织方管。Harness 就是那个组织方。

「暂停」是一种通用的 Harness 模式

别只把它当成游戏技巧。这个模式无处不在：

- **AI 操作电脑**：截屏 → 暂停 → 模型看截图想「点哪里」 → 执行点击 → 再截屏。屏幕在两次截屏之间其实是「冻结」在模型的认知里的。
- **AI 交易/监控**：市场每秒在变，但 Agent 是定时被唤醒的——每次醒来看到的是一张快照，决策完再睡过去。
- **第五章的闭环**：代码「跑完出结果」才喂回模型——本质也是把连续世界切成模型能消化的一帧一帧。

所有这些场景里，Harness 都在做同一件事：**把模型跟不上节奏的世界，切成模型跟得上的片段**。我把它叫「节拍器」——世界按自己的速度走，Harness 负责告诉模型「现在轮到你思考了」。节拍器打得越聪明，同一个模型的表现就越好。

反过来：什么时候不该暂停

理解了「暂停」为什么有效，就能看清它的代价：**暂停期间，世界是失控的**。

游戏暂停时乌龟不会动，所以没事。但现实里很多事不能暂停——你不能让股市暂停等你思考，不能让生产线上的机械臂暂停。这些场景要么不用大模型（用毫秒级的专用算法），要么把决策权分层：快的事交给快的系统，慢的事（策略、规划、复盘）交给大模型。

这又是一个「锅该分清楚」的例子（第二章）：AI 玩不好实时游戏，不是模型笨，是没有人给它装暂停键。**时间尺度的错配，永远是 Harness 的责任，不是模型的责任**。

下一章换一个战场：聊聊「数字员工」——一个被严重注水的词。你会发现，注水的原因恰好也和 Harness 有关。

第七章 · 冒牌数字员工

这两年「数字员工」这个词特别火。打开各种发布会，到处都在宣称：「我们有了 7×24 小时的数字员工」「一个 AI 顶一个团队」。

很多人号称自己的数字员工多牛逼多牛逼。但你扒开看，**实际上这些所谓的数字员工，就是一个工具，丢一个 prompt 驱动一下**，就号称有数字员工了。

高级一点的，好歹还会弄一个事件驱动来喂这个 prompt——意思是：出事了（比如来了一封新邮件），系统自动把这件事包装成一段话丢给 AI 处理。要么就是通过定时任务搞一个 workflow——每天早八点，自动驱动几个 Agent 各跑一遍。

哎，这也就叫数字员工了。

你觉得这样叫数字员工合适吗？

用「人」的标准量一量

判断一个东西配不配叫「员工」，最简单的办法是回到这个词本身：你对**真人**员工的定义是什么？

是给了他一个大目标，给了他一个 KPI（关键绩效指标——说人话就是「到月底拿什么数字考核你」），给了他一个约束（预算多少、什么不能碰），OK 了之后，他可能月底或者一周给你做一次汇报。**中间是他来找你，不是你来找他。**

再看那些「数字员工」：来了一封邮件，你（的系统）戳它一下，它处理一下。你不戳，它就躺在那儿。它不是员工，它是**算盘**——你拨一下，它动一下。事件驱动顶多是把「你用手拨」换成了「邮件来了自动拨」，拨的还是你。

大白话

大白话：员工和算盘的区别，不在于算得快不快，在于谁发起动作。员工自己知道该干什么、干得不对劲会来找你；算盘永远在等别人拨。目前市面上绝大多数「数字员工」，都是电动算盘。

注水注在哪

现在回头看第一章那个框架，注水点一目了然：**这些「数字员工」只有模型，几乎没有 Harness。**

一个没有工具、没有环境、没有闭环的模型，加上一个定时戳它的闹钟，包装一下就敢叫「员工」。这就像雇了一个人，但不给他工位、不给他电脑、不告诉他公司的任何情况，只许他每天来办公室坐十分钟回答一个问题——然后对外宣称「我们团队扩招了」。

为什么要较真这个词

有人会说：不就是个叫法吗，至于吗？

至于。**名字决定预期，预期决定投资方向。**如果你以为买个「数字员工」就完事了，你就不会去投资它底下的东西——工具、环境、闭环、数据管道。结果买来的「员工」干不了活，你得出的结论是「AI 不行」。错了，不是 AI 不行，是你雇的那个根本不是员工。

这个词被注水，伤害的恰恰是想认真做事的人：老板的预期被骗子拉高，预算被噱头项目分走，真正扎实的 Harness 建设反而显得「不性感」。

那真正的数字员工应该长什么样？下一章我们把「他来找你，不是你找他」这句话拆开——它是一个可以逐条检查的技术清单。

第八章·真员工长什么样

上一章立了个靶子：市面上大多数「数字员工」是电动算盘——你不拨，它不动。这一章把正面标准立起来：真员工长什么样，逐条拆成技术上能检查的东西。

真员工的四块拼图

回到上一章的定义：你给他一个大目标、一个 KPI、一个约束，然后他每周或每月汇报一次，中间**他来找你，不是你找他**。这句话听着朴素，但拆开是四个硬要求：

1. **目标要活得比会话长**。真员工的目标存在他脑子里，不因为你下班就消失。而对 AI 来说，「记住大目标」意味着 Harness 必须有一块持久的地方存放目标，每天醒来先读一遍——「我这一季度要把报表错误率降到 1% 以下」。会话结束目标还在，这是第一层。
2. **约束要内化成护栏**。预算不能超、客户数据不能外传——真员工把这些刻在习惯里，不用你盯着。对应到 AI，约束不能指望「模型自觉」，得做成环境和工具层的硬规则：API 额度、权限分级、写操作前的确认机制。
3. **进度要自己跟自己核**。真员工每晚心里过一遍「今天推进了啥、离 KPI 还差多远」。这就是第五章的闭环换了个时标——从「这次任务对不对」拉长到「这个月进度够不够」。Harness 里需要有一条定期自评的链路。
4. **汇报是员工主动发起的**。注意方向：汇报是**员工写给老板看**，不是老板去查员工。AI 版本就是：Agent 定期产出状态摘要，遇到自己判断不了的事（比如「两个数据源打架了，用哪个？」）**主动上报**，而不是闷头瞎选一个。

大白话

大白话：真员工 = 记住目标 + 守住底线 + 自己核进度 + 有事主动说。四条都做到，你才敢「给他一个目标就不管了」。少一条，你就得盯着；盯着的那个东西，叫工具，不叫员工。

自主性是一条光谱，不是一个开关

这四条不要求一步到位。人和工具之间不是非黑即白，而是一条光谱：

- **拨一下动一下**（纯工具）：上一章的电动算盘。
- **定时自己动，但只会例行公事**（闹钟级）：定时任务驱动的 Agent。有「自己动」了，但动的内容永远是那几样，遇到意外不会变招。
- **有事会喊人**（学徒级）：例行公事照常做，但遇到拿不准的知道停下来问。别小看这一条——「知道自己不知道」是自主性的第一块基石。
- **给目标自己跑，定期汇报**（员工级）：四条拼图齐了。

有意思的是，每一级往上走，加的东西**几乎都不在模型里**：持久记忆、护栏、自评链路、上报通道——全是 Harness。同一个模型，套在算盘级 Harness 里是算盘，套在员工级 Harness 里是员工。这又一次回到全书那句话：模型决定你是人，Harness 决定你是什么人。

一个现实的提醒

「员工级」不是银弹。真员工会犯错、会偷懒、会报喜不报忧，AI 员工也会——而且它的「报喜不报忧」更隐蔽，因为它不会心虚，它可能真的以为自己干对了。所以即便到了员工级，第五章的闭环和约束护栏依然是地基，只是你检查的频率从「每次都看」降到了「定期抽查」。

前面八章都在讲 Harness 怎么放大模型。从下一章开始，视角再往下沉一层：有些领域里，Harness 已经厚到成了真正的壁垒——厚到模型反而是可以被替换的那个零件。

第九章 · Agent 之下的壁垒

前面八章一直在说：Harness 很厚、很重要。这一章要把话说得更重一点——在某些领域，**壁垒根本不在 Agent 这一层，而在 Agent 之下。**

什么叫「Agent 之下」？一个做数据分析的 Agent，结构大致是这样叠的：最上面是模型（负责理解你的问题、规划分析步骤）；中间是 Harness（工具、上下文、闭环）；**最底下还压着一层**——真正执行分析的东西：分析语言、计算引擎、数据管道。模型的命令最终要翻译成这一层能执行的指令，才能碰到数据。

这一层，就是我说的「Agent 之下的壁垒」。在 Data Agent 里，它就是**底下的分析语言和引擎**。

为什么壁垒会沉到最底下

想明白一件事：大模型做数据分析时，它自己其实**不算数**。

你问「上个季度哪个区域增长最快」，模型不会自己去翻一百万行数据——它干的是「想清楚该算什么、写成一句 SQL 或一段分析代码、交给底下的引擎去执行、再把结果读回来解读」。真正和数据肉搏的是引擎。

这就带来一个残酷的推论：**引擎的能力上限，就是 Agent 的能力上限**。引擎算不动十亿行，模型再聪明也只能绕着小数据打转；引擎不支持某种关联分析，模型规划得再漂亮也执行不了；引擎慢得要等一小时，第五章那个「快速迭代闭环」就转不起来。

大白话

大白话：模型是军师，引擎是士兵。军师谋略再高，士兵只有三个，也打不了大仗。而练兵——把士兵从三个练成三万——是极其耗时耗钱的硬功夫，这才是别人抄不走的东西。

这层壁垒为什么难抄

模型是公共品——你可以调 API，别人也可以。上层的 Agent 编排逻辑，看看 demo 也能模仿个七八成。但底下的分析引擎，是另一回事：

- **它要解决真问题**：十亿行数据秒级响应、上百张表正确关联、口径变了全局一致——每一个都是十年量级的工程。
- **它的能力接口要和模型磨合**：模型「想做什么」和引擎「能做什么」之间的翻译层，错一点就出错误结果，而数据分析里错误结果比没结果更糟。
- **它沉淀的是领域知识**：分析语言里凝结了「这个行业的分析到底怎么做」的理解（下一章的三种融合形态就是例子），这不是写代码，是写认知。

换个角度看竞争

理解了「壁垒在下面」，你看 AI 公司的方式就变了。别问「你用了什么模型」——答案大家都一样。要问：**「模型之下，哪一层是你自己的？」**

做 Data Agent 的，答案应该是分析语言和引擎；做编程 Agent 的，答案可能是代码索引和执行沙箱；做客服 Agent 的，答案可能是工单系统和知识库管道。答不上来的公司，做的是「模型的包装纸」——模型一升级，包装纸就作废；答得上来的，模型升级对它反而是利好，因为引擎不变，换了个更聪明的军师而已。

下一章我们钻进这层壁垒的内部，看一个具体而硬核的例子：「结构化 + 非结构化融合分析」这句黑话，扒开其实是三种实实在在的形态。

第十章 · 表格与文档的握手

上一章说了，Data Agent 的壁垒在底下的分析语言和引擎。这一章钻进这层壁垒内部，拆一句到处都在讲、但很少有人讲清楚的黑话：「**结构化 + 非结构化融合分析**」。

先翻译两个词。结构化数据就是表格——一行一行、一列一列，数字和短文本，SQL 能算的那种。非结构化数据就是表格之外的一切：PDF 合同、聊天截图、Word 文档、会议纪要。企业的真实信息，大头其实在后者。

所谓「融合分析」，不是一个玄乎的新技术，其实是**三种实实在在的形态**。搞清楚这三种，这句黑话就再也唬不住你。

形态一：把文档抽成表，交给 SQL 算

第一种：把 **PDF、截图、Word 这些非结构化资料抽成表，交给 SQL 算**。

逻辑很朴素：SQL 是几十年磨出来的、算表格最可靠的工具，那就别跟它抢活——大模型只负责把「不是表的东西」变成「表」。一百份 PDF 合同，让模型逐份读，抽出「甲方、金额、期限、违约条款」填进一张表；抽完之后，「哪个客户违约风险最高」这种问题，SQL 一秒钟算完，一行都不会错。

分工原则：模型干它擅长的「读懂」，引擎干它擅长的「算准」。模型不碰算术，引擎不碰阅读。

形态二：把企业的「规矩」召回，绑进判断链

第二种更微妙：把**企业的口径、规则、经验这些文档知识召回，绑进数据分析的判断链里**。

什么叫「口径」？举个例子：「活跃用户」这个词，市场部定义为「月登录一次」，产品部定义为「周使用三次」。同一个词，两套规矩。裸模型算「活跃用户有多少」时，它**根本不知道你们公司用哪套规矩**，只会猜一个，还猜得理直气壮。

形态二做的事：分析开始前，先把相关的公司文档（口径定义、业务规则、历史经验）检索出来，垫在模型的判断底下。模型算之前先「读了员工手册」，答案自然按公司的规矩来。

这一步就是第三章说的「context 组织」在数据领域的具体化——**企业的隐性知识，不组织好递给模型，就等于不存在。**

形态三：表里的长文本，用大模型 UDF 按行处理

第三种最容易被忽略：**表里本来就有的长文本字段，用大模型 UDF 按行处理，产出的还是一张表。**

UDF（用户自定义函数）是 SQL 里的老机制：你可以在查询里调用一个函数，对每一行做自定义处理。传统 UDF 只能写死板的规则，比如「把字符串转大写」。现在函数体可以是一次大模型调用——对十万条客服记录逐条调用模型判断「这条是投诉还是咨询」，结果写回新的一列。

注意这个形态的美感：**进去是一张表，出来还是一张表。**模型被规训成了计算引擎内部的一个零件，处理完一百万行文本之后，下游的 SQL 该怎么算还怎么算。AI 没有颠覆数据分析的流水线，它是作为一个新的「零件型号」插进了老流水线。

大白话

大白话：三种形态一句话总结——文档变表（模型读、SQL 算）、规矩垫底下（先读员工手册再答题）、模型当函数（按行处理表里的文本，出来还是表）。共同原则只有一个：让模型干阅读的活，让引擎干算术的活，别让它们串岗。

为什么这三种形态是壁垒

回头看第九章：这三种形态，每一种都要引擎和分析语言原生支持才行——文档抽取要接得上表、知识召回要绑得进查询计划、UDF 要在引擎内部高效调度模型调用。这就是为什么它是「Agent 之下的壁垒」：换个模型，三种形态照旧运转；没有这层引擎，模型再聪明，也只能干第一种里的「读」，后面两头都接不上。

下一章把视野拉回来：不是所有领域的 Agent 都需要这么厚的底座。有的 Agent 几乎是裸的模型在跑——这种差异本身，就是一张有价值的地图。

第十一章 · 谁吃模型，谁吃 Harness

第十、九章把 Data Agent 的底座讲得很厚。但请别推出一个错误结论：「所有 Agent 都得配重甲」。真实的世界更有意思——**不同领域的 Agent, Harness 的厚度差别巨大**。这张厚度地图，本身就是一件有用的工具。

最薄的地方：Code Agent

在不同领域的 Agent 里，Harness 最薄的是 Code Agent（编程 Agent）。

为什么？回想第二章的「四块拼图」，编程这个世界几乎**白送**了其中好几块：

- **质检白送**：代码写得对不对，编译器一跑就知道，测试一跑就明白。验证是免费的、客观的、毫秒级的——不像别的领域，「对不对」需要人来判断。
- **环境白送**：代码是纯文本，读写执行一台普通电脑就够，不用 11T 磁盘。
- **上下文相对简单**：代码库自带结构（文件、目录、版本历史），「该给模型看哪些文件」这件事比「该给它看哪些企业知识」简单一个量级。
- **世界是纯文字的**：输入输出都是文本，正好是模型的母语。

所以 Code Agent 基本上**完全吃模型能力**：模型强一分，产品就好一分。这也是为什么编程是 AI 落地最快的领域——不是做产品的人多厉害，是这个赛道的地最平。

最厚的地方：Data Agent

地图的另一端是 Data Agent，我觉得难度最高。其他各个领域的 Agent，也会比 Code Agent 的 Harness 更厚一些。

厚在哪？前两章其实已经答了：底下要有自己的分析语言和引擎（第九章），要处理结构化和非结构化的三种融合形态（第十章），还要搞定口径、权限、数据管道……这些没有一样白送，全得自己建。**Code Agent 是在平地上跑，Data Agent 是先修路再跑。**

一个用成绩说话的案例

厚 Harness 的回报是什么？看个例子。像 InfiniSynapse（一个数据分析 Agent），它其实就非常吃 Harness 的能力。团队用 DeepSeek V4 Pro Preview 版，就做到了全球 Spider 榜单第六名（内部结果）。

说明一下背景：Spider 是学界公认的文字转 SQL 榜单——给你一句人话问题和一个数据库，考你能不能生成正确的查询语句。注意这个成绩的细节：**用的不是最顶尖的模型**。

DeepSeek V4 Pro Preview 不是榜单上最贵的脑子，但配上厚的 Harness——自己的分析语言、引擎、口径处理——就挤进了全球前列。

大白话

大白话：这个榜单成绩就是全书主论点的实锤。同一个考场上，别人靠更贵的脑子，它靠更厚的身体。模型决定下限，Harness 决定上限。

这张地图怎么用

对做产品的人：选一个赛道前，先问「这个领域的 Harness 有多厚」。薄的赛道（如编程）拼的是谁先用上最强的模型，护城河浅；厚的赛道（如数据分析、医疗、法律）前期修路苦，但路修好了就是你的。

对用 AI 的人：评价一个 Agent 产品时，先判断它所在领域该有的 Harness 厚度，再看它有没有。**在厚赛道里卖薄产品的，都是上一章说的电动算盘。**

到这里，全书关于「机器一侧」的论述讲完了。最后三章，转向人的一侧——因为这场竞赛里，真正拖后腿的其实是人。

第十二章 · 智商不涨之后

前面十一章讲的都是怎么给 AI 配装备。这一章问一个扎心的问题：**如果 AI 的智商从今天起一分都不涨了，会怎么样？**

很多人的直觉是「那 AI 就到头了」。我的看法正好相反：**就算 AI 现在的智商不再提升，我们管理 AI 的能力也远远落后于 AI 智商的提升。**反过来，如果 AI 的智商不提升，我们管理 AI 的能力依然还有大量提升空间。

两条曲线，差得有多远

想象两条曲线。一条是模型的智商：过去三年，它像坐火箭，每隔几个月跳一个台阶。另一条是「我们使用 AI 的能力」：工具链、流程、组织方式、协作习惯。这条曲线也在涨，但涨得像爬楼梯——而且大部分人和公司还停在负一层，用着「打开聊天框问一句答一句」的原始方式。

两条曲线的差值，就是**被浪费的 AI 智商**。一个残酷的观察：今天大多数人从顶级模型身上榨出来的价值，可能不到它能给的三成。就像一个组织雇了个天才，却只让他贴发票——然后抱怨「天才也没多厉害」。

旋钮理论

我们的工作中有很多旋钮——拧一下就变好的调节装置。AI 智商的自我提升是一个旋钮，我们使用 AI 的能力是另一个旋钮。关键是：**每一个旋钮都可以极大地提升效率，而且它们互不阻塞。**

这本书其实已经把这个「使用旋钮」拆成了几个小旋钮，每个都拧过一次：

- 工具旋钮（第三章）：模型没变，接上可靠的工具，产出上一个台阶。
- 上下文旋钮（第三章）：模型没变，资料组织好递过去，判断准一个台阶。
- 环境旋钮（第四章）：模型没变，磁盘内存系统给够，能折腾的事多一个量级。
- 闭环旋钮（第五章）：模型没变，允许它自查自改，错误率掉一个台阶。

- 节奏旋钮（第六章）：模型没变，把时间尺度错配翻译好，能做的任务多一类。
- 底座旋钮（第九到十一章）：模型没变，底下的引擎和语言建厚，能力上限整个抬高。

注意每一句的开头都是「模型没变」。这就是旋钮理论的要点：**这些提升和模型换代无关，今天就可以拧，拧了就见效。**而追新模型那个旋钮，反而控制不了——你只能等公司发布。

大白话

大白话：一边等工厂出更快的发动机，一边把自己这辆车的变速箱、轮胎、驾驶技术全升一遍。后者全在你手里，而且每一项都立竿见影。

这章对不同的人意味着什么

对个人：焦虑「模型又更新了，我的技巧过时了」大可不必。过时的只是讨好某个模型的花招；你在工具、上下文、流程上的积累，随每次模型升级自动增值。

对团队：别把所有预算押在「等更强的模型」上。那条曲线你控制不了。把一半预算投给自己的 Harness——这是你能控制的曲线，而且竞争对手未必在爬。

对悲观者：「AI 智商涨不动了」即使成真，也远不是终局。现有智商被充分利用的那一天，离现在还很远——光是「用好已经存在的聪明」，就够整个行业忙很多年。

下一章讲一个具体的、正在发生的换挡：AI 产品该管的东西，正在从「会话」变成「结果」。

第十三章 · 从管会话到管结果

打开你手机里的 AI 应用，看看它帮你「管理」了什么。

十有八九，答案是一列**会话**：左边一栏聊天记录，按时间排好，能搜索，能重命名，能删除。现在做 AI 产品的，大家其实还在做**会话管理**——管理「你和 AI 聊过什么」。

但我觉得这个方向错了。我现在关注的，是 **AI 给我的结果的管理，而不是会话管理**。如何高效地呈现 AI 给我的结果，并方便我做管理，变得很重要。这是做 AI 产品的人应该多去思考的。

会话是过程，结果才是资产

想一个道理：你和一位顾问合作三个月，最后值钱的是什么？

不是你们的会议纪要——虽然它记录了过程。值钱的是那几份交付物：一份分析报告、一套方案、一张决策表。会议记录堆在那儿，三个月后你自己都不会再翻；但那份方案你会用、会改、会拿去说服别人。

现在的 AI 产品把全部心思花在了管「会议纪要」上，而对「交付物」几乎不管。AI 帮你写了一版方案，混在几十轮对话的第三条里；过了一周你想找它，得在聊天记录里考古。更糟的是，方案改了五版，每一版散在不同对话里，**最新版是哪一版，只有天知道**。

大白话

大白话：会话管理管的是「垃圾邮件式的流水账」，结果管理管的是「你真正要带走的东西」。一个是录像带，一个是作品库。现在所有人的产品都在造录像带仓库。

结果管理长什么样

把「结果」当一等公民来管理，产品形态会很不一样：

- **结果是独立对象**：一份报告、一张表、一段代码，从对话里「长出来」之后独立存在，有名字、有版本、有状态（草稿/已定稿）。对话只是它的出生证明。

- **版本是向前滚的**：改了五版就叠成一条版本链，默认呈现最新版，旧版可回溯——而不是五份相似的文本散落在五个对话里。
- **结果可以再加工**：基于这份报告，让 AI 续写、换算、做图——结果本身成为下一个任务的输入。这其实就是第三章「context 组织」的用户界面版。
- **结果是可盘点的**：像看文件柜一样看「AI 给我产出的全部东西」，而不是像翻监控录像一样翻聊天。

为什么这是「下一个战场」

回看全书的逻辑链：Harness 越厚，Agent 干活越多、干得越久（第七、八章的数字员工）；干活越多越久，产出的结果就越多越杂；结果越多越杂，「管结果」就越痛。**会话管理是为「问答时代的 AI」设计的，结果管理才是为「干活时代的 AI」设计的。**

这个换挡本质上也是 Harness 思路的自然延伸，只是从「机器一侧」延伸到了「人一侧」：前面十二章讲怎么让 AI 干好活，这一章讲干完的活怎么交到你手里、你怎么接住。AI 的产出如果不能被人高效地验收、整理、复用，前面所有的工具和闭环都在最后一公里漏掉了。

所以，对做产品的人：别再卷「聊天框体验」了，卷「交付物管理」。对用 AI 的人：评估一个工具时多问一句——「它给我的东西，我能找回来、能接着用、能管起来吗？」

最后一章，把视野拉到最大：说了这么多 Harness 能补的东西，那什么是 Harness 补不了的？答案是：目前 AI 身上还缺一种能力，缺了它，AI 就不是 AGI。

第十四章 · 为什么还不是 AGI

这本书几乎一直在说 Harness 的好话：工具能配、环境能堆、闭环能建、上下文能组织。最后一章，该诚实地讲讲边界了——**什么是 Harness 再厚也补不了的东西？**

答案：个体演化能力。这也是我认为当前 AI 不能称为 AGI（通用人工智能——说人话就是「像人一样什么都能学、越干越强的智能」）的原因：**它没有个体演化能力，也就是参数是不更新的。**

「参数不更新」是什么意思

先翻译。模型的「参数」是它大脑里几十亿个旋钮的刻度，训练的时候调好，之后**封存**。你今天和它聊了一下午，它帮你改了十版方案，对你的项目了如指掌——然后呢？会话一关，这些全没了。明天它见到你，还是那个刚从工厂出来的它，你上次教它的东西，它一个字都没学会。

这 and 人的差距是根本性的。你入职第一天的你和第三年的你，大脑里的「参数」是真的变了——神经连接被三年的经验重写过。而 AI 的「经验」只是每次会话临时垫在桌上的资料（第三章的上下文），资料一撤，打回原形。

大白话

大白话：现在的 AI 像一个每天失忆的天才员工。你可以每天早上把员工手册重新递给他（这就是上下文工程），但他本人永远不会因为干了三年而变成老员工。

缺了演化，具体缺什么

「参数不更新」听着抽象，落到团队里是四样具体的东西，每一样都是老员工才有、天才实习生没有的：

- **品味**。一个方案「感觉不对」——这种判断力是几百次成败刻进直觉的。模型有通用的好品味，但没有「我们团队」的品味：你们老板讨厌什么样的汇报、这个客户吃哪一套，它永远要从零学。

- **记忆**。不是上下文窗口里那种便利贴式的记忆，是「三年前那个项目为什么失败」这种长在身上的记忆。便利贴会丢，长在身上的不会。
- **感知**。老员工走进办公室，空气不对都能察觉——团队在焦虑、客户在犹豫。模型的感知范围只有每次递给它的那叠材料，材料之外的世界对它不存在。
- **独当一面的探索能力**。给一个模糊的方向，老员工能自己开路、自己承担后果、在没人管的几个月里长出新本事。这正是第八章「员工级」的终极形态——而没有个体演化，它最多是一个「每天重置」的员工。

补上它需要算力和算法的双重突破

为什么不让模型「用着用着自己更新参数」呢？因为这比现在所有的 AI 工程都难。

一是**算力**：训练一次大模型要烧掉一个机房几周，而「个体演化」要求每个实例、每天、持续地更新自己——量级完全不同。二是**算法**：今天的训练方式适合「一次性从海量数据里学」，不适合「从自己昨天的一次失败里学」——从单一样本里稳健地学到东西、又不把已经会的忘掉（这有个名字叫「灾难性遗忘」），是未解决的研究问题。

这不是工程排期问题，是两个硬突破都还没发生。在此之前，「天才实习生 + 每天早上递员工手册」就是技术现实，而全书讲的 Harness——工具、上下文、环境、闭环——就是让这套现实运转到极限的全部办法。

回到开头那句话

还有个必须保持的清醒：别以为有了 AI，真问题就变简单了。**真正的软件系统依然复杂度很高，但 AI 带来了大量助力**。复杂度没有被消灭，只是你和它之间多了一个聪明的中间层。

所以回到第一章：模型决定你是人，Harness 决定你是什么人。现在可以给这句话加上最后一层意思——

AI 的智商曲线，你控制不了，交给实验室。个体演化的突破，你也控制不了，交给时间。**但 Harness 是你的：你的工具、你的上下文、你的环境、你的闭环**。它是你在「AI 每天失忆」的时代里，唯一能把经验沉淀下来的地方——模型学不会记住你，但你的 Harness 可以。

从这个意义上说，建设 Harness，就是建设你和 AI 之间那段「不会失忆的关系」。

这就是你的避雷针，也是你的护城河。开始动手吧。

Harness 决定你是什么人 · 竹小竹8167