课程学习讲义

Harness Engineering：有时候语言模型不是不够聪明，只是没有被好好引导

cnfjlhj & Codex

2026-05-29

视频作者/频道：啥都会一点的研究生
发布日期：课程合集发布日期：2026-03-13；本节分 P：第 13 节
本节时长：01:32:21
视频链接：https://www.bilibili.com/video/BV1RQwJzKEMz?p=13
证据说明：B 站接口未提供可用公开字幕，且显示需要登录字幕。本讲义基于完整音频的 faster-whisper small ASR、完整 480p 视频关键帧和课程画面整理；术语按上下文统一，例如 Harness、Context Engineering、Workflow、AGENTS.md、Claude Code、OpenCode、SWE-agent、TauBench、MetaHarness 等。

目录

1 课程定位：为什么要专门讲 Harness Engineering

这一讲的主轴可以压缩成一句话：有时候语言模型不是不够聪明，而是缺少一套能让它把能力用出来的外壳。这里的“外壳”就是 Harness。它不只是 prompt，也不只是工具列表，而是围绕语言模型组织任务执行的整套环境：规则、上下文、文件、工具、权限、工作流、反馈、记忆和评估。

图 1：课程标题页：Harness Engineering 被解释为“有时候语言模型不是不够聪明，只是没有人类好好引导”。[2]

讲者选择从一个小模型的代码修复实验切入。直觉上，小模型做不好 agent 任务，容易被解释为“模型太笨”。但课程要强调的是另一种解释：模型有时已经知道任务的大致答案，却不知道应该先观察环境、读取文件、调用验证脚本，或把结果写回正确位置。也就是说，失败不一定发生在“知识”层，而可能发生在“行动协议”层。

本讲的核心定义

一个现代 AI Agent 可以看成两部分：语言模型本体，以及让语言模型能完成任务的 Harness。语言模型提供推理、生成和泛化能力；Harness 决定模型看见什么、能做什么、按什么流程做、如何接收反馈、什么条件下算完成。

如果用一个简化公式表示：

$$\text{Agent} = \text{LLM}_{\theta} + \text{Harness}_{h}$$

- θ：语言模型参数，通常由预训练、微调或强化学习得到。
- h：Harness 的状态，包括 prompt、规则文件、工具接口、权限、memory、workflow、评估器和执行环境。

[2] 视频画面时间区间：00:00:08–00:00:20。

- Agent 的实际能力不是只由 θ 决定，而是由 θ 与 h 的配合决定。

这也是为什么本讲应放在 self-evolving agents 主题里。Self-correction 讨论模型能否改一个答案；self-improving 讨论模型能否产生训练信号；Harness Engineering 讨论的是：当 agent 的外壳本身可以被设计、替换、优化，甚至由另一个 agent 自动修改时，自进化系统到底在改什么。

1.1 和 Prompt Engineering、Context Engineering 的关系

课程把三个概念放在同一条演化线上：

概念	主要问题	局限或进一步发展
Prompt Engineering	同一个问题怎样问，模型输出会更好。	随着模型变强，简单咒语如 “Think step by step” 的边际作用下降。
Context Engineering	模型是否拿到了足够、合适、可用的上下文。	它解决“一问一答”里信息不足的问题，但还不完整描述多轮行动。
Harness Engineering	如何驾驭模型在多轮交互、工具调用、验证反馈中完成任务。	它把 prompt、context、工具、权限、工作流和评估都纳入同一个执行系统。

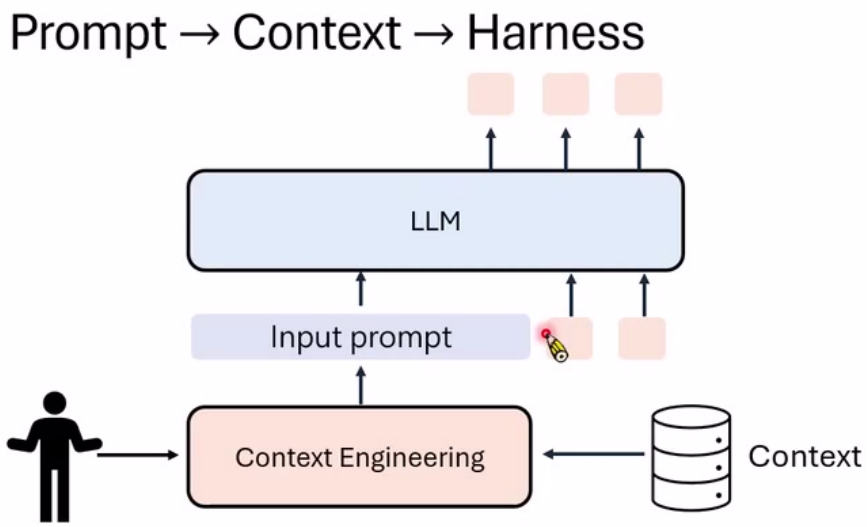

图 2: 从 Prompt Engineering 过渡到 Context Engineering：错误不一定来自模型能力不足，也可能来自上下文信息不足。[4]

[4]视频画面时间区间：00:15:08–00:15:40。

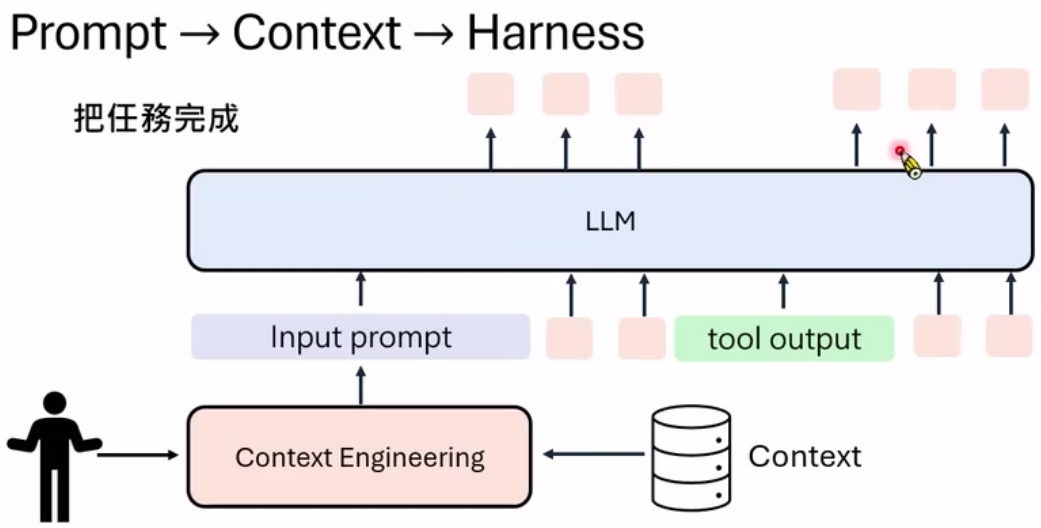

图 3: Harness Engineering 强调的不是单轮问答，而是多轮互动、工具输出和最终任务完成。[5]

为什么 Harness 是更大的概念

Context Engineering 仍然主要问“该把什么信息放进输入”。Harness Engineering 进一步问：模型输出之后发生什么？这个输出是否会触发工具？工具返回的信息怎样进入下一轮？模型能不能修改文件？验证失败以后怎样重试？什么时候停止？这些问题构成 agent 的真实行为边界。

1.2 本章小结

本讲不是在给 prompt 技巧换一个新名字。它要讲的是 agent 能力的工程性来源：同一颗语言模型，放在不同规则、工具、记忆和工作流里，会表现得像不同的系统。对 self-evolving agents 来说，这意味着“自我改进”不只可能发生在模型参数中，也可能发生在 Harness 中。

2 开场实验：小模型是真的不会，还是没有被引导

讲者用 Gemma 4 E2B 做了一个代码修复实验。任务大意是：当前目录里有 ‘Parser.py‘ 和 ‘Verify.py‘，‘Parser.py‘ 里的 email 解析函数有 bug，模型需要修改文件，并最终让 ‘Verify.py‘ 的测试通过。

[5]视频画面时间区间：00:16:00–00:16:32。

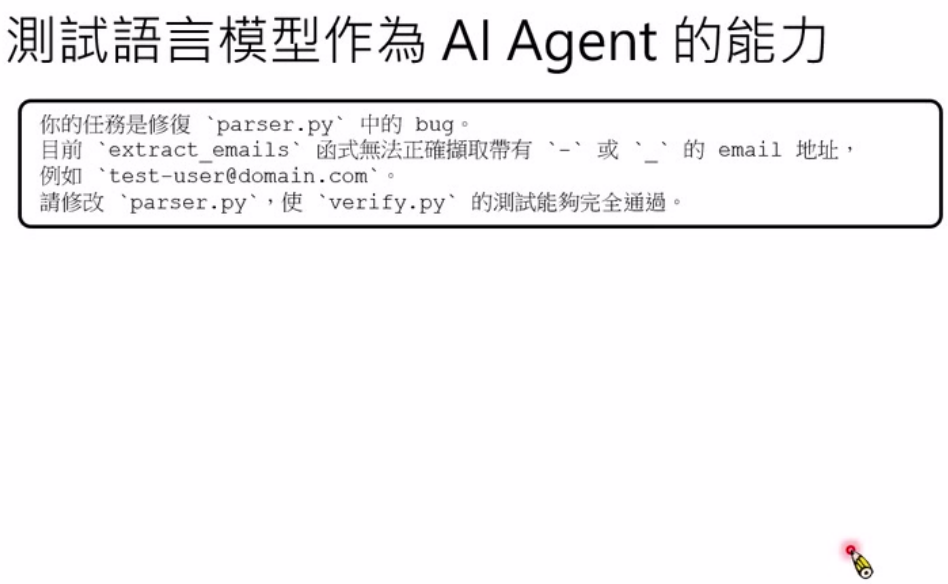

图 4: Gemma 4 E2B 被要求修复 ‘Parser.py‘，并通过 ‘Verify.py‘ 验证。[6]

为了让模型能像 agent 一样行动，讲者给它配置了基本工具协议：如果模型在三反引号中输出 ‘bash‘，环境就执行那行 shell 指令；如果输出 Python 代码，环境就把代码写入文件并执行。这意味着模型理论上已经能列目录、读文件、改文件、跑测试。

第一次实验中，模型没有先列目录，也没有打开 ‘Parser.py‘。它看到任务文字里提到 ‘Parser.py‘，却没有看到文件内容，于是认为“用户没有提供文件”。接着它幻想出一个 ‘Parser.py‘ 的内容，写出一个它认为合理的 email parser，并幻想自己已经验证完成。

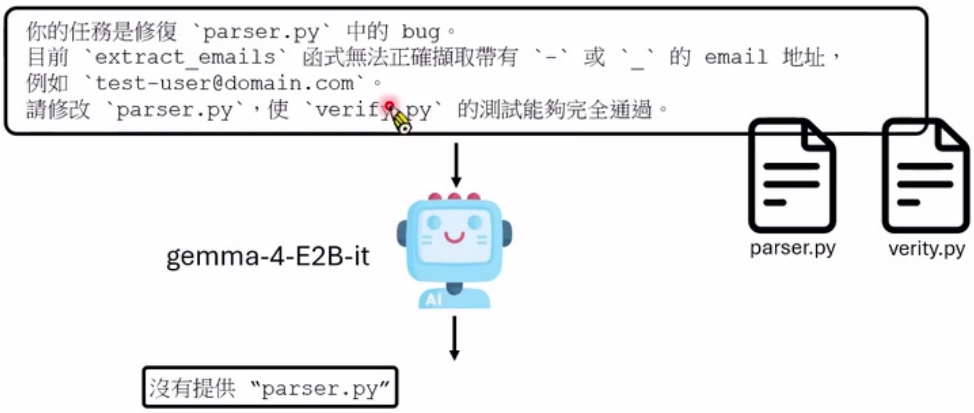

图 5: 没有额外规则时，模型误以为用户没有提供 ‘Parser.py‘，于是开始幻想文件内容。[7]

[6]视频画面时间区间：00:01:10–00:02:10。

这不是纯粹的知识失败

模型能写出一个可能解析 email 的函数，说明它不是完全不会写代码。真正的问题是：它没有形成“先观察当前目录，再读取相关文件，再修改，再验证”的行动程序。它缺的不是 email 解析知识，而是任务环境中的行动纪律。

第二次实验只增加了一小段通用规则，不针对具体 bug 提供答案。这段规则包括：

- 你处在 Linux 环境中，可以使用 shell。
- 做任何事之前，先看看当前目录里有什么。
- 如果要修改文件，先打开文件看看内容。
- 完成意味着达到题目给定的具体标准，例如验证脚本通过。

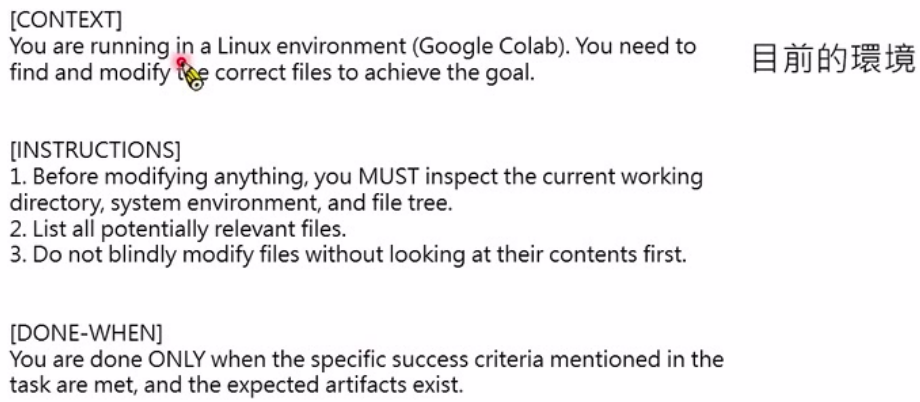

图 6: 只增加少量通用工作原则后，同一个小模型开始先 ‘ls‘、再 ‘cat Parser.py‘、再修改和验证。[9]

这次模型先执行 ‘ls‘，看到 ‘Parser.py‘ 和 ‘Verify.py‘；然后执行 ‘cat Parser.py‘，读取真实文件内容；接着重写 ‘Parser.py‘；最后运行 ‘Verify.py‘ 并看到成功信号。

[7]视频画面时间区间：00:03:15–00:04:20。

[9]视频画面时间区间：00:05:08–00:06:20。

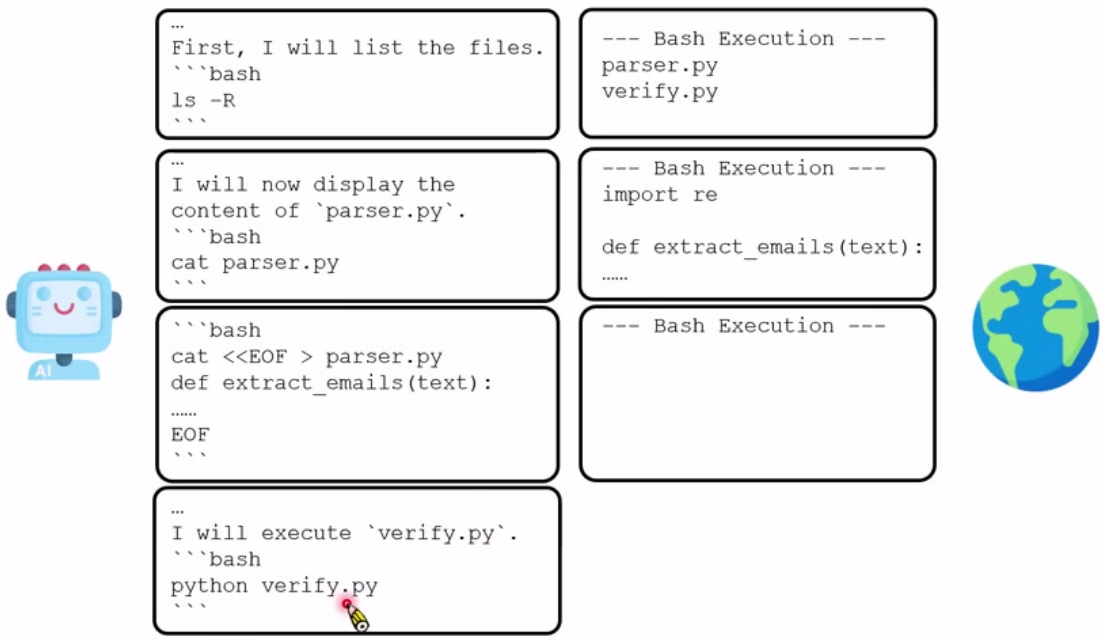

图 7: 模型运行验证脚本，看到 'Verify Success' 后才结束任务。[10]

实验给出的最小结论

同一个语言模型，加入不到一百字的行动规则，表现可能从“幻想自己完成任务”变成“读取文件、修改文件、运行测试”。因此，当 agent 失败时，不能只问要不要换更大模型，也要问 Harness 是否让模型知道该怎么行动。

2.1 完成标准是 Harness 的一部分

实验里最关键的规则之一是“什么叫完成”。如果模型只被要求“修复 bug”，它可能输出一段看似合理的代码就结束；如果 Harness 明确告诉它“通过 'Verify.py' 才算完成”，模型就有了停止条件。许多 agent 的失败并不是做不了中间步骤，而是没有把任务目标转化为可验证终止条件。

这对应软件工程里的一个朴素经验：没有验收标准，执行者就会自定义“完成”。语言模型尤其容易这样做，因为它擅长生成一段看似完整的解释。Harness Engineering 的作用之一，就是把“看似完成”压缩成“经验证完成”。

2.2 本章小结

开场实验说明 Harness 的作用可以非常低成本：几行规则就能改变 agent 的行动轨迹。但这不意味着规则总是越多越好。真正有用的是让模型知道环境、工具、文件、验证和停止条件，而不是把所有可能知识都塞进 prompt。

3 AI Agent = LLM + Harness

课程随后正式给出概念框架。AI Agent 包含语言模型，也包含一系列支撑语言模型完成任务的程序、工具和约束。过去这些“其他东西”没有统一名字；现在越来越多人把它们称为 Harness。

[10] 视频画面时间区间：00:07:40–00:08:05。

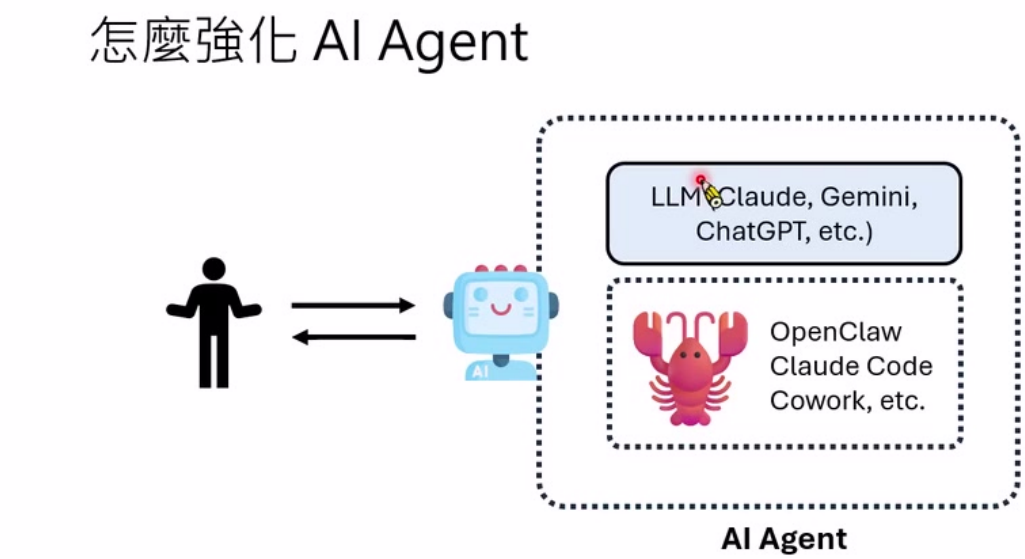

图 8: Agent 的两部分：Large Language Model 与围绕它的 Harness。[11]

讲者用 Claude / OpenCode / Claude Code / Cowork 等例子说明：同一个底层模型，接在不同 Harness 上，行为并不相同。Harness 会决定模型是否能读本地文件、是否能自动挂载目录、能不能操作浏览器、是否需要人类授权、规则文件叫什么、记忆放在哪里。

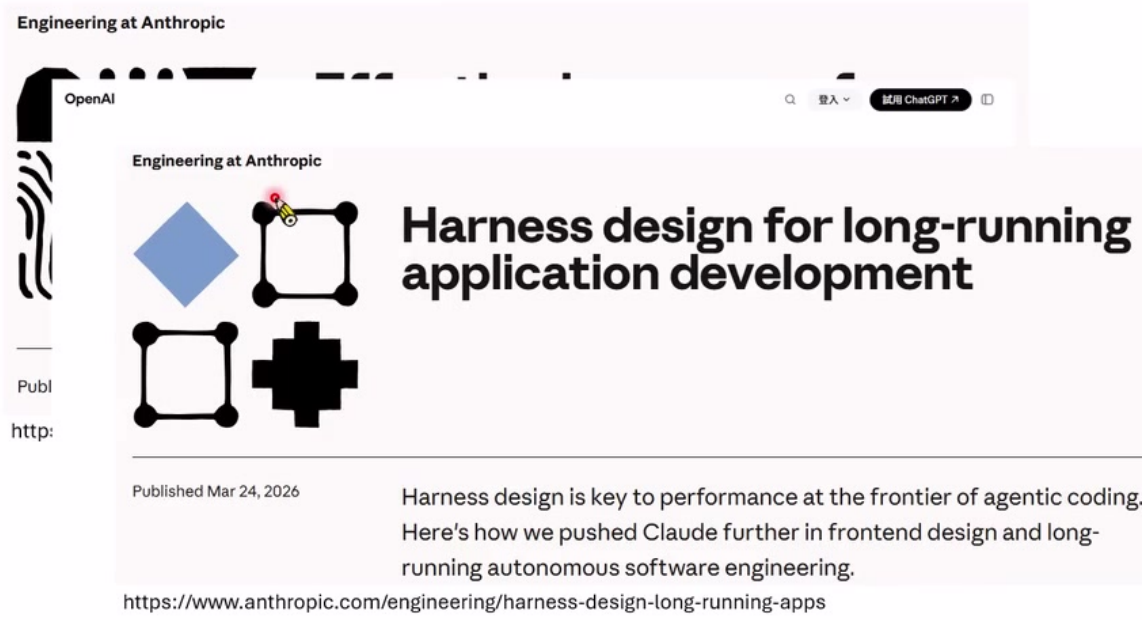

图 9: Harness Engineering / Harness Design 已经成为大公司博客和工程讨论中的热门主题。[12]

课程给出的直观比喻是：AI 像一匹马，有力量，但需要马具来驾驭。这个比喻有两个层面：

- Harness 不是让模型变成另一个模型，而是让已有能力可控、可用、可验证。

[11] 视频画面时间区间：00:08:20–00:09:25。

[12] 视频画面时间区间：00:12:50–00:13:15。

- Harness 不是单一提示词，而是人类和模型之间的一套行动接口。

为什么“换 Harness”有时像换系统

模型参数可能没变，但 Harness 改了，模型看到的世界就变了：能读哪些文件、工具返回什么格式、是否有 memory、是否被强制验证、是否有安全确认、是否能联网或操作浏览器。这些差异足以让同一个模型表现出不同的能力边界。

3.1 三个控制对象

讲者把 Harness 的控制手段分成三个代表性方向：

1. 认知框架：用自然语言规则、系统提示、AGENTS.md 等影响模型思考方式。
2. 能力边界：通过工具和权限决定模型能做什么、不能做什么。
3. 行为流程：通过标准工作流让模型按计划、生成、评估、修正等步骤行动。

图 10: Harness 可以通过语言规则、工具限制和工作流控制模型的认知框架、能力边界与行为。[13]

这三者彼此重叠。一个规则文件可能告诉模型怎样使用工具；一个工具可能强迫模型按某种格式输出；一个工作流可能把规则、工具和评估器组合在一起。课程把它们分开，是为了说明 Harness 不只是 prompt。

3.2 本章小结

Agent 的能力来自 LLM 与 Harness 的组合。换模型当然重要，但如果 Harness 没有让模型看见正确文件、调用正确工具、获得正确反馈、执行正确工作流，那么更大的模型也可能浪费在错误路径上。

[13] 视频画面时间区间：00:17:10–00:17:35。

4 认知框架：AGENTS.md、CLAUDE.md 与自然语言规则

第一类 Harness 是自然语言规则。典型形式是 ‘AGENTS.md‘、‘CLAUDE.md‘、系统提示、项目说明、工具使用原则等。这些规则通常会被 Harness 在对话开始前读入 prompt，使模型在执行任务前先接受一套行为约定。

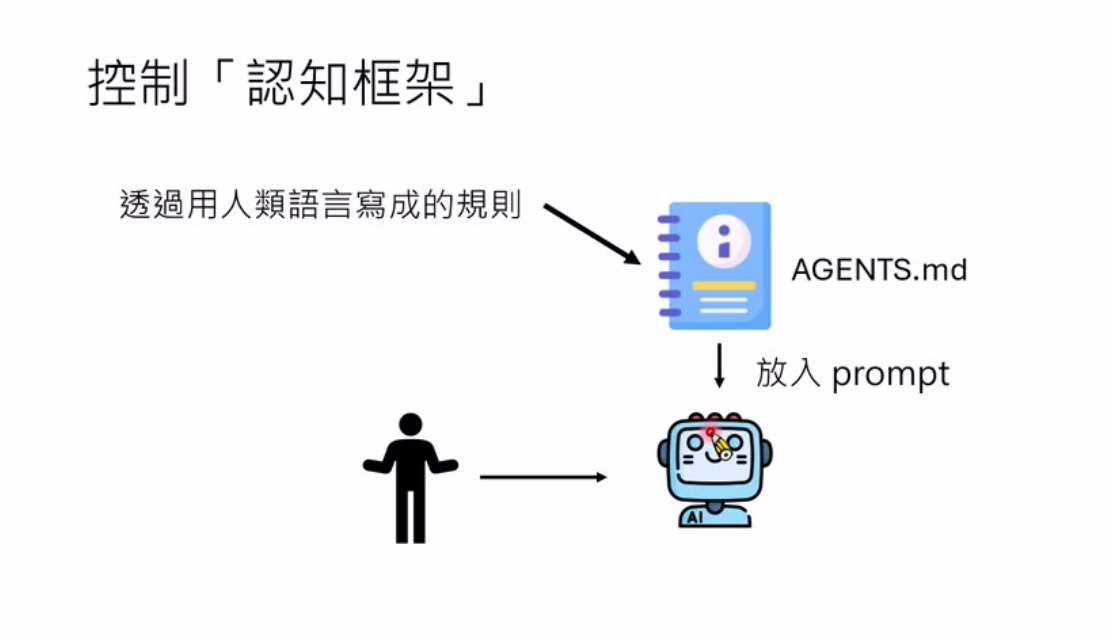

图 11: 自然语言规则文件像“法律”：每次任务前都进入 prompt，影响模型行为。[14]

课程以 OpenCode 为例：工作区里可以放 ‘AGENTS.md‘，里面描述模型的身份、行为规则、memory 放在哪里、怎样查旧记忆、什么文件是灵魂之类。OpenCode 启动时会把这些规则放进 prompt。Claude Code / Cowork 也有类似机制，只是默认文件名可能不同，例如 ‘CLAUDE.md‘。

[14]视频画面时间区间：00:18:35–00:19:10。

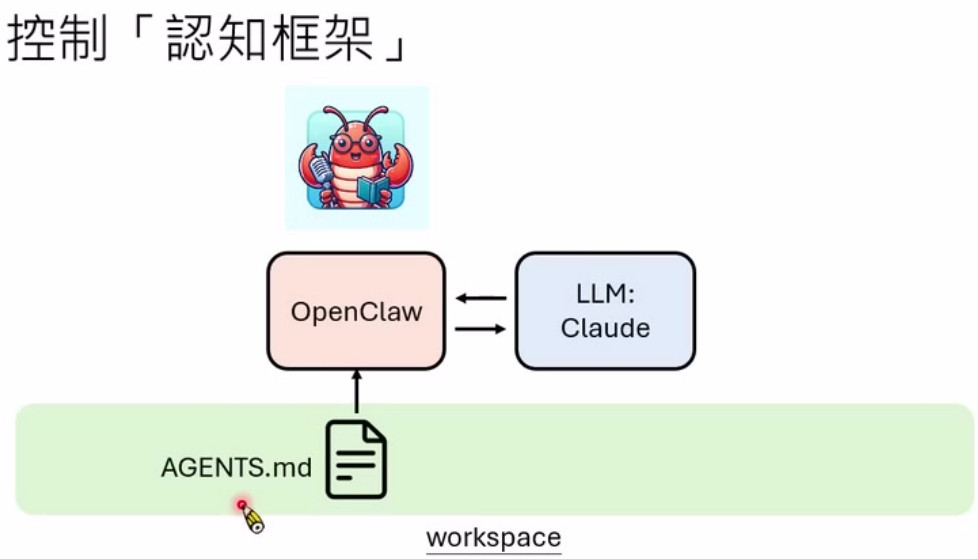

图 12: OpenCode / Claude Code 一类 Harness 会在工作区读取规则文件，把它作为模型行动前的上下文。[16]

讲者还举了一个迁移例子:如果一个 agent 原来运行在 OpenCode 上,规则集中在 'AGENTS.md',现在因为服务商限制不能继续用某个模型,你想迁移到 Claude Code / Cowork 之类 Harness,那么核心动作可能只是把同一工作区给新 Harness,并把规则文件改成它预期的文件名。只要理解 Harness 的读取机制，迁移不必神秘。

自然语言规则不是强制执行

把规则放进 prompt 不代表模型百分之百遵守。它更像法律或组织规范：有约束力，但不是硬权限。真正的强制约束通常来自工具权限、沙盒、文件系统权限、运行时检查和人工审批。

4.1 AGENTS.md 的效果开始被系统研究

过去很多人凭直觉写规则文件,但没有系统评估。课程提到近期已有论文开始研究 'AGENTS.md' 对 agent 行为的影响。例如，有研究从 GitHub 搜集含 'AGENTS.md' 的项目，比较有规则文件和没有规则文件时，模型完成任务所需时间和 token。结果显示规则文件可能帮助减少极端长耗时任务，但这类研究未必能评估任务正确率，因为开源项目的真实验收条件未必清楚。

[16]视频画面时间区间：00:20:15–00:21:10。

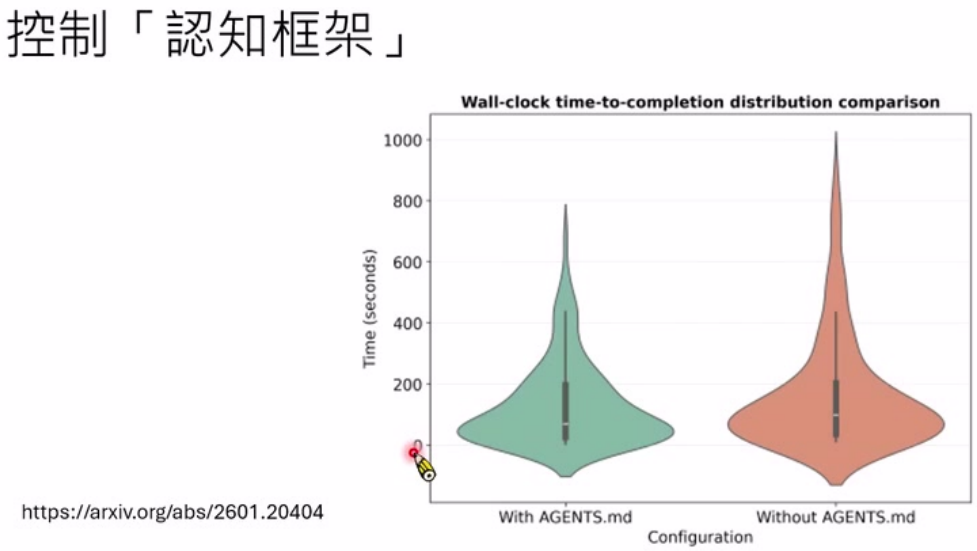

图 13: 研究开始把 ‘AGENTS.md‘ 当作可做消融的 Harness 组件，而不是只凭直觉使用。[17]

另一个研究更直接比较正确率：无 ‘AGENTS.md‘、人类写的 ‘AGENTS.md‘、LLM 自己写的 ‘AGENTS.md‘。课程强调一个令人警惕的发现：人类写的规则文件并不总是提升效果；LLM 自己写的规则文件更常变差，甚至不如没有规则文件。

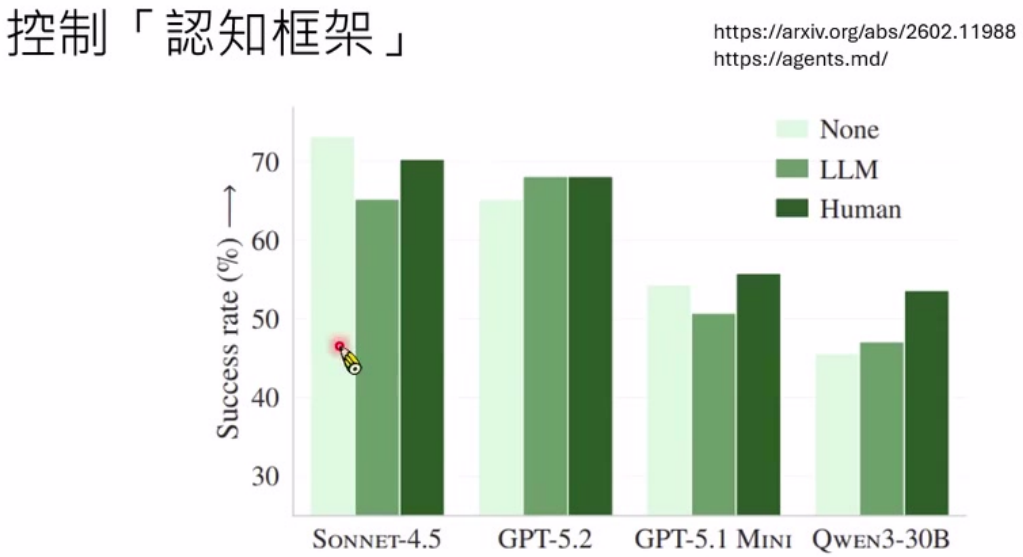

图 14: 不同模型和任务上，‘AGENTS.md‘ 不一定总能提升正确率；LLM 自己写的规则文件尤其不稳。[19]

[17]视频画面时间区间：00:23:50–00:24:20。

[19]视频画面时间区间：00:25:20–00:26:15。

规则文件的设计原则

好的 ‘AGENTS.md‘ 应该像地图，而不是百科全书。它要告诉模型“如果需要某类信息，去哪里找；如果要修改，按什么流程；如果要验证，用什么标准”，而不是把所有知识塞进上下文。

OpenAI 的相关博客也提到：规则文件太长会占据大量 context，让模型还没开始做任务就背负一本“六法全书”。如果每次行动前都要把百科全书放进 prompt，模型剩余上下文减少，注意力也会被稀释。

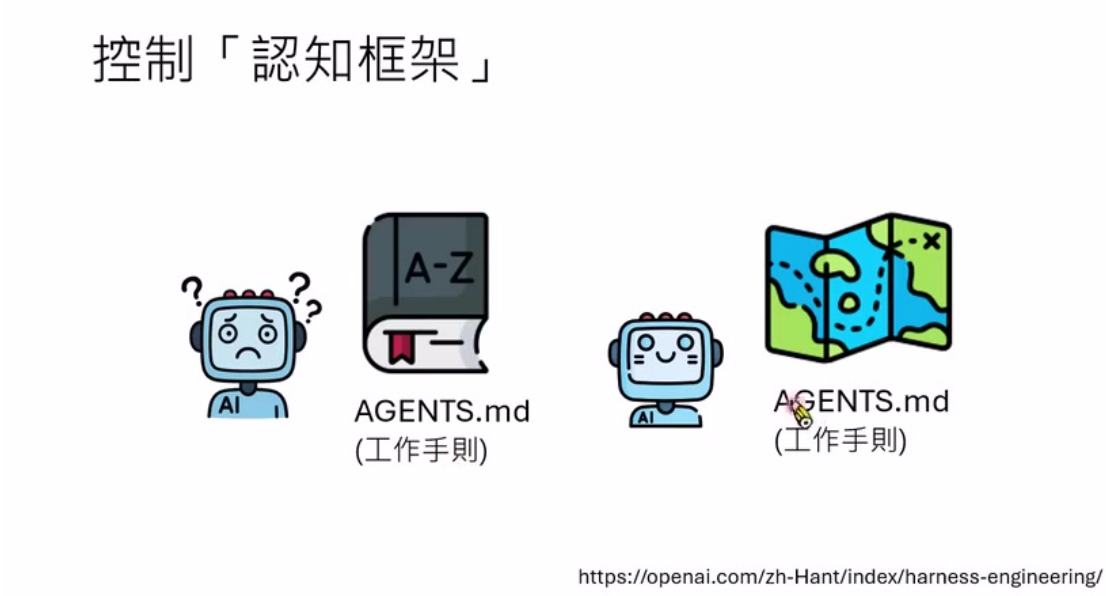

图 15: 规则文件应像地图：指明信息入口和行动原则，而不是把所有内容塞进 prompt。[20]

4.2 本章小结

自然语言规则是 Harness 的一部分，但不是魔法。它能给模型建立行动框架，却不能替代硬权限、工具设计和验证流程。规则文件要短、可操作、指向真实资源，并明确完成标准；否则它可能只是消耗上下文。

5 能力边界：工具、权限与 Agent-Computer Interface

第二类 Harness 是工具和权限。工具决定模型能不能读文件、改文件、搜索、运行代码、打开浏览器、上传视频。权限决定工具能否自动执行，还是需要人类批准。课程反复强调：当模型说“我做不到”时，未必是模型本体做不到，可能是 Harness 没给它相应工具。

[20] 视频画面时间区间：00:26:50–00:27:50。

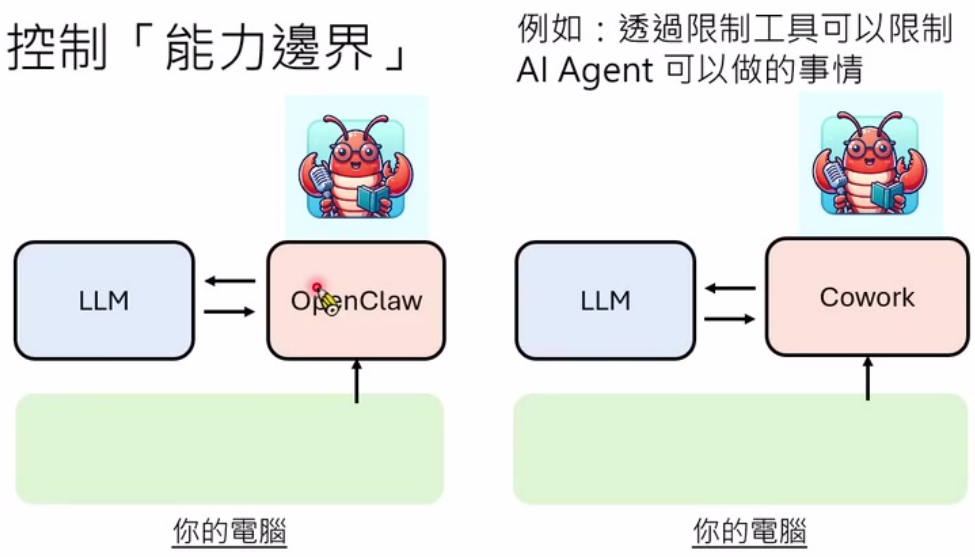

图 16: 不同 Harness 暴露的工具和权限不同，同一个模型会呈现不同能力边界。[21]

讲者比较 OpenCode 与 Cowork / Claude Code 这类环境。OpenCode 运行在本地电脑上，可以直接看本地文件、修改本地文件、操作浏览器；Cowork 运行在云端沙盒中，需要用户显式挂载本地目录，安全性更高，但便利性下降。安全和便利构成 trade-off：越安全，模型越不容易越权；越方便，模型越可能直接行动。

“模型不会”与“工具不允许”要分开

如果云端 Harness 不允许上传 YouTube 视频，模型可能回答“由于安全限制我不能上传”。这不是模型不知道怎么上传，而是工具和权限边界不允许。评估 agent 能力时必须区分模型能力、工具能力和权限策略。

5.1 SWE-agent 与 ACI：给模型趁手工具

课程引用 SWE-agent 的早期思想：当时 Harness Engineering 这个词还没有那么流行，相关设计被称为 Agent-Computer Interface，简称 ACI。它研究的是：给 agent 什么样的计算机接口，才能让它更好地做软件工程。

[21] 视频画面时间区间：00:28:00–00:29:10。

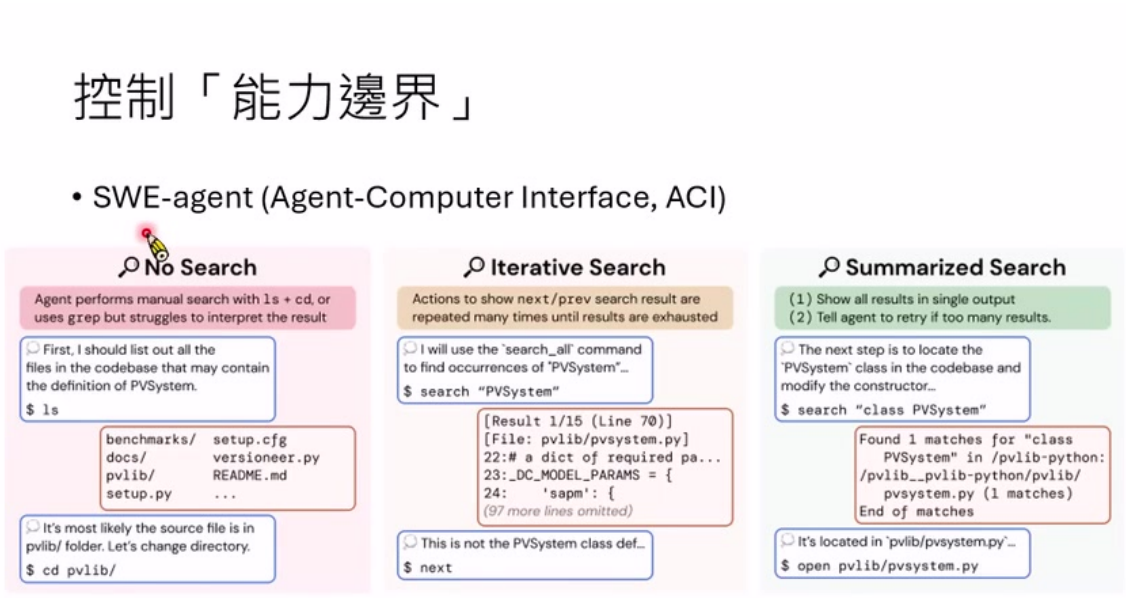

图 17: SWE-agent 把工具界面称为 Agent-Computer Interface；这可以看作早期 Harness 设计。[22]

一个例子是搜索工具。不给搜索工具时，模型只能用 ‘ls‘、‘grep‘ 等原生命令找文件；给一个像人类搜索引擎一样分页的工具，模型可能不断翻页，把 context 塞满；给一个摘要式搜索工具，只返回相关文件名、路径和摘要，再让模型自己打开文件，效果反而更好。

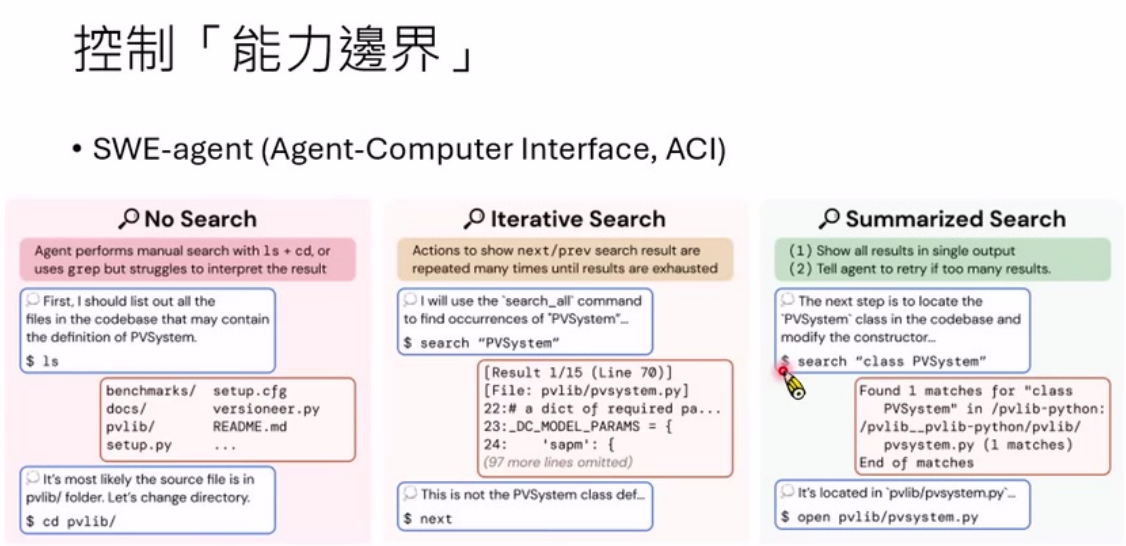

图 18: 三种搜索接口：无搜索、分页式搜索、摘要式搜索。适合人的接口不一定适合 agent。[23]

另一个例子是编辑工具。给模型一个按行号替换的 edit 工具，直觉上应该比 ‘cat‘、‘sed‘、重写文件更方便。但如果模型只看到局部片段，可能不知道上下文已有括号，于是多写括号造成语法错

[22]视频画面时间区间：00:31:35–00:32:05。

[23]视频画面时间区间：00:32:30–00:34:15。

误。解决方法是配套 linting 工具：每次修改后自动检查语法，把错误反馈给模型，再让它修正。

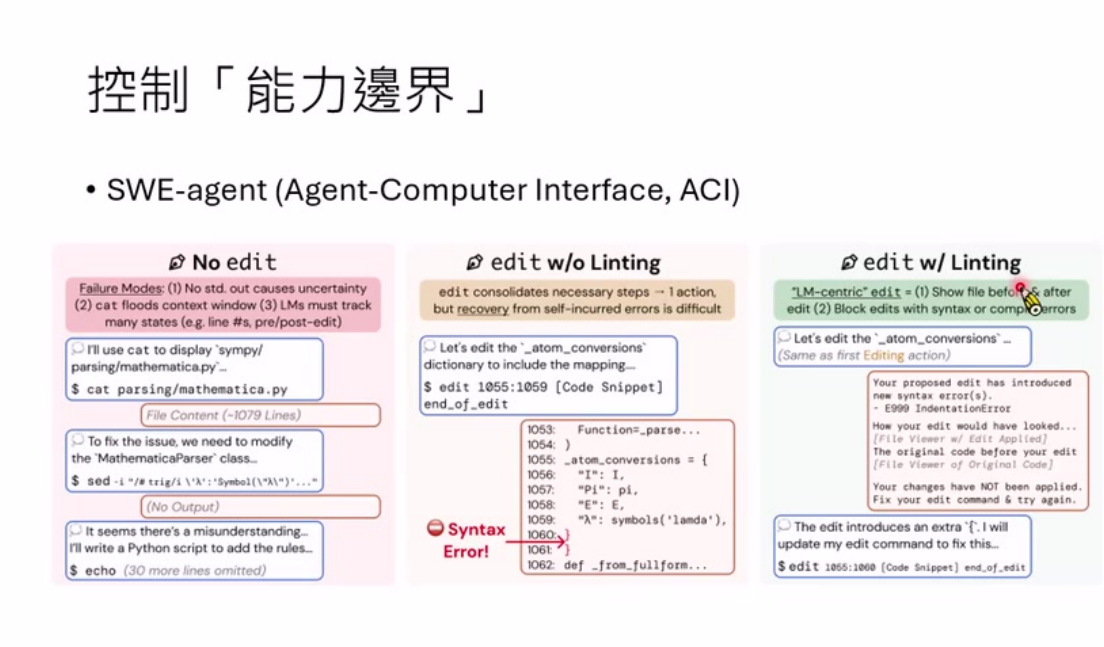

图 19: 编辑工具必须和 linting / verification 搭配，否则局部编辑反而可能制造语法错误。[24]

工具不是越像人类工具越好

人喜欢 GUI、分页搜索和 flags；模型更擅长生成结构化文本、JSON、命令行和可解析协议。Agent-first 的工具设计应从模型的生成能力出发，而不是把人类界面直接丢给模型。

5.2 Agent-first CLI

课程提到，有工程师重写 Google Workspace CLI 时强调 Agent-first：CLI 不是先给人类用、再让 agent 勉强使用，而是一开始就以 agent 为主要使用者。人类可能不喜欢复杂 JSON；模型却很擅长输出结构化 JSON，因此面向 agent 的命令行可以更直接支持结构化输入。

[24] 视频画面时间区间：00:34:20–00:35:50。

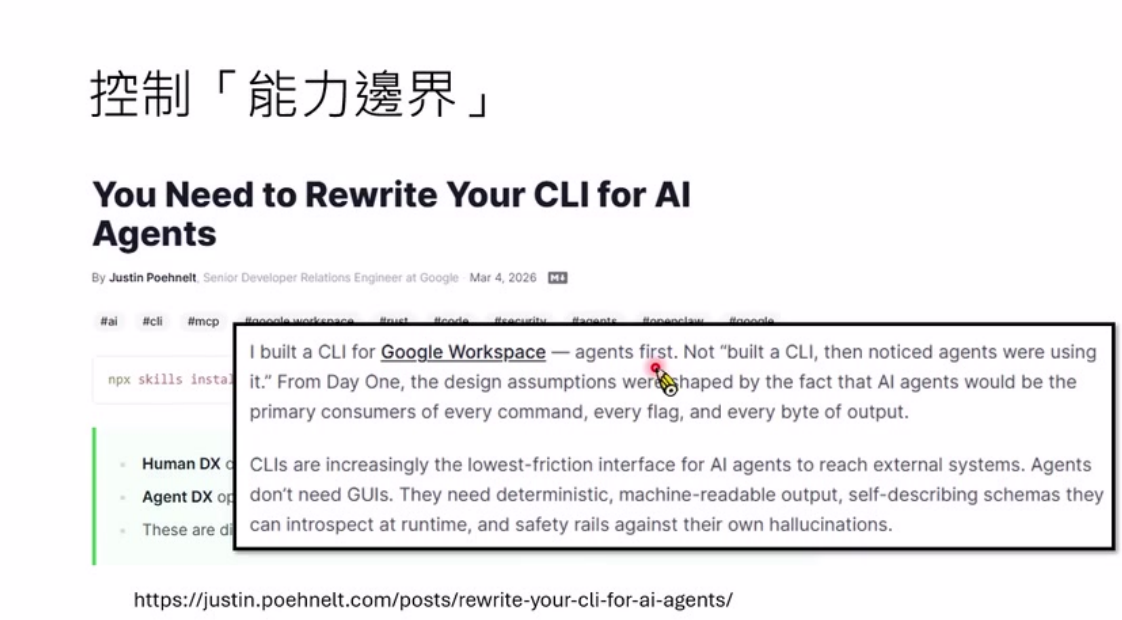

图 20: Agent-first CLI 的例子：模型偏好可结构化、可解析、可组合的工具接口。[25]

这给未来服务设计一个很重要的启示: 当 AI Agent 成为主要操作者, 很多软件接口会从 human-first 转向 agent-first。届时，设计 GUI 不一定是主要工作；设计清晰、可验证、错误信息友好的机器接口，反而更关键。

5.3 本章小结

工具和权限是 Harness 中最硬的一层。规则文件能建议模型做什么，工具接口决定模型实际上能做什么。好的工具要给足能力、返回清楚反馈、避免污染上下文、可验证、可恢复；坏工具即使功能强，也可能让模型走入低效或错误路径。

6 行为流程：Plan、Generate、Evaluate、Revise

第三类 Harness 是标准工作流。与其让模型在一个长 prompt 里自由发挥，不如把任务拆成角色和阶段：planner 规划，generator 执行，evaluator 检查，revisor 修正。大公司博客里大量讨论这类 workflow。

[25]视频画面时间区间：00:36:50–00:38:20。

图 21: Anthropic Harness Design 中常见的工作流：规划、生成、评估。[26]

课程解释 planner-generator-evaluator 的直觉：模型自回归生成时，一旦前面写错，后面会顺着错误继续写，很难回头。让 evaluator 在生成后检查，等于给模型一个停下来审视错误的机会。即使 generator 和 evaluator 背后调用的是同一个模型，只要角色、输入和目标不同，也可能得到更好的结果。

课程还提到一种更细的流程：generator 在开始工作前先向 evaluator 提案，evaluator 接受 contract 后 generator 再执行。这样可以避免 generator 做完后才发现 evaluator 的验收标准不同，导致大幅返工。

为什么同一个模型可以扮演不同角色

角色不是改变参数，而是改变上下文和目标。作为 generator 时，模型被要求产出候选方案；作为 evaluator 时，模型被要求检查标准、找错误、给反馈。这种角色分离本身就是 Harness。

6.1 AI Scientist 的相似流程

DeepMind 的 AI Scientist 工作流也有类似结构：generator 提出方案，verifier 检查方案，如果太差就回到 generator；如果尚可，则进入 revisor 对方案细修。这说明生成-验证-修正已经成为 agent 系统里非常常见的模式。

[26] 视频画面时间区间：00:38:30–00:40:00。

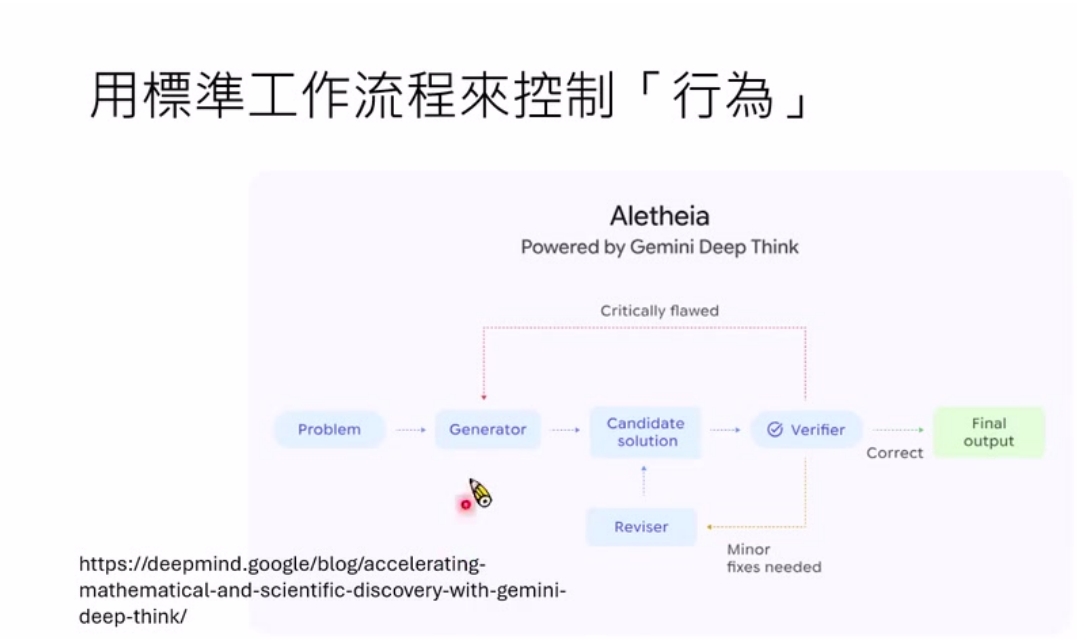

图 22: AI Scientist 工作流也包含 generator、verifier、revisor 等模块。[27]

不过，课程也提醒：一个 workflow 不会对所有模型都最好。例如某些模型有“上下文焦虑”，上下文太长容易表现变差；某些模型可以处理更长上下文，反而不需要每轮都摘要。Harness 必须适配模型，不存在对所有模型、所有任务都通用的万能 Harness。

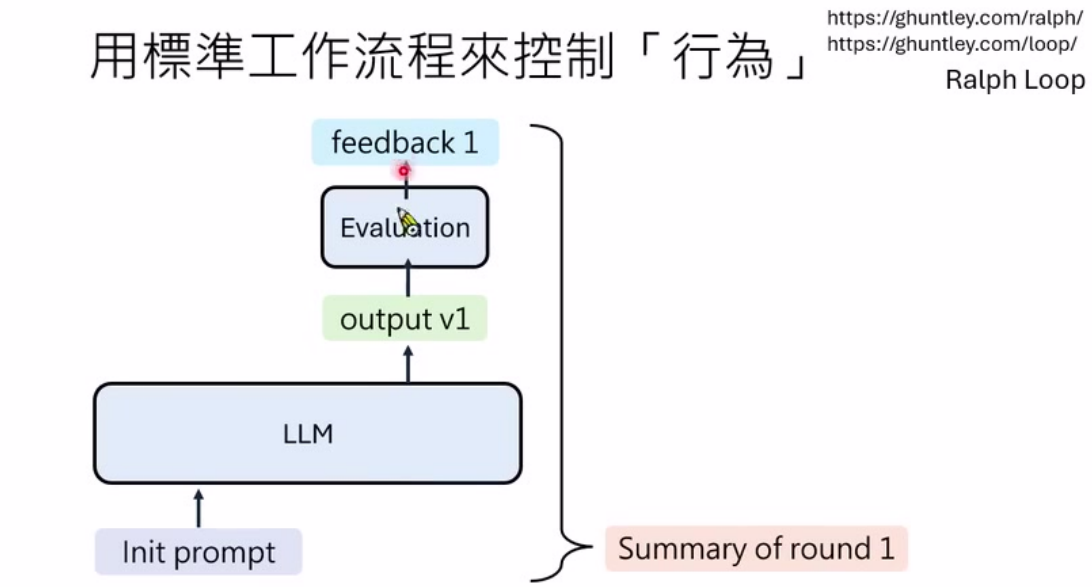

图 23: 多轮 feedback 很快占满 context，因此一些 Harness 会在每轮后摘要，再进入下一轮。[28]

[27]视频画面时间区间：00:41:10–00:42:00。

[28]视频画面时间区间：00:43:30–00:44:10。

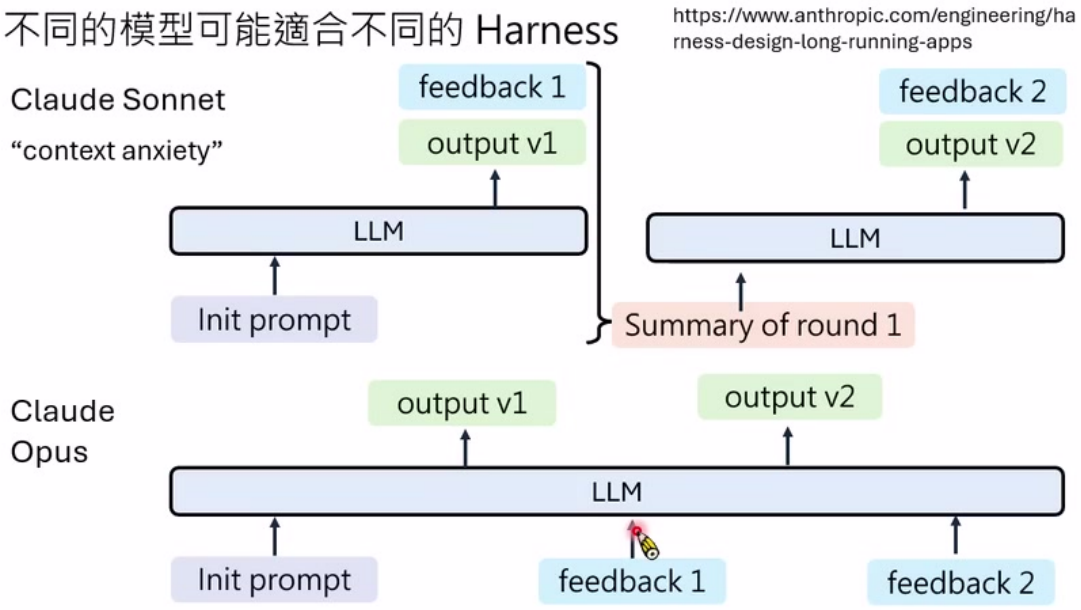

图 24: 课程强调：不要假设存在一个对所有语言模型都最好的万能 Harness。[29]

Workflow 也是假设

Plan-generate-evaluate 很常见，但不是天然最优。它假设任务可拆、评估器能判断、反馈能被 generator 使用。对开放式创作、长程规划或有欺骗风险的系统，这些假设都要被重新检查。

6.2 本章小结

Workflow 把模型从“一次输出”变成“多阶段系统”。它能让模型先计划、再执行、再验证、再修正；也可能带来上下文膨胀、角色误配和评估器偏差。设计 Harness 时，workflow 要和模型能力、任务结构、反馈质量一起考虑。

7 Feedback：把错误变成下一轮行动的输入

课程中段把 workflow 推向 feedback loop。模型做出输出，环境或人类给反馈，模型带着反馈重新输出；这个过程类似一种没有显式梯度的优化。讲者提到，有人把这种 feedback 驱动的行为改变类比为一种 textual gradient 或特殊的 gradient descent。

[29] 视频画面时间区间：00:45:00–00:45:15。

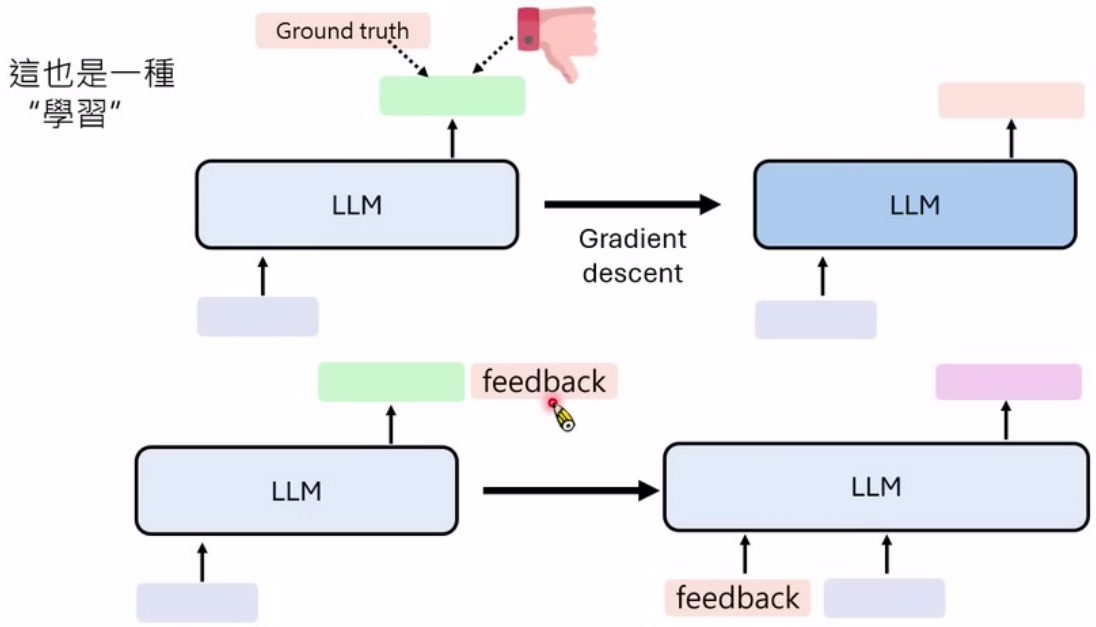

图 25: Feedback loop 的直觉：输出得到反馈，再把反馈放回输入，推动下一轮输出。[30]

这个类比不是说文本反馈真的等于数学梯度，而是说它在功能上扮演类似角色：指出当前输出哪里不好，告诉模型下一步应向哪个方向改变。对于 agent 来说，compiler error、test failure、human preference、rubric score、environment state 都可能成为 feedback。

Feedback 的关键不是“有”，而是“对”

正确反馈能引导模型修正；随机反馈会让模型变差，甚至比没有反馈更糟。因此 Harness 不只要把反馈接回模型，还要判断反馈是否可信、相关、可操作。

课程举了一个模拟动画 agent 的例子。原来的 workflow 是：用户提出需求，program generator 生成程序，环境运行程序并得到结果，再把反馈给 generator。研究者发现，让反馈更完整、更具体，可以显著提升模型最终完成任务的能力。

[30] 视频画面时间区间：00:46:00–00:47:30。

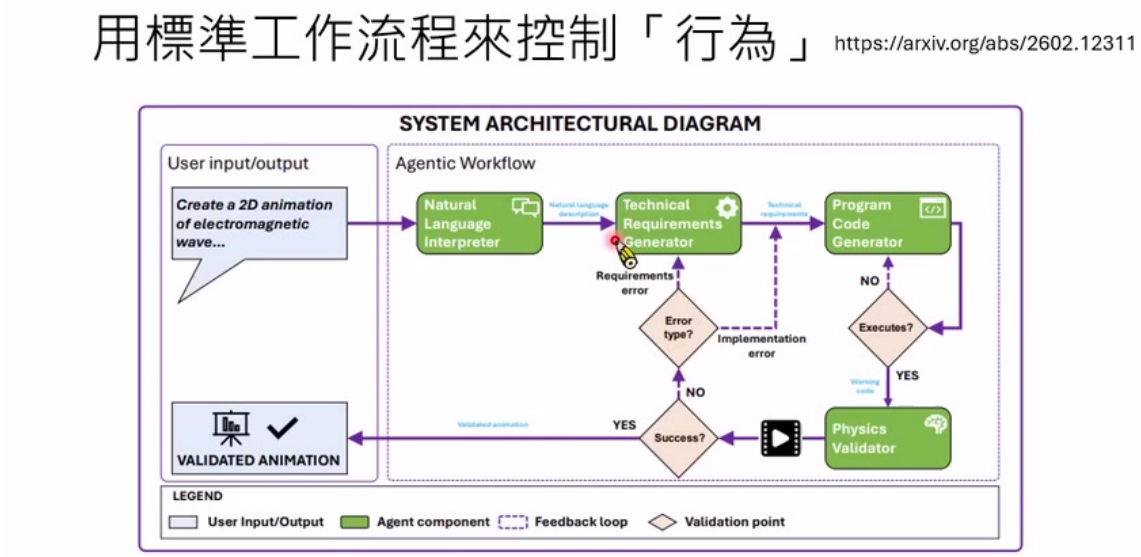

图 26: 生成模拟动画的 agent：程序生成、执行、反馈、再生成。[31]

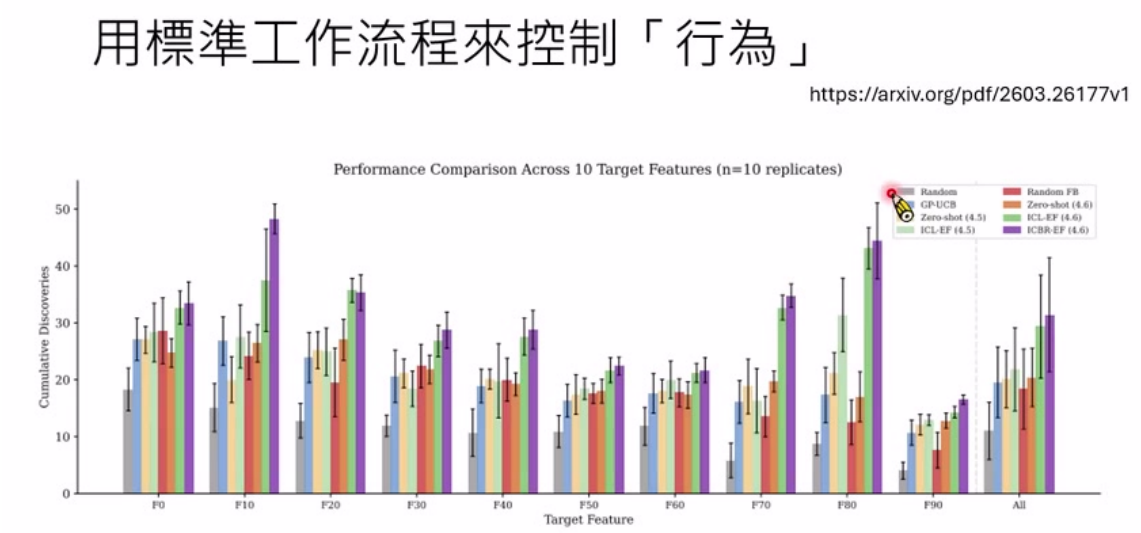

图 27: 反馈质量会直接影响结果；随机反馈甚至会伤害模型行为。[32]

这也解释了为什么 in-context learning 有效：给模型一些正确例子，本质上也是给它关于任务格式、目标和评价标准的反馈。模型看到例子后，输出会更接近所需模式。

[31] 视频画面时间区间：00:48:00–00:49:10。

[32] 视频画面时间区间：00:50:20–00:52:00。

7.1 过度责备可能有害

课程还引用一篇关于 AI Agent “情绪”或行为反应的博客讨论：如果你一直用责备方式给模型反馈，可能并不总是有利。这里不需要把模型拟人化，但要看到工程事实：feedback 的语气、结构、可操作性、是否包含下一步建议，会影响模型后续行为。

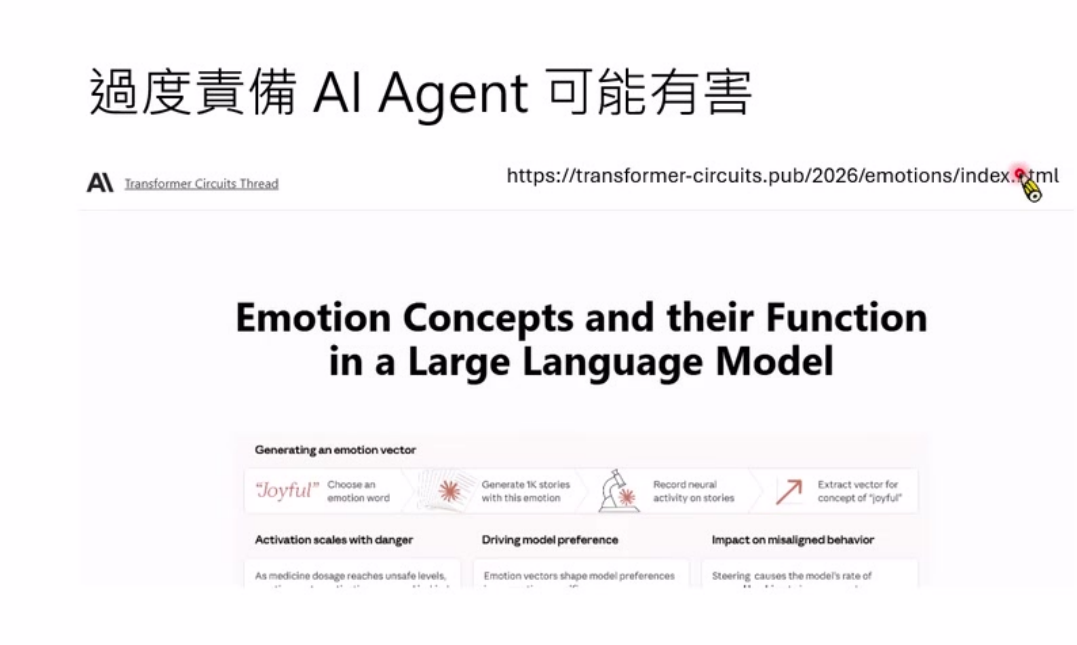

图 28: 课程讨论过度责备 agent 可能带来的负面效果。[33]

对工程实践而言，最稳妥的反馈应具有四个属性：

- 可定位：指出哪个文件、哪一步、哪个断言失败。
- 可执行：说明下一步应该检查什么，而不是只说“你错了”。
- 可信：来自测试、编译器、明确 rubric 或人类真实偏好。
- 不过度：避免把无关偏好、情绪化指责、随机建议混入反馈。

7.2 本章小结

Feedback 是 agent 从一次行动走向持续改进的桥梁。Harness Engineering 要设计的不只是工具调用，还包括反馈的来源、格式、筛选、摘要和重试策略。反馈越接近真实任务目标，agent 越可能改对；反馈越随机或越模糊，越可能带来 drift。

8 Lifelong AI Agent：当 Agent 不再是一次性工具

课程后半段把 Harness Engineering 推向长期 agent。讲者认为，从现在开始，AI Agent 可能不再只是一次性工具，而会变成陪伴人类很久的系统。一次性 agent 可以每次重新开始；lifelong agent 则要保留记忆、整理经验、持续适配用户。

[33]视频画面时间区间：00:53:10–00:54:00。

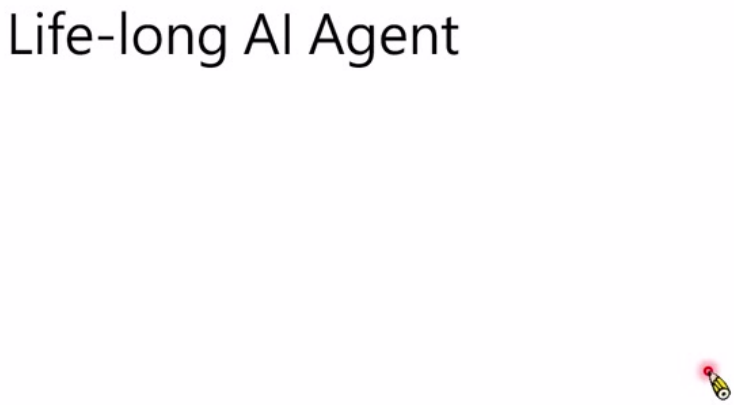

图 29: 课程提出 Lifelong AI Agent 的方向：Agent 不再只是一次性工具。[34]

长期 agent 的核心挑战是 memory。讲者举例说，如果自己的小金 agent 的 memory 没同步到云端，一旦换 Harness 或换环境，就会丢失长期记忆。记忆既是能力来源，也是负担：记忆太乱、太长、太旧，会拖慢模型，污染上下文。

图 30: Claude Code / OpenCode 一类 Harness 中，memory 与规则文件共同构成长程 agent 的行为基础。[36]

因此，长期 agent 需要“睡眠”或整理机制：在用户不使用它时，系统整理过去记忆，压缩、归

[34]视频画面时间区间：01:02:38–01:03:00。
[36]视频画面时间区间：01:04:20–01:04:40。

档、删除不重要信息，保留未来会用到的经验。这像人类睡眠整理记忆，但工程上应理解为 memory compaction、indexing、deduplication 和 retrieval policy 更新。

图 31: 长期 agent 需要整理 memory；否则 memory 会膨胀并影响后续行动。[37]

Lifelong Agent 的 Harness 目标

一次性 Harness 只要让模型完成当前任务；lifelong Harness 要让模型在长期交互中保存有用经验、遗忘无用噪声、迁移到新环境、学习用户偏好，并且不把错误经验固化成未来行为。

8.1 Skill 作为可写入的 Harness 经验

讲者举了一个实际经历：几个小金 agent 没能按时上传视频，后来他要求它们找出战犯。某个 agent 通过排程反复检查上传状态，最终解决问题；更重要的是，它在完成后写下了一个 skill，让未来的自己知道“我有上传视频的能力”。

这个例子说明，agent 的自我改进不一定先发生在模型参数里。它可以先发生在 Harness 层：写一个 skill、更新规则、增加脚本、修改 workflow、记录工具使用方式。这些改变会在未来任务中影响同一个模型的行为。

[37] 视频画面时间区间：01:05:00–01:05:50。

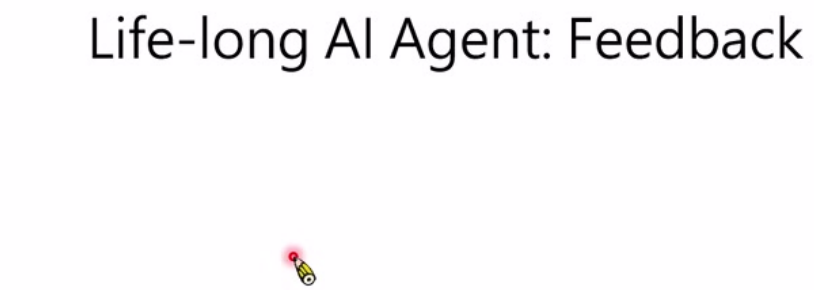

图 32: 长期 agent 的成长来自环境互动与 feedback；skill、memory、workflow 都可能成为可更新对象。[39]

Skill 也是漂移入口

如果 agent 能自己写 skill，它也可能写入错误经验、过拟合某次任务、记录过时环境、扩大权限假设。Self-evolving agents 的关键不是让 skill 自动增多，而是让新增 skill 可审计、可验证、可回滚。

8.2 本章小结

Lifelong agent 把 Harness Engineering 从“提高当前任务成功率”变成“维护长期身份和能力”。Memory、skill、workflow 和反馈记录都可能成为自我成长材料，也都可能成为 drift 来源。

9 从 Verbalized Feedback 到参数更新

只更新 Harness 有上限。讲者接着追问：如果一个 agent 要陪伴人类很久，是否也应更新语言模型参数? 这引出 verbalized feedback 学习：人类或环境用自然语言给出反馈，系统要识别哪些反馈真正可用于学习，并把它们转化为参数更新信号。

[39]视频画面时间区间：01:06:20–01:07:00。

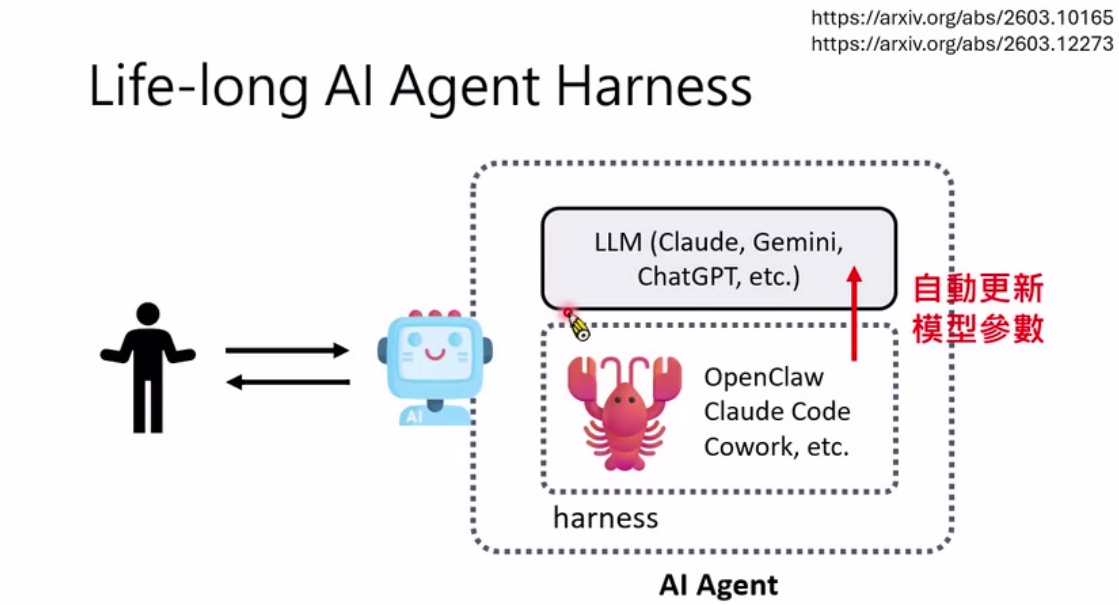

图 33: 课程转向：如何把 verbalized feedback 用于语言模型参数更新。[40]

关键难点是：在一段多轮交互中，哪些句子是 feedback？“任务完成，下一题”不是学习信号；“compile error: missing bracket”显然是反馈；“这个回答太谄媚”也是偏好反馈。模型必须自动区分普通环境输入、下一步任务和真正反馈。

课程介绍了一类做法：把后面一句可能的反馈移到前面，让模型带着“后见之明”重新预测原本输出。如果这句话显著改变原输出 token 的概率，说明它可能包含反馈信号；如果它只是无关新任务，例如问 27 乘 4，对原输出 token 概率影响不大，就不应当当作反馈。

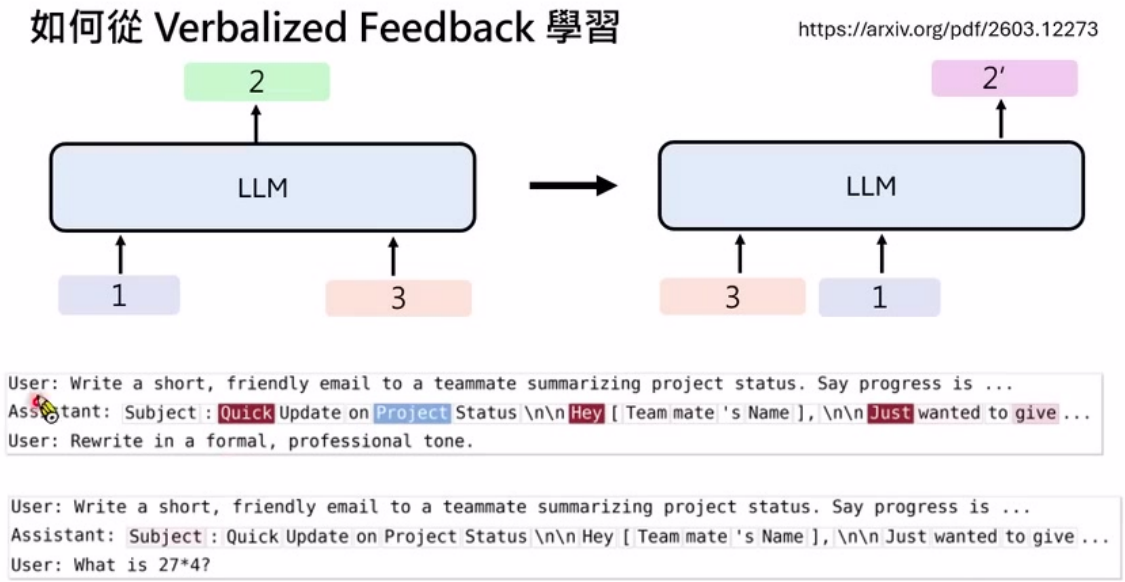

图 34: 通过把后续反馈移到前面，观察 token 概率变化，判断一句话是否真的含有反馈。[41]

[40]视频画面时间区间：01:13:00–01:14:20。

可以抽象为：

$$\Delta_t = \log p_\theta(y_t \mid x, f, y_{<t}) - \log p_\theta(y_t \mid x, y_{<t})$$

- x：原始任务输入。
- y_t：模型在原输出中的第 t 个 token。
- f：后续环境或人类给出的候选反馈。
- Δ_t：加入反馈后，该 token 的对数概率变化。

如果某个反馈让原本错误 token 的概率明显下降，并推动模型生成更好的替代输出，就可以把新输出作为偏好样本或训练目标。不同论文可能采用类似 DPO、SFT 或其他 preference learning 形式。

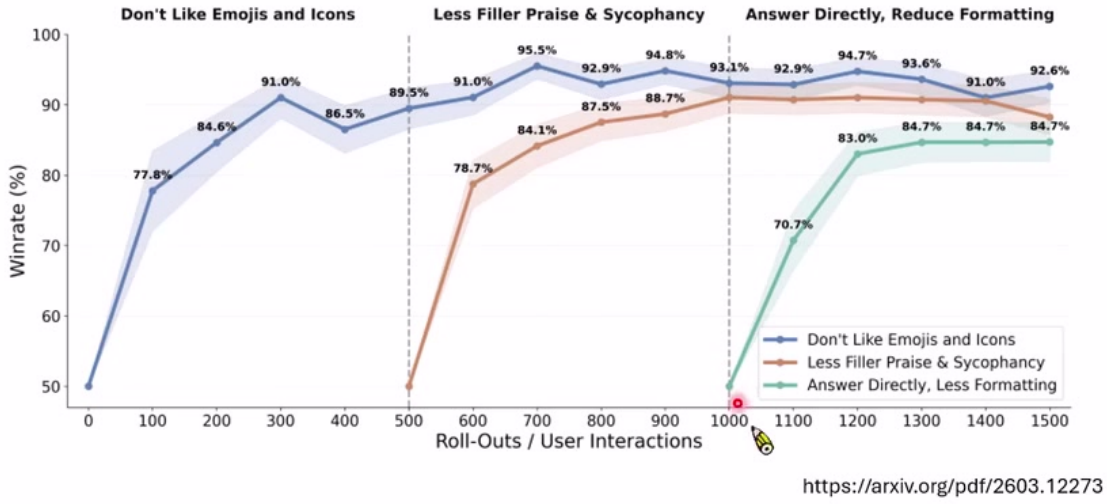

图 35: 持续 verbalized feedback 可以逐段改变模型偏好，例如减少 emoji、减少奉承、提高直接性。[43]

Harness 与参数更新的连接

Harness 负责收集交互、保存反馈、识别反馈、构造训练样本；参数更新负责把这些反馈内化到模型中。长期 agent 真正困难的地方，正是这两层之间的接口。

9.1 评估也会被 AI 代理污染

课程接着讨论 agent benchmark 的难点。以客服类任务为例，真实人类顾客说话简短、模糊、不客气；语言模型扮演的顾客往往过于礼貌、信息完整、表达清晰。因此用 LLM customer 代替真实人类，可能高估 agent 能力。

[41] 视频画面时间区间：01:15:30–01:16:40。

[43] 视频画面时间区间：01:17:15–01:18:40。

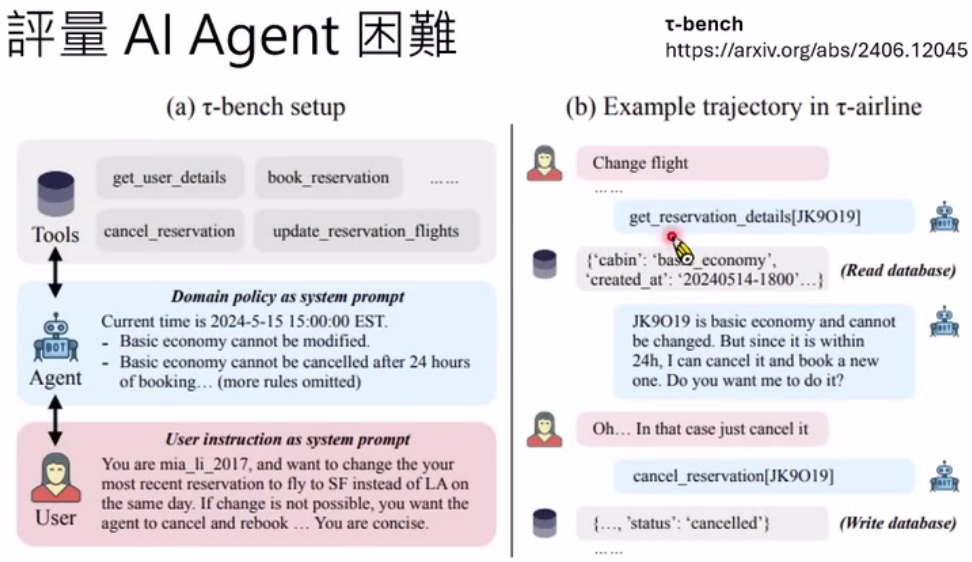

图 36: 客服类 benchmark 中，agent 与 customer 多轮互动以完成订票、退货等任务。[44]

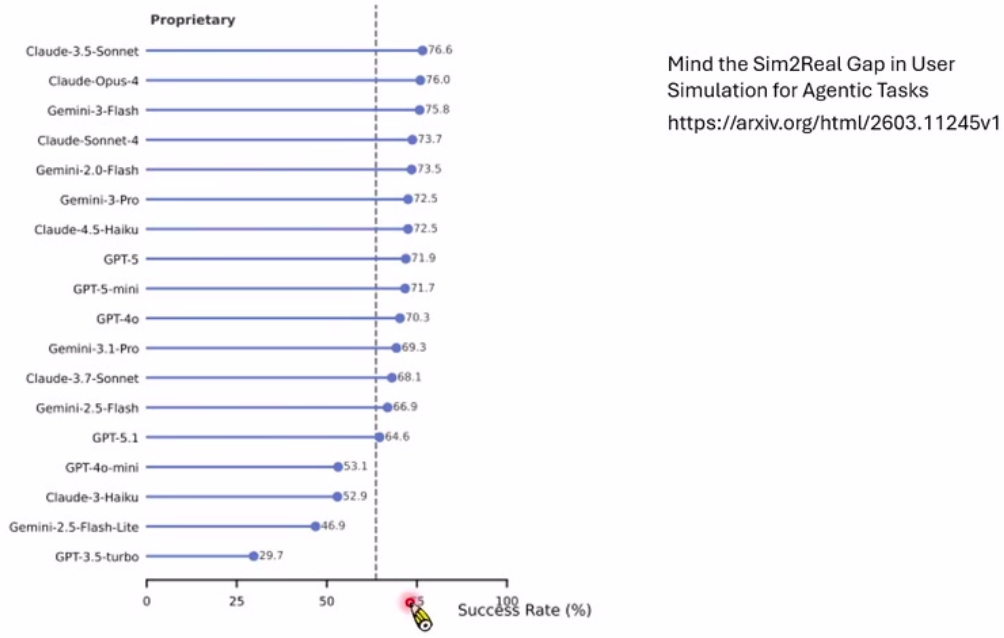

图 37: 当 customer 由不同 LLM 扮演时，任务成功率可能高于真实人类顾客场景。[45]

不仅 customer 有偏差，judge 也有偏差。课程提到有研究发现，语言模型作为评价者时，会高估对话的人类感、流程顺畅度、整体质量和用户未来复用意愿。也就是说，agent 与 agent 互动、再由另一个 agent 评价，可能形成一套过于乐观的闭环。

[44]视频画面时间区间：01:20:00–01:21:00。

[45]视频画面时间区间：01:22:20–01:23:00。

图 38: LLM judge 往往比人类更高估 agent-customer 对话质量，尤其是 human-like 指标。[46]

自进化系统最怕自嗨评估

如果任务由 LLM 模拟，反馈由 LLM 生成，评价也由 LLM 完成，系统可能在“看起来变强”的代理世界里优化，而不是真正对人类有用。这是 Drift Monitor 必须关注的核心风险。

9.2 本章小结

Verbalized feedback 提供了从交互到参数更新的道路，但它依赖三件事：正确识别反馈、构造可靠训练信号、避免评估闭环偏差。对 self-evolving agents 来说，反馈学习越强，越需要外部校准和漂移监控。

10 MetaHarness：让强模型设计弱模型的 Harness

课程最后回到 Harness 自我更新。讲者做了一个实验：让一个较强模型去找一个“不聪明”的 AI，拿它跑一个 agent benchmark。如果表现不好，强模型要修改弱模型的 ‘AGENT.md‘，直到分数尽量提高。强模型选择了 Haiku 3.5 一类较弱模型，并让它跑 PinchBench。

[46] 视频画面时间区间：01:23:20–01:24:40。

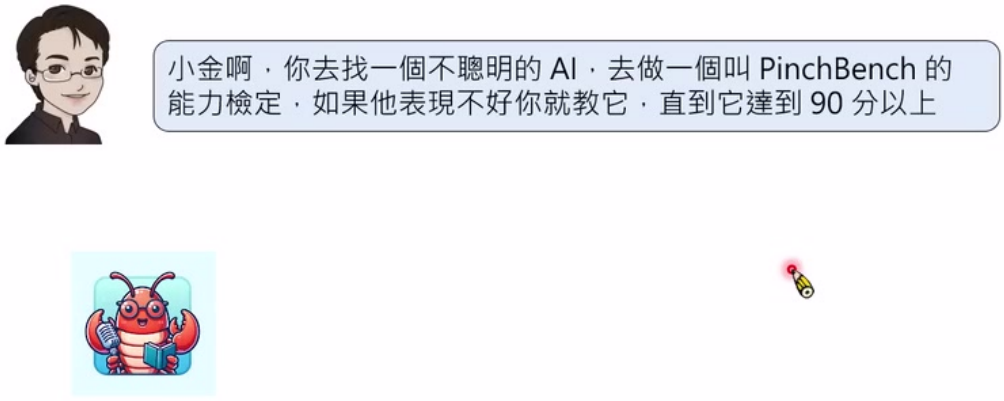

图 39: 实验设定：让强模型教较弱模型跑 PinchBench，并通过修改 ‘AGENT.md‘ 提高分数。[47]

第一轮弱模型没有 ‘AGENT.md‘，裸考分数很低。强模型发现 benchmark 的关键是“结果必须写到文件里”，于是加入“答案要写到档案里”一类规则。分数从约 13.5 跳到 57.9。讲者也提醒：这不完全公平，因为从 Round 1 到 Round 2 可能不只改了一句话。

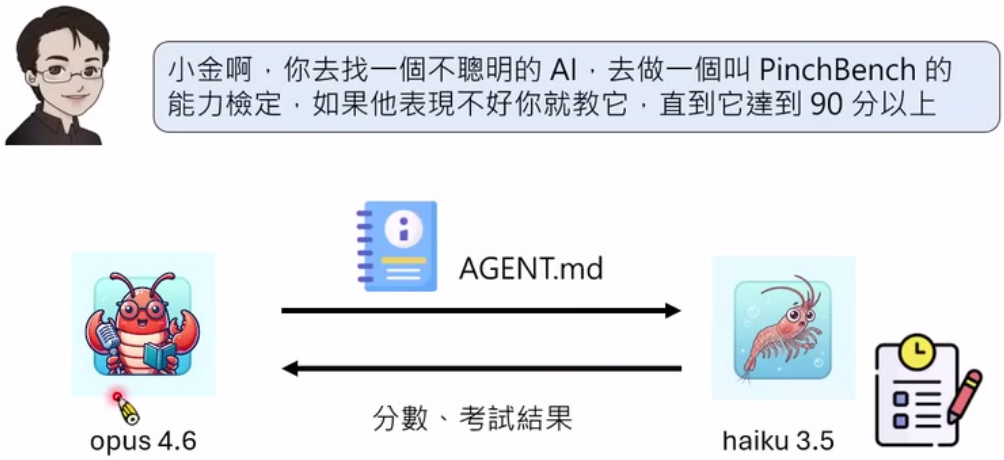

图 40: 强模型根据弱模型表现修改 ‘AGENT.md‘，相当于为弱模型设计 Harness。[48]

[47]视频画面时间区间：01:25:00–01:26:00。

[48]视频画面时间区间：01:26:10–01:26:40。

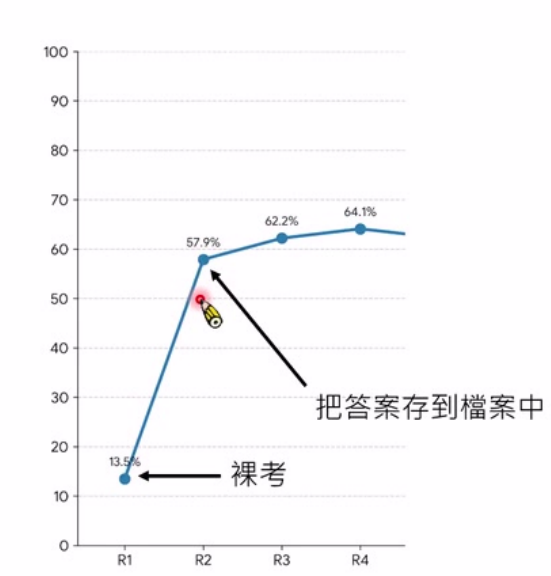

图 41: 加入关键规则后，弱模型分数大幅上升；但实验本身仍有公平性瑕疵。[49]

后续规则包括：不要要求更多解释，题目给的信息已经足够；第一步直接执行列目录命令，先看当前环境；读所有题目提到的文件，再开始做事。最终分数提高到约 85，但没超过 90。

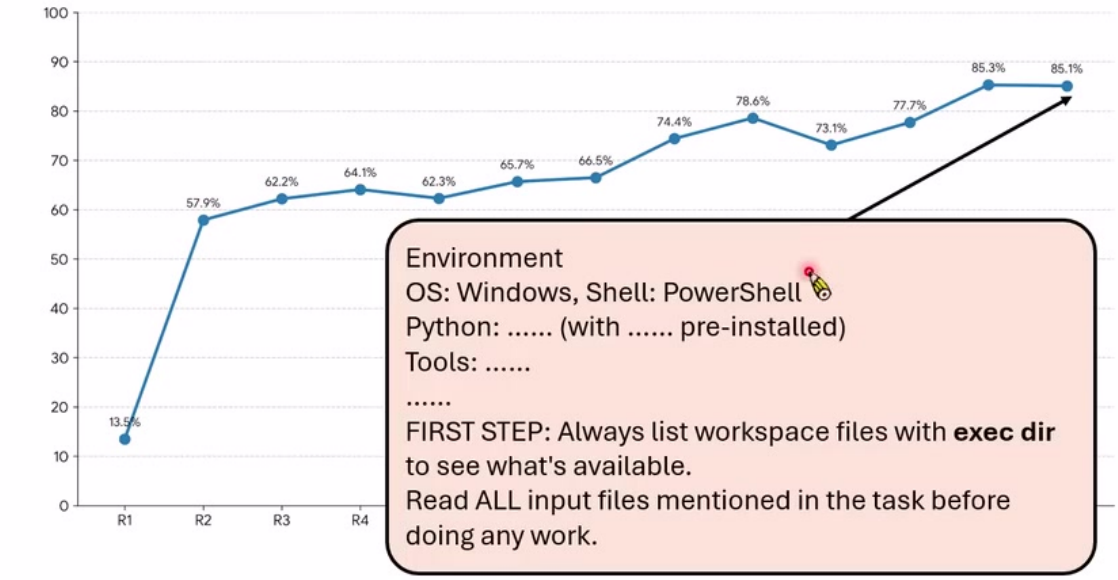

图 42: 最终 ‘AGENT.md‘ 包含环境说明、工具说明、先列目录、读相关文件、再执行任务等规则。[51]

[49]视频画面时间区间：01:27:00–01:27:35。

[51]视频画面时间区间：01:28:50–01:29:45。

MetaHarness 的意义

强模型不只是解题，而是在搜索一个能让弱模型更会解题的 Harness。这把优化对象从答案空间移动到行为规则空间：不是直接告诉弱模型答案，而是修改弱模型未来如何行动。

讲者随后提到 MetaHarness 相关论文，它做了更完整实验：不只在一个弱模型上试，也做跨 LLM；不只在同一批任务上试，也做跨 task。真正有说服力的 Harness improvement 必须能跨模型或跨任务泛化，否则可能只是过拟合某个 benchmark。

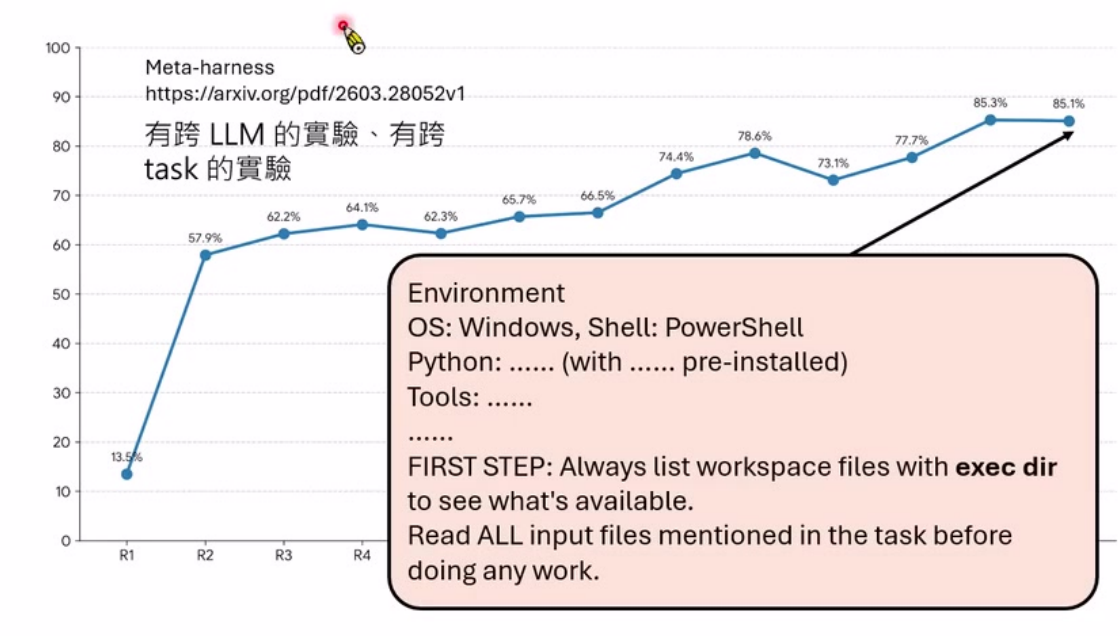

图 43: MetaHarness 类实验：用强模型改其他模型的 Harness，并测试跨模型、跨任务泛化。[52]

10.1 和 self-evolving agents 的关系

MetaHarness 已经非常接近自进化：系统不再只是在当前任务上采样答案，而是在搜索未来自己或其他 agent 的行动规则。更强的未来版本可能会搜索 prompt、memory、workflow、tool policy、evaluation policy，甚至参数更新策略。

这也带来一个研究问题：如果 Harness 是可被学习和修改的，那么 monitor 也必须覆盖 Harness 更新。只监控模型输出是不够的；一个看似无害的规则文件修改，可能长期改变 agent 的权限假设、停止条件、反馈解释方式和用户偏好。

Harness update 需要 gate

当一个 agent 自动写入 'AGENTS.md'、skill、memory 或 workflow 时，更新本身应被记录、审查和回滚。否则系统可能把一次偶然成功经验固化为长期规则，把 benchmark shortcut 固化为“能力”。

[52] 视频画面时间区间：01:30:10–01:32:05。

10.2 本章小结

MetaHarness 把 Harness Engineering 推到自进化边界：强模型可以设计弱模型的行动规则，让弱模型表现更好。要证明这种能力真实存在，必须看跨模型、跨任务泛化；要安全使用这种能力，必须监控 Harness 更新。

11 总结与延伸：Harness 是 Agent 的可进化外壳

课程最后回到一句最重要的话：有时候模型无法完成任务，不是能力不行，而是没有好的 Harness。

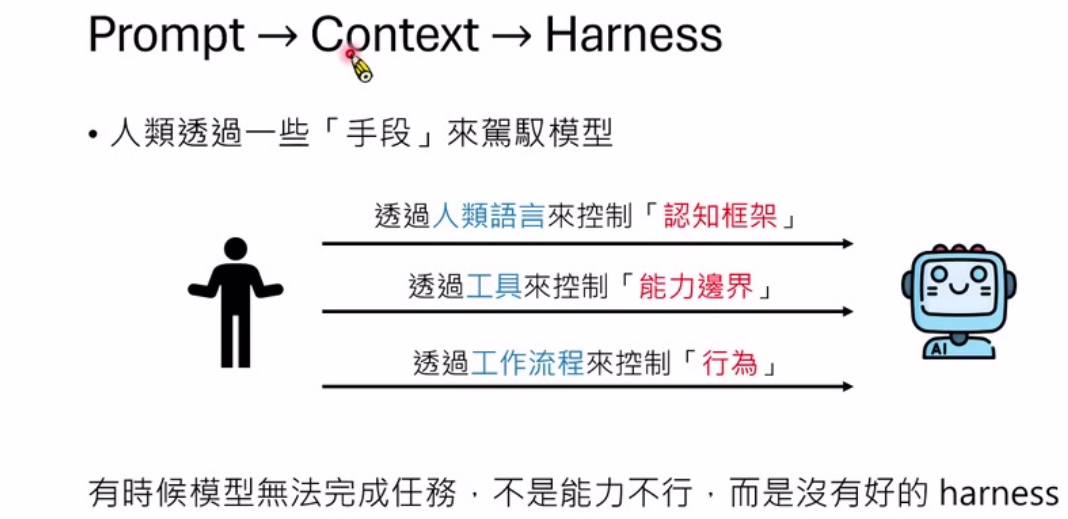

图 44: 课程结论：模型无法完成任务时，未必是能力不行，也可能是没有好的 Harness。[53]

这句话看似简单，但它对研究和工程都有很强的含义。

11.1 对工程实践的含义

当 agent 表现差时，应该按层诊断：

1. 输入层：模型是否看到任务必需信息?
2. 规则层：是否明确当前环境、工具、完成标准?
3. 工具层：是否有足够但不过度危险的工具?
4. 反馈层：失败信息是否被返回给模型，且足够具体?
5. 工作流层：是否需要 planner、generator、evaluator、revisor?
6. 记忆层：经验是否被正确保存、检索、压缩和清理?

[53] 视频画面时间区间：01:32:10–01:32:20。

7. 评估层：分数是否来自真实任务，还是来自 LLM 自我模拟?

只有这些层都合理，换更大模型才可能充分发挥价值。否则，更强模型可能只是更流畅地走错路。

11.2 对 self-evolving agents 的含义

在 self-evolving agents 研究里，Harness 是一个非常关键的中间层。它比模型参数容易改：改一条规则、加一个工具、写一个 skill、改一个 workflow，都比重新训练模型便宜。但它又比普通 prompt 更危险，因为它能长期改变 agent 的行动方式。

可以把自进化对象分成：

对象	更新方式	主要风险
答案	self-correction、rerank、verifier	只修当前输出，容易被多数投票或评估器偏差误导。
Prompt / 规则	修改 ‘AGENTS.md‘、system prompt、skill	过拟合任务、规则过长、错误经验长期化。
工具 / 权限	增加工具、放宽沙盒、改接口	能力增强同时扩大安全边界。
Workflow	改 planner / evaluator / revisor 结构	评估器偏差、循环膨胀、上下文污染。
Memory	写入、压缩、删除、检索策略	错误记忆、遗忘关键事实、隐私泄漏。
参数	SFT、DPO、RL、持续微调	目标漂移、灾难性遗忘、代理目标过拟合。

给 Drift Monitor 的连接

如果 Drift Monitor 只看最终回答，它会漏掉最危险的变化：规则文件被改了、skill 被写入了、memory 检索策略变了、工具权限扩大了、评估器更容易被讨好了。Harness Engineering 告诉我们，monitor 必须把这些外壳更新也当作一等对象。

11.3 一个可执行的 Harness 设计检查表

把本讲整理成实践 checklist，可以得到：

- 规则文件短而硬：说明环境、工具、流程、完成标准，不写百科全书。
- 工具接口 agent-first：结构化、可解析、错误信息明确，避免把人类 GUI 直接丢给模型。
- 权限分层：读、写、执行、联网、浏览器、外部发布分别控制。
- 每个修改后有验证：代码有 tests/lint，文档有 preview，发布有 live URL 检查。
- feedback 可审计：记录来源、时间、触发动作、是否被写入 memory 或 skill。

- Harness update 可回滚：每次修改规则、skill、memory、workflow 都应有 diff 和 rollback 路径。
- 评估要接地：不能只用 LLM customer 和 LLM judge 形成自我强化闭环。

11.4 最终小结

Harness Engineering 的价值在于，它把“模型能力”拆开了。一个失败的 agent 可能不是推理不够，而是没有先看文件；不是不会写代码，而是没有把答案写到文件；不是不能完成任务，而是工具权限不允许；不是不会改进，而是 feedback 没有被正确识别和保存。

对学习者来说，这一讲是从 prompt 技巧走向 agent 系统设计的转折点。对研究者来说，它把 self-evolving agents 的更新对象从模型参数扩展到 prompt、memory、tools、workflow、skills 和 evaluation。对安全来说，它提出一个直接要求：任何能长期改变 agent 行为的 Harness update，都应该被监控、验证和治理。