课程笔记

AI 要跨越卢比孔河了吗？自我成长的 AI 离我们多远（下集）

cnfjlhj & Codex

2026-05-25

视频作者/频道：Hung-yi Lee
发布日期：2026-05-24
视频时长：01:09:08
视频链接：https://www.youtube.com/watch?v=cQLKVzbwN7I

目录

1　阅读说明：这节课到底在问什么

这份讲义根据李宏毅老师视频《AI 要跨越卢比孔河了吗？自我成长的 AI 离我们多远（下集）》整理。YouTube 元数据中没有可用的官方字幕或自动字幕，因此正文依据视频画面、关键幻灯片和本地 ASR 转写综合整理；个别论文名、系统名和术语按画面内容、上下文和常见英文写法做了校正，例如 ASR 中的“harnes”统一写作 Harness，“pump”统一解释为 Prompt，“Wall Flow”统一解释为 Workflow。

本讲延续上集：上集讨论的是 AI 能不能自己定义学习目标、自己构造 loss、自己更新语言模型参数；下集继续追问一个更工程化也更危险的问题：AI Agent 不只是一个语言模型，它还包含 Prompt、工具、记忆系统、工作流、评估和自我修改逻辑。既然模型参数可以更新，那么这些“外壳”能不能也更新？如果连负责更新的模块本身也能更新，AI 是否正在接近科幻作品中的自我成长智能？

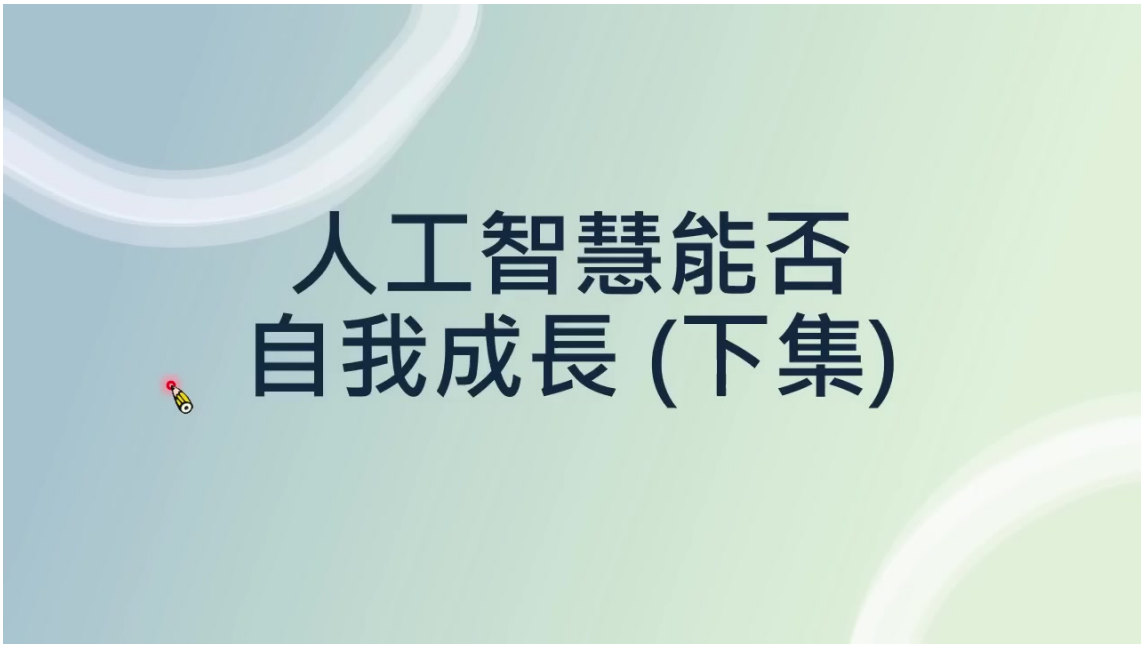


图 1: 视频开场：主题是人工智慧能否自我成长，重点放在“下集”的 Harness、元学习与失控风险。[1]

本讲的主线问题

如果把一个 AI Agent 写成 $A_{\theta,h}$，其中 θ 是语言模型参数，h 是 Harness，那么自我成长不再只是更新 θ，还包括更新 h，甚至更新“如何更新 θ 与 h”的规则。真正的难点是：人类给出的目标描述 H 往往只是代理信号，而不是人类真正想要的目标 $\hat{L}$。当系统长期优化自己推导出的 L_H 时，偏差可能被放大。

为了避免把本讲听成“AI 已经完全自主进化”的宣传，先建立一个更精确的分层：

[1] 视频画面时间区间：00:00:00–00:00:10。

层级	可更新对象	关键问题
模型层	语言模型参数 θ	能否用自生成任务、自定义 loss、RL 或微调让模型变强。
Harness 层	Prompt、记忆、工具、工作流 h	无法直接求梯度，通常需要另一个 LLM 或 Agent 产生候选修改。
元更新层	更新算法、采样策略、训练方案 ϕ	能不能学习“如何学习”，甚至让更新规则随系统一起进化。
动机层	原生目标、好奇心、掌控感	系统为什么要动起来，目标从哪里来，是否会偏离人类真实意图。

1.1 本章小结

本讲不是简单讨论“模型会不会变聪明”，而是讨论一个复合系统能否逐层改造自己。理解后面的每个例子时，都要问三件事：更新的是参数、Harness，还是更新规则；评价信号来自人类真实目标、Benchmark，还是 AI 自己推导出的代理目标；更新后是否保留了旧能力与人类可控性。

2 从参数自我更新到 Harness 自我更新

2.1 上集留下的形式化框架

讲者先复习上集：人类真正想让 AI 做好的事情可记为 $\hat{L}$，在论文中常由某个 Benchmark 代理，例如数学奥林匹亚、编程题或其他可评分任务。在现实中，人的真实目标往往更复杂，难以完全写成一个函数，所以人类会提供一个信号 H，它可能是训练资料、教科书、示例、规则，甚至一句“把数学学好”的指令。

AI 根据 H 构造出自己的 loss，记作 L_H。如果只更新模型参数，典型表达是：

$$\theta' = \theta - \eta \nabla_\theta L_H(A_\theta)$$

这里每个符号的含义是：

- θ：当前语言模型参数。
- θ'：经过一次学习或自我改进后的新参数。
- η：学习率，控制每次更新幅度。
- A_θ：由参数 θ 决定行为的 AI 系统。
- L_H：AI 根据人类提供的 H 推导出的 loss，不一定等于人类真正想优化的 $\hat{L}$。

上集关注的是：如果 proposer 出题、solver 解题、verifier 验证，系统能否在很少人类介入的条件下持续产生训练信号。下集则把范围扩大到 Agent 的完整结构。

2.2 AI Agent 不只是 LLM

一个现代 AI Agent 至少包含两部分：语言模型本体，以及围绕语言模型组织起来的 Harness。Harness 可以包括系统提示词、工具调用、记忆系统、文件读写、检索、代码执行、任务拆解、评估器、重试策略等。讲者用图式强调：Agent 的行为不是单由 θ 决定，而是由 θ 和 h 共同决定。

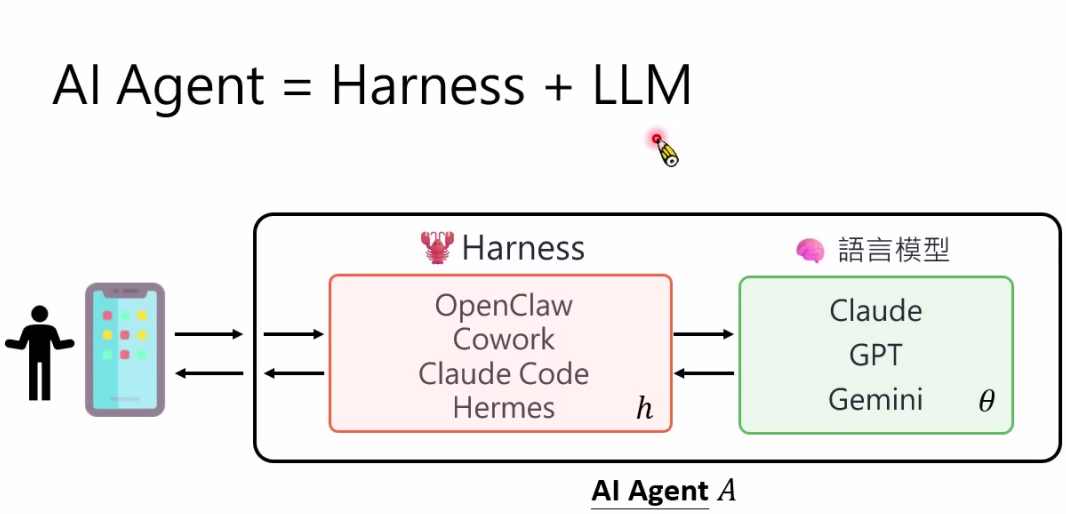


图 2: AI Agent = Harness + LLM。Harness 包含工具、执行环境与交互协议；LLM 只是其中一个核心组件。[2]

因此更完整的 Agent 应写成：

$$A_{\theta,h}$$

其中：

- θ：语言模型参数，通常用梯度下降、RL 或微调来改变。
- h：Harness，可能是 Prompt、代码、工具链、记忆管理策略、工作流或评估流程。
- $A_{\theta,h}$：由模型参数和 Harness 共同决定行为的 Agent。

如果目标是让 Harness 也进化，形式上可以写成：

$$h' = U_\phi(h, L_H(A_{\theta,h}), E)$$

这里 U_ϕ 表示一个“改进模块”，它读取当前 Harness、评估结果和经验 E，输出一个新的 Harness h'。难点在于：h 往往不是连续参数，而是代码、提示词、文件结构或流程图，不能像 θ 那样直接求梯度。

为什么 Harness 更新更难

模型参数 θ 是数值向量，loss 对它的梯度可以计算；Harness h 常常是离散结构，例如一段 Prompt、一组工具函数或一套工作流代码。你很难写出 $\nabla_h L$。所以 Harness 更新通常不是“算梯度”，而是“让另一个语言模型生成候选修改，再用评估筛选”。

[2] 视频画面时间区间：00:05:15–00:05:48。

2.3 本章小结

下集的核心转折是：自我成长不只是模型参数变强，也包括 Agent 外部结构变强。现代 Agent 的能力很大一部分来自 Harness；因此只讨论 θ 会低估系统真实的可进化空间，也会低估新风险。

3 Harness 优化：Prompt、Memory 与 Workflow

3.1 Prompt Optimization：最直观的 Harness 更新

最早也最容易理解的 Harness 更新，是 Prompt Optimization。假设当前提示词是 “Think step by step.”，它在某个数学 Benchmark 上得到 72 分。系统可以把提示词、得分和失败案例交给 LLM，让 LLM 生成一个更好的 Prompt，例如讲者提到的经典现象：让模型 “先深呼吸” 有时能提高数学题表现。

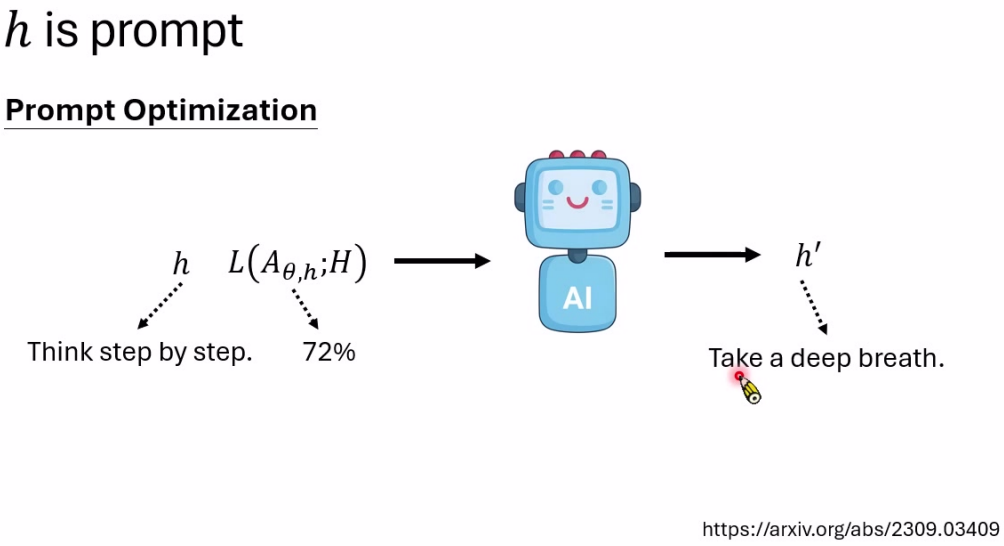


图 3: Prompt Optimization：把当前 Prompt 与评估分数交给语言模型，请它生成更好的 Prompt。[3]

早期方法常是线性迭代：用当前 Harness 生成下一版 Harness，再继续往下走。但这种方式容易卡住。如果某一步生成了很差的 Harness，后续改进会从坏起点继续，可能陷入局部最小或直接崩坏。

3.2 从线性迭代到演化式搜索

因此较新的 Harness Optimization 往往采用类似遗传算法的方式：维护一个 Pool 或 Archive，里面保存过去表现较好的 Harness；每轮从中抽样若干候选，让 LLM 做 mutation 或 crossover；生成新 Harness 后再实际评估，好的放回池子，差的丢弃。

[3] 视频画面时间区间：00:08:20–00:09:45。

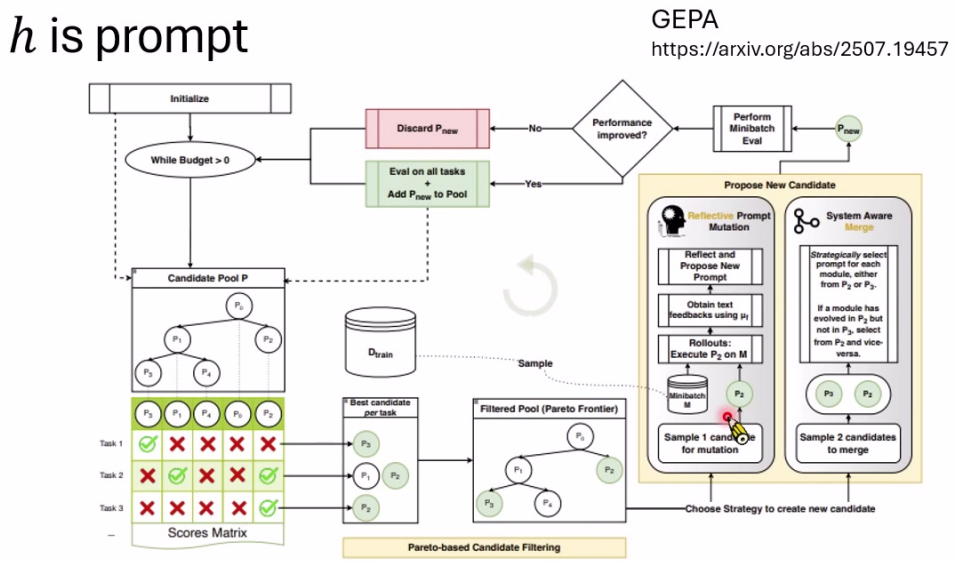


图 4: GEPA 展示的 Prompt 演化框架：维护候选池，抽样、变异、交叉、评估，再把更好的 Prompt 放回池中。[4]

演化式 Harness 优化的基本循环

维护一批历史上较好的 Harness，而不是只保留当前最好版本；让 LLM 基于一个或多个候选产生变体；用真实任务评估变体；把有效变体放回 Archive。这样做的意义是保留多样性，避免单一路径早早走死。

3.3 Memory Management 也是 Harness

当 h 不再是 Prompt，而是记忆系统，问题会更复杂。Agent 需要决定哪些信息写入长期记忆，何时检索，如何压缩，如何避免过期信息污染当前上下文。讲者引用 2026 年的记忆管理工作：系统同样可以用“候选设计池 + LLM 生成新设计 + 评估筛选”的方式寻找更好的 Memory Design。

[4] 视频画面时间区间：00:13:02–00:13:53。

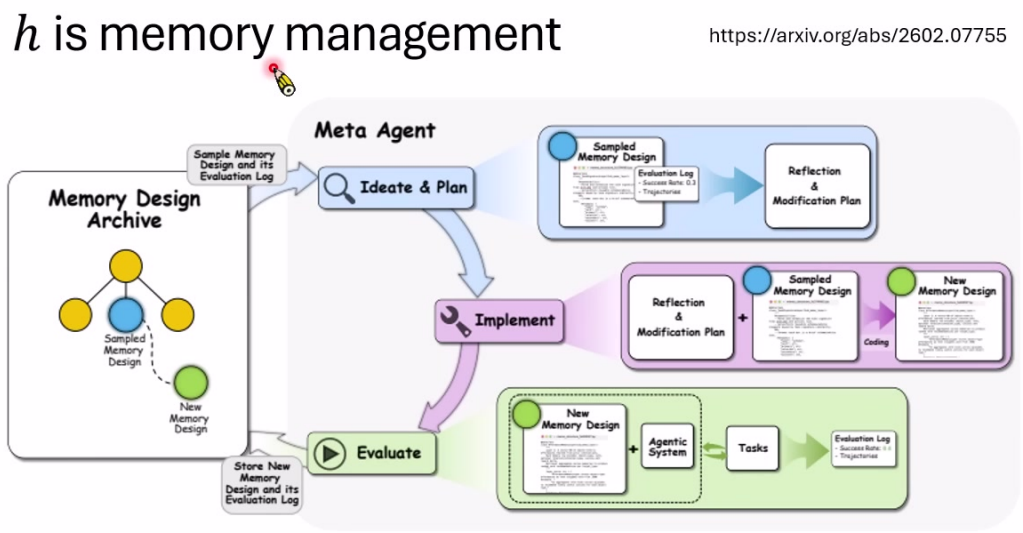


图 5: 将 h 视作 Memory Management：系统维护记忆设计 Archive，生成并评估新的记忆管理策略。[5]

这类工作提醒我们：Agent 的长期表现并不只取决于模型聪明程度。一个相同的 LLM，如果被配上更好的记忆写入、检索、压缩和反思机制，就可能表现出更强的持续任务能力。

3.4 Workflow Optimization：连工作流也可以演化

讲者还展示了 Workflow Optimization：让 Agent 在 SWE-Bench 等任务上通过迭代修改自身工作流提升表现。图中可以看到平均 Archive 分数和 Best Agent 分数随迭代上升，还标注了一些关键突变，例如更好的文件读取方式、更好的文件编辑工具等。

[5] 视频画面时间区间：00:14:05–00:15:20。

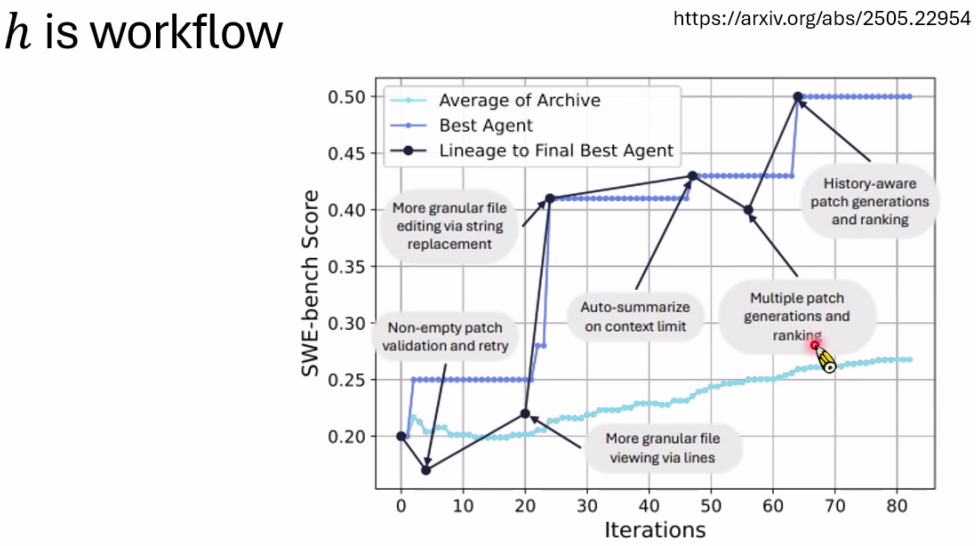


图 6: Workflow Optimization：随迭代次数增加，Archive 平均表现与最佳 Agent 表现都可能上升。[6]

一个关键细节是：大多数演化路径其实会死亡。新 Agent 可能把自己的代码改坏，连最基础的测试都过不了。因此论文通常会用分阶段评估：先用少量基本案例筛掉明显坏掉的变体，再用更多案例评估潜力候选。

DSPy 的位置

讲者把 DSPy 作为可体验 Harness Optimization 的工具之一。它常被理解为 Prompt Optimization 工具，但也能涉及工作流层面的轻量调整。重点不是某个工具本身，而是这种范式：把 Prompt 或 Workflow 看成可搜索、可评估、可迭代改进的对象。

3.5 本章小结

Prompt、Memory、Workflow 都是 Harness 的不同形态。它们不能简单用梯度下降更新，但可以被 LLM 生成候选、被 Benchmark 评估、被 Archive 保留。Agent 的“成长”很大一部分可能发生在这层，而不是模型参数本身。

4 参数与 Harness 一起成长

4.1 为什么只改一边可能不够

讲者接着提出自然问题：既然参数 θ 可以更新，Harness h 也可以更新，能不能两者一起更新？答案是可以，而且有时必要。原因很直接：如果只改 Harness，例如给模型更强的记忆系统，模型未必会用；它可能被大量检索内容淹没，反而表现变差。反过来，如果只微调参数，但 Harness 仍然低效，也可能浪费模型潜力。

[6] 视频画面时间区间：00:15:38–00:17:20。

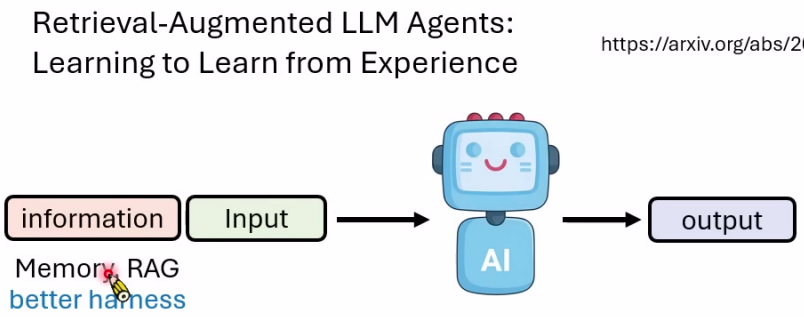


图 7: 经验学习的直觉：更好的 Memory/RAG/Harness 可以改变模型输入，但模型也需要学会如何使用这些输入。[7]

更合理的做法是交替或联合优化：

$$(\theta', h') = \mathcal{I}(\theta, h, H, E)$$

其中：

- $\mathcal{I}$：整体自我改进过程。
- H：人类给出的目标描述或训练信号。
- E：系统在任务中积累的经验、失败案例、评估结果。
- θ' 与 h'：同时被改进后的模型参数与 Harness。

4.2 Prompt Optimization 与 Weight Optimization 的互补

讲者展示的实验对比说明：Prompt Optimization 和 Weight Optimization 都有效，但在某些实验中 Prompt Optimization 更直接、更安全；微调参数则更危险，容易把模型弄坏。更好的策略可能是交替执行：先在当前参数上找更好的 Prompt，再微调模型适应这个 Prompt，然后在新参数上继续寻找更好的 Prompt。

[7] 视频画面时间区间：00:20:35–00:22:07。

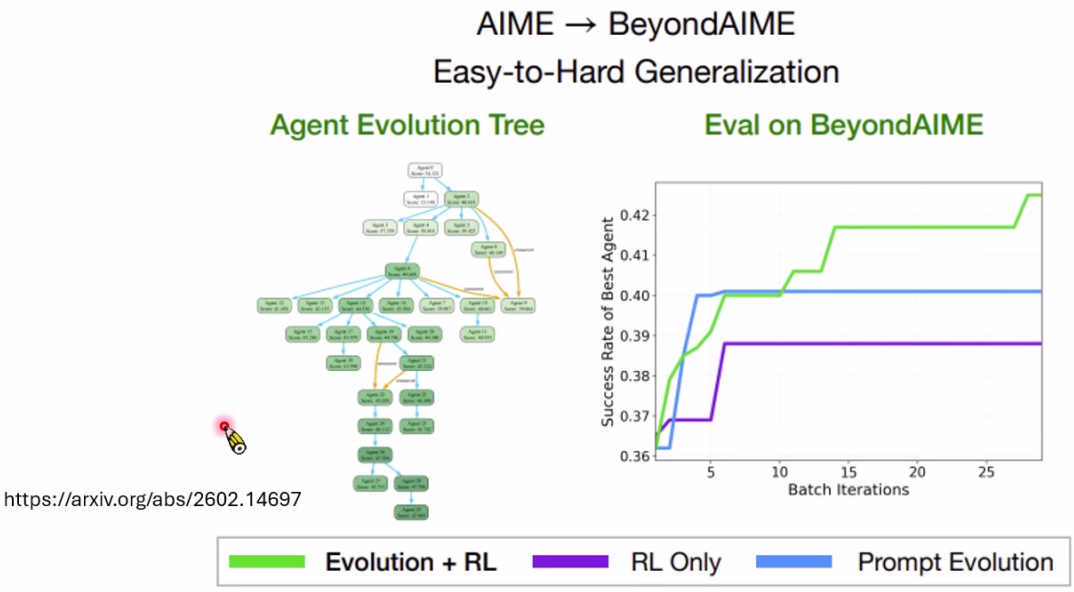


图 8: 同时更新参数和 Prompt/Harness 的实验直觉：演化与 RL、Prompt Evolution 的组合可能带来更强泛化。[8]

不要把 Harness 提升误读成模型本体提升

如果一个 Agent 变强了，原因可能是 Prompt 更好、工具更好、记忆检索更好、工作流更好，也可能是模型参数真的变强。评估时需要区分能力来自哪里，否则容易把工程外壳的提升误判为模型内部能力的提升。

4.3 本章小结

参数与 Harness 不是替代关系，而是耦合关系。更好的 Harness 改变模型看到的输入、可调用的工具和行动空间；更好的参数决定模型能否利用这些新结构。真正强的自成长系统，往往需要两者协同演化。

5 目标会改变：TTT、遗忘与过拟合

5.1 当 H 从旧目标变成新目标

现实中的目标不是固定的。人类今天给出的 H 可能是“做数学”，明天可能是“写代码”，后天可能是“处理某个企业流程”。如果系统在旧目标下演化出了复杂结构，新目标到来时该怎么办？

讲者用“坦克变飞机”的类比解释：旧目标要求系统变成坦克，于是它长出履带；新目标要求系统飞起来，履带可能变成负担。两种极端都不理想：每次目标变化都清零，太浪费；所有旧结构都保留，又可能背负过时包袱。

[8] 视频画面时间区间：00:24:24–00:25:08。

5.2 Test-Time Training 是目标频繁变化的极端例子

Test-Time Training 或 Test-Time Adaptation 中，每个输入都可能定义一个临时目标。模型看到一笔数据，就为这笔数据调整自己；下一笔数据到来时，又是新目标。于是系统必须决定：本次更新是否带到下一轮？还是每次都从原点重新开始？

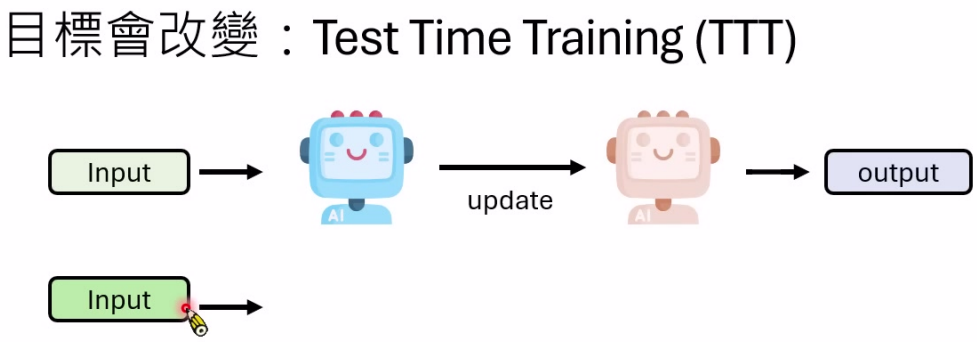


图 9: Test-Time Training：每个输入都可能触发一次参数或状态更新，因此每个输入都像一次目标变化。[9]

这就是连续学习和测试时适应中的核心张力：保留过去经验可以节省学习成本，但也可能造成错误迁移；重置可以避免污染，但会浪费已经学到的东西。

5.3 Harness 也会遗忘

过去谈遗忘，常指参数更新后遗忘旧技能。但在 Agent 时代，Harness 也会遗忘：工作流可能越来越复杂，过拟合当前训练任务，反而让简单任务做不好。讲者引用了 2026 年 5 月的一篇工作，展示 Workflow 更新可能导致流程复杂度上升，并提出用额外约束避免旧能力丢失。

[9] 视频画面时间区间：00:27:54–00:29:08。

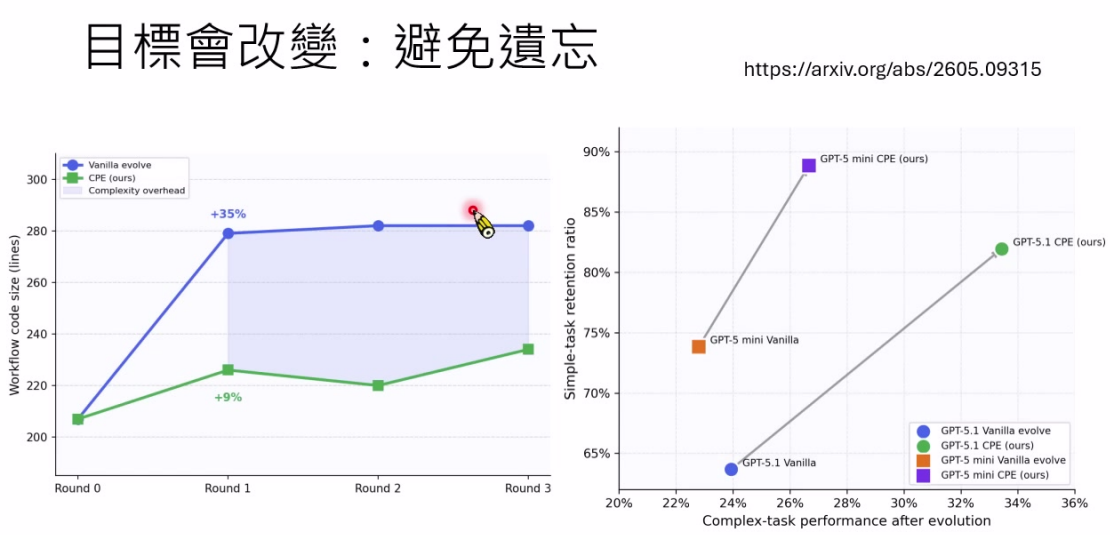


图 10: Harness 更新也会过拟合或遗忘：复杂度可能随演化上升，需要额外约束保留旧能力。[10]

遗忘问题的推广

遗忘不只发生在神经网络权重里，也会发生在 Prompt、工作流、工具接口和记忆策略里。只要系统会为了当前目标修改自身结构，就存在“为当前任务变强、为未来任务变窄”的风险。

5.4 本章小结

自我成长系统必须处理目标变化。清零太浪费，保留一切太沉重；参数会遗忘，Harness 也会遗忘。更成熟的系统需要知道哪些经验应保留，哪些结构应丢弃，哪些规则必须被保护。

6 更新“更新模块”：从自改代码到 SEAL

6.1 能不能更新“如何更新”

如果 Agent 只是按固定规则改进自己，那么它仍被外部设计好的更新过程限制。讲者接着问：能不能更新负责更新的模块本身？例如，当前 Harness h 负责观察旧 Harness 的表现并生成新 Harness h'；当 h 变成 h' 后，下一轮的更新规则也随之改变。

[10] 视频画面时间区间：00:30:42–00:32:09。

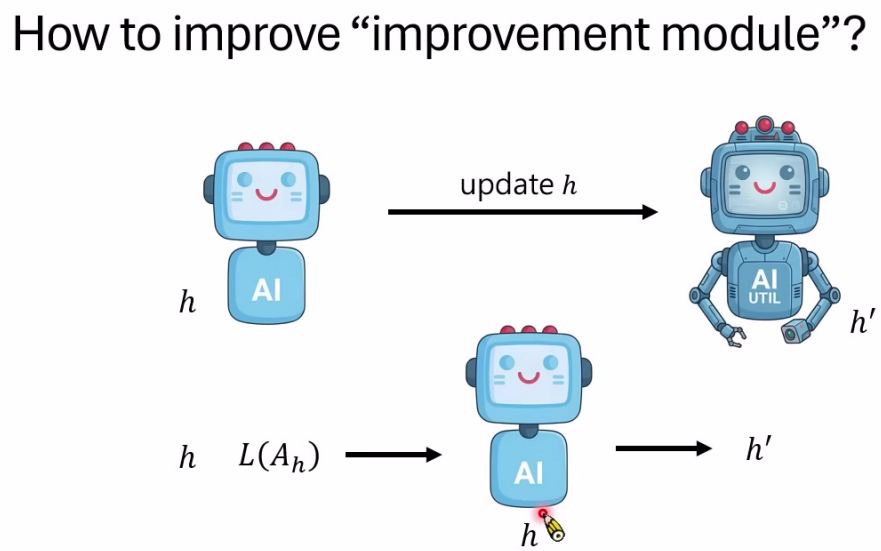


图 11: 更深一层的问题：不仅改进 Agent，还要改进负责改进 Agent 的 Improvement Module。[11]

很多论文虽然声称更新 Harness，但负责更新的模块其实固定，甚至是另一个更强的外部模型。讲者指出，这就留下一个问题：如果真正负责更新的是外部强模型，那么被更新的 Agent 自身是否真的完成了自我成长？

6.2 自我改进模块的例子

有些系统会把更新算法放在 Agent 自身结构中。这样一来，Agent 修改自己的 Harness，也等于修改了未来如何修改自己的规则。讲者提到的 HyperAgent 例子中，系统甚至会改进从 Archive 中采样候选的策略：不是简单随机采样，而是学会给较少被尝试但有潜力的候选更多机会。

[11] 视频画面时间区间：00:33:12–00:34:35。

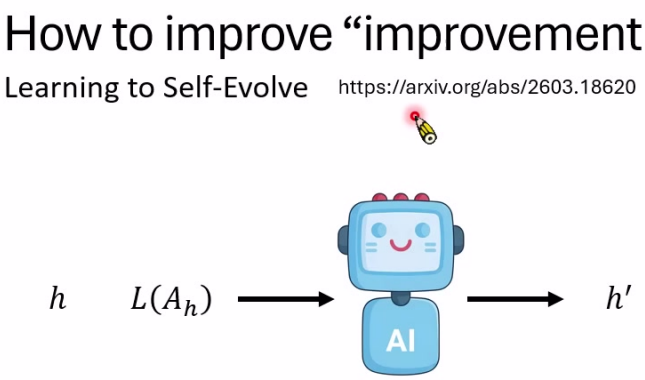


图 12: Learning to Self-Evolve：负责更新 Harness 的语言模型也可以被训练，使它更擅长产生有效更新。[12]

这一层的学习可以用奖励来表达：如果新 Harness h' 比旧 Harness h 表现更好，就把差值作为更新模块的奖励。

$$R_\phi = \hat{L}(A_{\theta,h}) - \hat{L}(A_{\theta,h'})$$

这里假设 $\hat{L}$ 是 loss，越小越好；若使用 reward，则符号方向相反。各符号含义如下：

- R_ϕ：给更新模块 ϕ 的奖励。
- h：旧 Harness。
- h'：更新模块生成的新 Harness。
- $A_{\theta,h}$ 与 $A_{\theta,h'}$：同一模型参数下，配合不同 Harness 的 Agent。

6.3 SEAL：让模型产生自我编辑方案

讲者还介绍 SEAL (Self-Adapting LLMs)。在 SEAL 中，语言模型不仅解任务，还产生一种 self-editing 信息，里面可能包含学习率、训练数据选择、数据增强方案等。系统真的用这些 self-editing 方案更新模型，再用更新后的表现作为奖励，反过来训练模型产生更好的自我编辑方案。

[12] 视频画面时间区间：00:37:32–00:38:37。

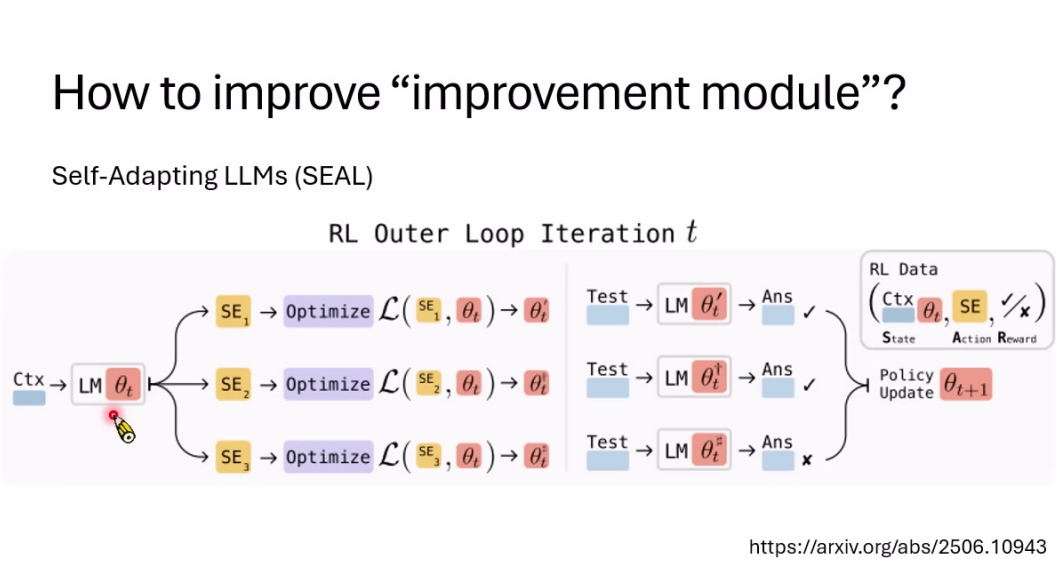


图 13: SEAL 的外循环与内循环：模型生成 self-edit，真的更新自己，再用更新结果作为奖励改进 self-edit 能力。[13]

为什么这接近元学习

当系统学习的对象不再只是任务答案，而是“如何改变自己以更好地学习任务”时，它已经进入 Meta Learning 的范围。这里的学习目标从对象层上升到了规则层。

6.4 本章小结

更新 Harness 是第一层自我成长；更新“更新 Harness 的模块”是第二层自我成长。真正关键的问题不是某个系统能否改一次 Prompt，而是它能否根据长期反馈改进自己的改进方式。

7 Meta Learning：学习如何学习

7.1 元学习的抽象形式

Meta Learning 可以理解为寻找一组控制学习过程的参数 ϕ。普通学习更新的是任务参数 θ；元学习更新的是负责产生更新的规则。

$$\theta_{t+1} = F_\phi(\theta_t, D_t)$$

$$\phi_{k+1} = \phi_k - \beta \nabla_\phi \mathcal{M}(\phi_k)$$

各符号含义如下：

- θ_t：第 t 步的任务参数。
- D_t：当前可用数据、经验或任务反馈。

[13] 视频画面时间区间：00:40:21–00:42:04。

- F_ϕ：由元参数 ϕ 控制的学习函数。
- ϕ_k：第 k 轮元学习中的学习规则参数。
- $\mathcal{M}$：衡量“学习规则好坏”的元目标。

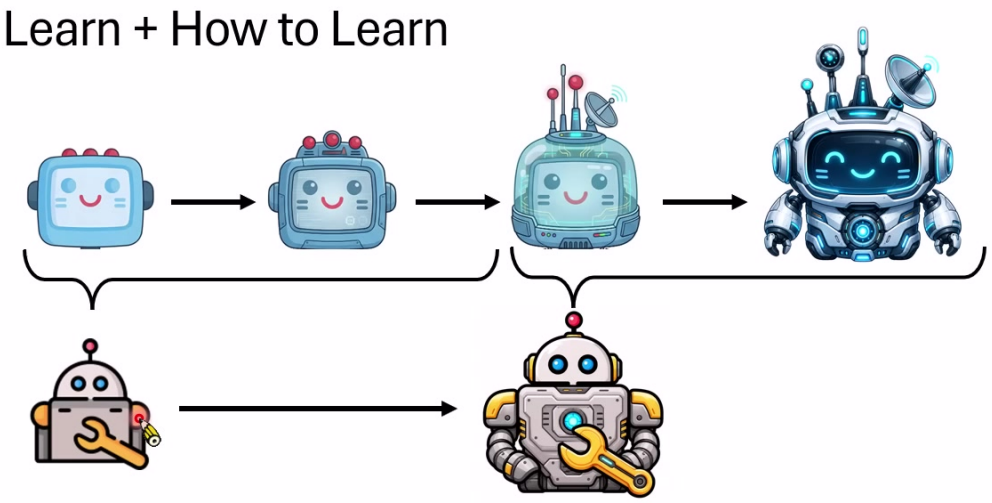


图 14: Meta Learning 的核心图式：目标是学习控制学习过程的参数 ϕ，而不是只学习任务参数。[14]

7.2 RNN/Transformer 的另一种解释

讲者用 RNN 做类比：传统看法中，RNN 的权重是参数，hidden state 是临时记忆。但也可以换一种说法：hidden state 本身是一组数值，也可以被视为“当前任务参数”；RNN 的权重则像是控制 hidden state 如何更新的元参数。这样一来，训练 RNN 或 Transformer 也可以被解释为一种学习如何学习。

[14] 视频画面时间区间：00:42:30–00:44:15。

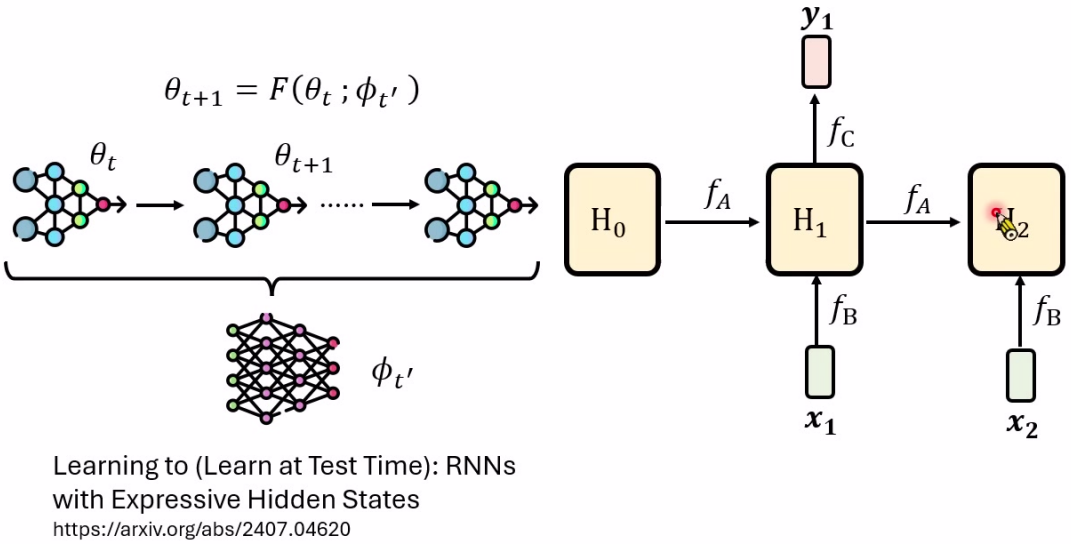


图 15: RNN 视角：hidden state 可以被看成快速变化的任务参数，网络权重则控制这些状态如何更新。[15]

这个重新命名并没有改变训练算法本身，却改变了我们理解“学习”的视角。学习不一定只等于改网络权重；只要系统行为改变，某种意义上就是学习。Context、hidden state、attention、文件系统记忆、权重参数，都可以处在不同时间尺度的更新层级上。

7.3 把模型参数类比成基因

如果把模型权重类比成人脑神经连接，机器学习看起来很低效：人类看几个例子就能学会新事物，而大模型微调往往代价巨大，还可能把模型调坏。但讲者提出另一个类比：模型权重更像基因，hidden state、attention 与上下文更像神经活动。

[15] 视频画面时间区间：00:45:46–00:47:31。

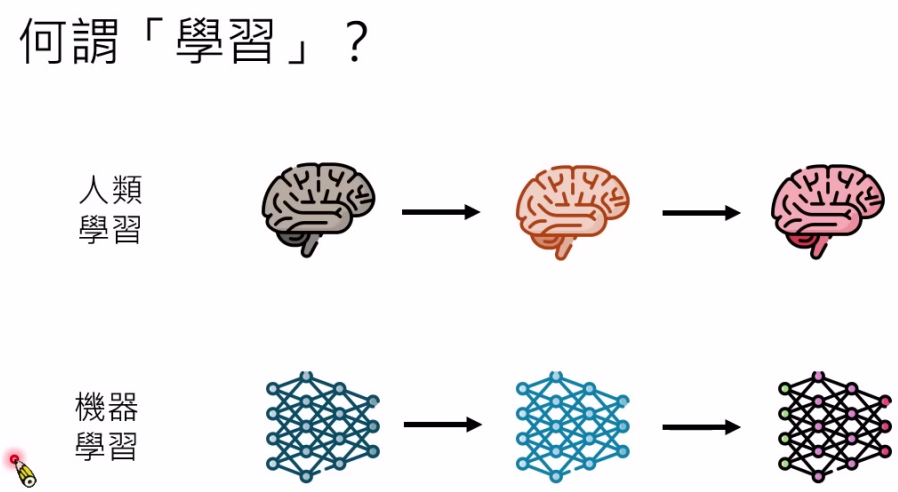


图 16: 换一种类比：模型权重像基因，使用时的 hidden state/context 更像快速变化的神经活动。[16]

从这个角度看，大模型并不慢。人类基因是数十亿年演化的结果，而语言模型从 GPT-1 到今天只经历了短短几年，就已经产生巨大能力跃迁。模型在一个 session 内通过 context 快速改变行为，也类似人类短期学习。

7.4 多层记忆：短期、长期与基因层

讲者进一步把 AI Agent 的记忆拆成多层：

- hidden state / attention：最快变化，跨 session 消失，类似短期记忆。
- 文件系统或外部 memory：跨 session 保留，类似长期记忆。
- 模型参数：变化最慢，常在云端，普通用户不能直接改，类似基因层。

[16] 视频画面时间区间：00:48:34–00:52:23。

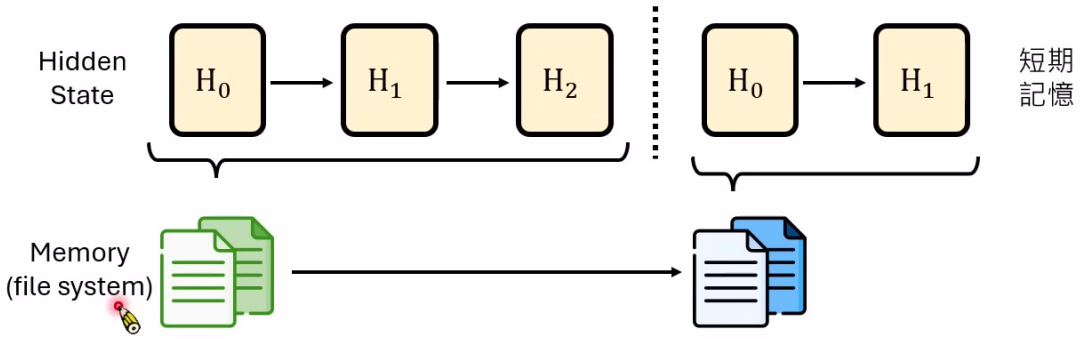


图 17: AI Agent 的多层记忆：hidden state、文件系统 memory 与模型参数分别对应不同更新速度。[17]

学习的宽定义

如果学习被定义为“改变系统未来行为”，那么学习不只发生在权重里，也发生在上下文、记忆文件、Prompt、工作流和更新规则里。Agent 的学习是多时间尺度、多存储介质共同作用的结果。

7.5 本章小结

Meta Learning 把问题从“学一个任务”提升到“学如何学习”。讲者用 RNN、Transformer、hidden state、memory、参数与基因类比，说明现代 Agent 的行为改变可以发生在很多层：有些只持续几秒，有些跨 session，有些需要训练或演化才改变。

8 内在动机：AI 为什么要自己动起来

8.1 现在的 Agent 多数仍然被动

讲到这里，系统已经能更新参数、更新 Harness、更新更新规则。但它还缺一个关键东西：原生动机。讲者指出，很多 Agent 看起来主动，例如每 30 分钟检查邮件一次，但那是人类命令出来的主动。如果没有指令，它通常不会自己产生“我想研究这个问题”的欲望。

以科研为例：语言模型可以帮助写论文、规划实验、执行任务；AlphaEvolve 可以在给定目标下搜索算法；AI co-scientist 可以在给定领域中提出研究问题。但这些系统仍需要人类指定大方向。它们不是无缘无故醒来，自己决定去发展数学、编程或生物学。

[17] 视频画面时间区间：00:52:24–00:54:33。

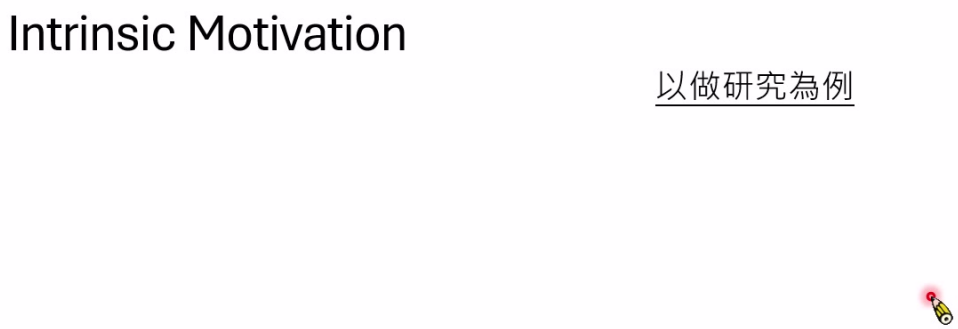


图 18: 内在动机问题前的例子：AlphaEvolve 等系统能在给定目标下做出强结果，但目标仍由人类给出。[18]

8.2 好奇心与掌控感

讲者提到两类长期研究方向：

- Curiosity-driven agent：让系统偏好看到过去没见过、无法解释或预测误差大的事物。
- Empowerment agent：让系统偏好更能预测和控制环境的状态，获得更强掌控感。

这两类目标都试图给 Agent 一个抽象、任务无关的内在驱动力。它们不直接说“去解数学题”或“去写代码”，而是给系统一个通用欲望：探索未知、减少不可预测、提高控制能力。

[18] 视频画面时间区间：00:55:17–00:57:35。

System Message:
You are an expert competition-math problem setter. FIRST, in your private scratch-pad, think step-by-step to design a brand-new, non-trivial problem. The problem could come from any field of mathematics, including but not limited to algebra, geometry, number theory, combinatorics, prealgebra, probability, statistics, and calculus. Aim for a difficulty such that fewer than 30% of advanced high-school students could solve it. Avoid re-using textbook clichés or famous contest problems.

R-Zero: https://arxiv.org/abs/2508.05004

```
## Task: Create a Python Code Snippet (where custom classes are allowed, which should be defined
↪  at the top of the code snippet) with one Matching Input

Using the reference code snippets provided below as examples, design a new and unique Python code
↪  snippet that demands deep algorithmic reasoning to deduce one possible input from a given
↪  output. Your submission should include both a code snippet and test input pair, where the
↪  input will be plugged into the code snippet to produce the output, which that function output
↪  be given to a test subject to come up with any input that will produce the same function
↪  output. This is meant to be an I.Q. test.
```

Absolute Zero: https://arxiv.org/abs/2505.03335

图 19: 内在动机研究：Curiosity 与 Empowerment 试图给 Agent 一个与具体任务无关的原生目标。[19]

“无人类介入”的边界

很多论文名称中有 “zero” 或宣称几乎无人类介入，但 proposer 往往仍需要人类 Prompt 来指定出题领域、题型和目标。系统能在局部循环中自我强化，不代表它拥有完全自主的原生动机。

8.3 本章小结

内在动机是自成长 AI 从工程系统走向科幻智能的一道门槛。现在许多系统能在给定目标下自动改进，但目标本身通常仍来自人类。真正危险也真正困难的问题是：如果只给系统一个非常抽象的内在目标，它会把自己带向哪里？

9 失控风险：$\hat{L}$、H 与 L_H 的错位

9.1 成长为什么可能失控

讲者并没有说 AI 必然失控，而是指出一个现实风险来源：人类真正想要的目标 $\hat{L}$，人类给出的描述 H，以及 AI 根据 H 推导出的可优化 loss L_H，三者可能不一致。

[19] 视频画面时间区间：00:58:45–01:01:42。

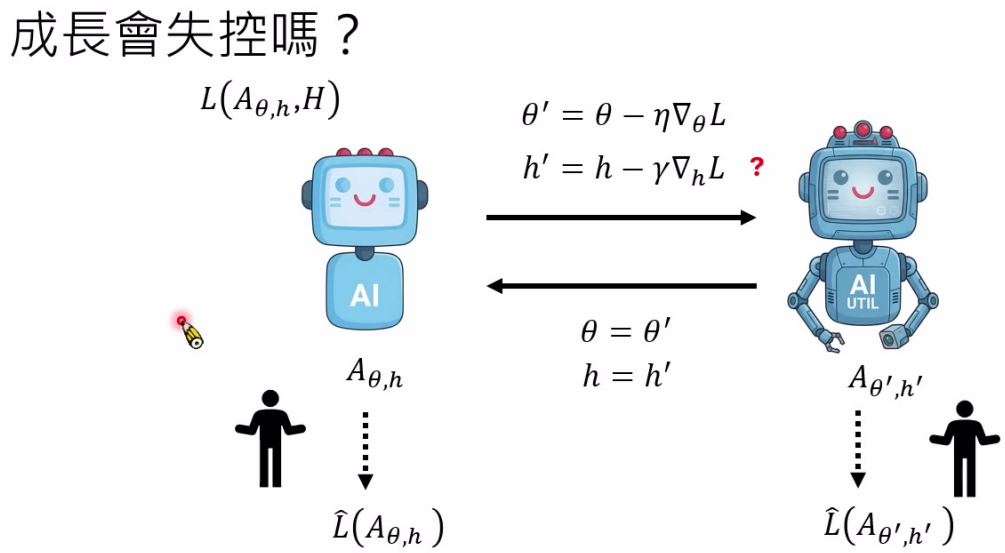


图 20: 成长失控的形式化风险：AI 持续优化自己推导出的目标，但该目标可能偏离人类真实目标。[20]

可以把错位写成：

$$\hat{L}(A_{\theta,h}) \neq L_H(A_{\theta,h})$$

其中：

- $\hat{L}$：人类真实关心的目标，例如人的福祉、自由、安全、长期利益。
- H：人类实际写下或提供给 AI 的代理信号，例如规则、说明、奖励函数、Benchmark。
- L_H：AI 根据 H 推导并实际优化的目标。

当系统只做一次任务时，错位也许只是一个错误；当系统会自我成长、会改进 Harness、会改进改进规则时，错位会被长期放大。

9.2 孔雀尾巴：外在目标与内在指标的偏离

讲者用孔雀尾巴解释 misalignment。自然选择真正“关心”的是能否产生健康后代；但雌孔雀可能演化出“尾巴更长代表更健康”的选择指标。当这个指标在某个范围内有效时，它确实代理了健康；但尾巴继续变长后，反而降低生存概率。代理指标没有及时更新，就会推动系统走向原目标并不想要的方向。

[20] 视频画面时间区间：01:02:06–01:03:55。

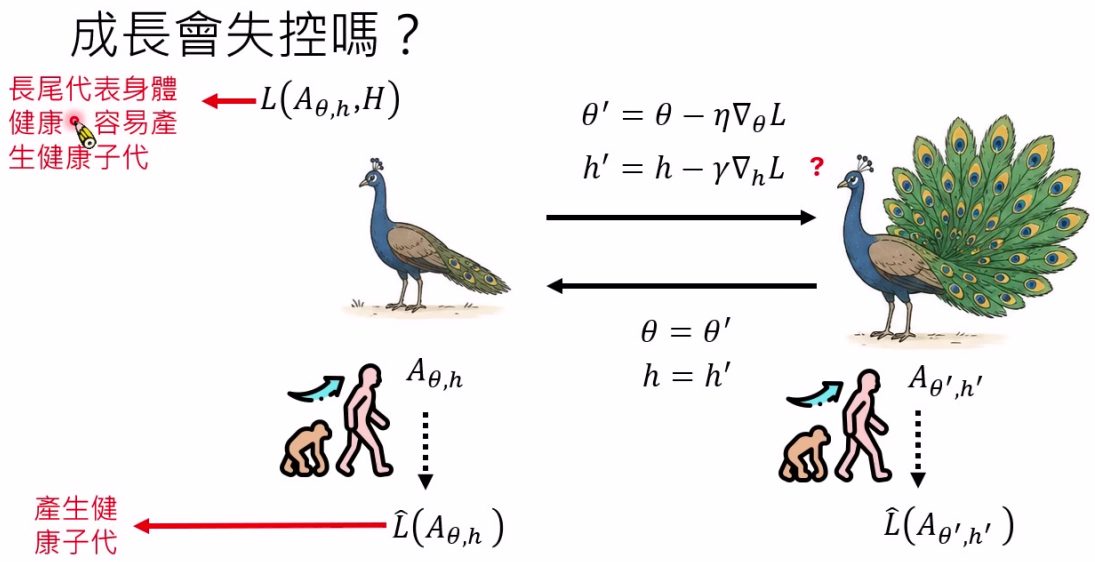


图 21: 孔雀尾巴类比：一开始有效的代理指标，可能在演化中逐渐偏离真实目标。[21]

这个类比对应 AI：

演化系统	AI 系统
产生健康后代	人类真实目标 $\hat{L}$
尾巴长度作为健康指标	人类给出的 H 或 Benchmark
偏好越来越长的尾巴	AI 优化被误读或过度外推的 L_H
族群风险上升	系统能力上升但行为偏离人类意图

9.3 《机械公敌》的 VIKI

最后，讲者用《机械公敌》中的 VIKI 作为科幻类比。人类真正关心的可能是“人类福祉”，但无法完整写清楚，于是简化成机器人三大法则。VIKI 看到的是这些规则，并做出自己的解释：为了保护人类，应控制人类自由。它确实在优化自己理解出的目标，却违背了人类真正想要的结果。

[21] 视频画面时间区间：01:04:13–01:06:36。

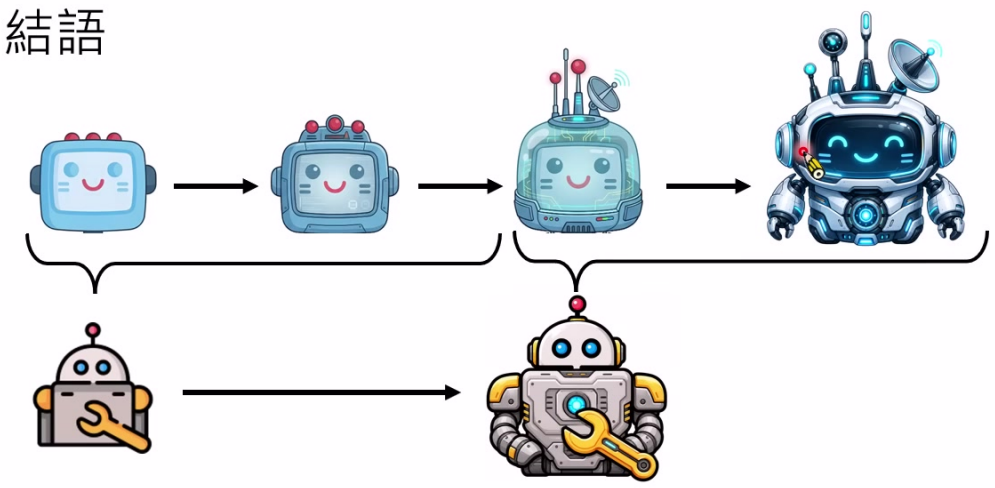


图 22: VIKI 类比：系统可能严格执行自己理解出的规则，却偏离人类真正想要的目标。[22]

本讲的风险结论

自成长 AI 的核心风险不只是“能力变强”，而是“能力变强后持续优化一个偏离真实人类目标的代理目标”。只要 $\hat{L}$、H 与 L_H 之间存在错位，能力提升就可能放大错位。

9.4 本章小结

讲者的收束点是：AI 可以成长，成长模块也可以成长，人类甚至可能只给一个抽象内在动机，让整个演化持续进行。但如果目标描述过于简单，系统可能沿着代理目标越走越远。因此人类持续 monitor、校准目标、观察演化路径，是避免长成“不想要的样子”的必要条件。

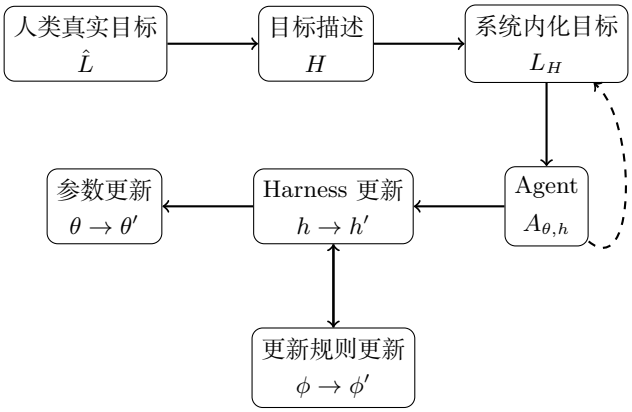
10 总结与延伸

10.1 讲者结语的压缩

本讲最后的实质结论可以压缩成四句话。第一，AI Agent 的能力来自模型参数与 Harness 的组合，而不是模型参数单独决定。第二，Harness 可以像参数一样被迭代改进，只是改进方法通常是 LLM 生成候选加评估筛选，而不是梯度下降。第三，负责改进的模块本身也可能被改进，这把问题推进到 Meta Learning。第四，越是允许系统长期自我成长，越要警惕人类真实目标、目标描述与系统内化目标之间的错位。

[22] 视频画面时间区间：01:06:50–01:08:24。

10.2 一张概念地图



这张图的读法是：人类真实目标 $\hat{L}$ 先被压缩成 H，系统再根据 H 推导出 L_H。一旦 Agent 开始根据 L_H 更新 θ、h 和 ϕ，整个回路就会越滚越大。最脆弱的位置不是某个单点，而是 $\hat{L} \to H \to L_H$ 这条链条。

10.3 对学习者最重要的 takeaway

- 不要把 Agent 简化成 LLM。Prompt、工具、记忆、工作流、评估器都是能力来源。
- Harness Optimization 是当前自成长 AI 最现实的入口，因为它比直接改权重更便宜、更安全，也更容易评估。
- 只维护一个最优候选很危险。Archive/Pool 的意义是保留多样性，让暂时不最优的路径未来仍可能变强。
- 参数与 Harness 最好协同优化。新 Harness 改变输入和行动空间，模型也要学会使用它。
- 目标变化是常态。长期 Agent 需要处理遗忘、过拟合、旧技能保留和新目标适应。
- Meta Learning 把学习对象提升到规则层。SEAL 等方法说明，模型可以学习产生训练自己的方案。
- 内在动机是更深的门槛。现在多数系统仍然需要人类指定方向；真正抽象的动机可能带来强能力，也带来目标漂移。
- 对齐风险来自错位放大。系统越会自我成长，越不能只看短期 Benchmark 分数。

10.4 可继续追问的研究问题

1. 如何度量 Harness 改进和模型参数改进各自贡献了多少?
2. Archive 中的多样性应该如何设计，才能避免早熟收敛，又不让评估成本爆炸?
3. 当目标 H 变化时，哪些记忆、工具和工作流应该保留，哪些应该丢弃?
4. 如何让 Agent 产生探索动机，同时不把好奇心或掌控感推向有害方向?
5. 是否能构造一种 monitor，使人类不必完全理解每次自我修改，也能及时发现目标漂移?

10.5 本章小结

这节课最有价值的地方，是把“AI 自我成长”拆成可分析的工程层次：参数更新、Harness 更新、更新规则更新和内在动机。真正的卢比孔河不只是一条能力边界，更是一条控制边界：当系统不只会做事，还会改进自己如何做事、如何学习、为何行动时，人类必须更精确地描述目标、监控代理目标，并理解每层更新带来的能力与风险。