RESEARCH NOTE · 31 JUL 2026 · UPDATED 3 AUG 2026

Data-driven RSI：Eval、Synthetic Data Ladder 与轨迹产线

RSI 描述一种能够持续递归的自我改进：当前模型参与创造更强的后继模型，后继模型再推进下一轮。本文从最基础的一段——Data——往下推。

Eval | Synthetic data | Harness-Evolve

> ***编辑说明***：*本文由作者与 Codex（GPT-5.6-sol）共同完成。文中的想法仍在整理，个别地方可能表述得不够清楚，也可能存在理解偏差，欢迎指出。*

RSI 想做到什么

RSI 是 Recursive Self-Improvement，递归自我改进。当前模型参与创造更强的后继模型；后继模型继承这套能力，继续发现问题、提出改进并训练下一代。每一轮都改进模型，也改进产生模型的方法。循环可以持续下去，才有“递归”的含义。

今天，这套循环主要由基础模型团队维持。OpenAI、Moonshot AI（Kimi）、智谱 AI（GLM）等团队的研究者不断寻找能力边界、搭建 eval、生产数据、运行训练，再决定下一版往哪里走。模型负责完成一次次任务，人负责改进制造模型的过程。

RSI 希望模型也能做后一部分工作。没有研究者一直守着，LLM 仍能找到能力缺口，想出可验证的改进，准备下一轮训练材料，甚至参与 architecture 的迭代，再检查新模型是否真的变强。新模型接着做同样的事。

为什么先从 Data 开始

Data 是一个容易动手的起点。下一次算法突破会从哪里来，一个小实验放大以后还灵不灵，很难预先知道。数据实验要具体得多：给模型一批没见过的任务，找出反复出现的失败，再把这些失败做成新任务和新轨迹，训练下一版模型。

每一轮的失败都应改变下一批数据。这样一来，模型已经开始决定自己接下来学什么。完整的 RSI 还很远，但这一小段现在就可以做实验。

接下来有三个问题：

- “数学更强”“coding 更强”这种模糊目标，怎么评估到足以指导下一步？
- 一批 synthetic data 看起来不错，怎么知道目标模型能不能学到，又该做多少？
- 过去靠人摸索出来的合成经验，怎么变成一条可以反复跑的轨迹产线？

下文只谈这三件事。

文中的几个词

文章只研究长循环中的一段：当前模型帮忙找能力缺口、做下一轮训练数据；训练出的后继模型，要在全新的任务上稳定进步。这里暂时不讨论模型在一次推理中直接改写参数。

开放目标评估（open-ended evaluation）的目标可以很宽，比如“数学研究能力”。但每一轮都要留下具体材料：做过哪些任务，哪里失败，验证结果是什么，下一批数据准备补什么。

我暂时把这套数据做法叫作 **Harness–Evolve**。Harness 是模型工作的环境。模型在里面反复尝试、修改和筛选轨迹，这一段叫 Evolve。选出的轨迹最后拿去做 SFT，模型的权重也到这一步才更新。

为什么一定会走到 synthetic data

Data-driven 再往前一步，就会碰到 synthetic data。

这在 post-training 里已经很明显了。很多 SFT 样本、偏好对、批评与重写、verifier 筛过的 rollout、tool-use trajectory，都由模型、规则、工具和人共同构造。现在不少能力提升，靠的就是这些被设计过的经验。

PT 也在发生类似的变化。自然数据仍是地基。清洗、去重和筛选做得越好，新的增量就越多地来自重写、转换、难度控制、推理补全和代码执行筛选。已有材料也会被重新做成适合训练的任务。PT 与 post-training 的数据边界会慢慢变淡。

自然数据仍是地基，给了模型知识、语言和对世界的基本认识。它很少会照着某个 checkpoint 的失败生长。模型刚好卡在哪一步，互联网通常不会立刻送来一万条难度合适、反馈清楚的练习。

所谓数据墙，说的是适合下一版模型的数据长得太慢。它得有用、没被反复吃过，质量还要过关；网上有没有新东西只是其中一小部分。对公开文本存量与训练需求的估算也指出，高质量人类文本可能限制训练规模继续增长（Villalobos et al., 2024）。没人知道墙会在哪一天出现。可以确定的是，新增 token 已经不能稳定换来新增能力。

Will DePue 把互联网称为 deep learning 的“一次性补贴”（DePue, 2026）。过去几十年，人类写下网页、代码、论文和讨论时并没有考虑模型训练，这些材料后来恰好组成了一份巨大、便宜而且跨领域的数据集。下一阶段很难再复制同样的偶然性。互联网之外更有价值的部分，主要是组织内部的工作流、专家的隐性判断、没有被记录的失败，以及只能在真实环境里观察到的过程。

数据墙里混着两个问题。**Volume** 问已有领域还缺多少好样本；**coverage** 问哪些任务、工具、边界情况和长程过程从来没有被记录。Synthetic data 擅长补 volume。碰到 coverage，还是要先从人和真实环境里拿到锚点，模型才能围绕它展开和组合。

> *稀缺的是下一步刚好有用的经验。*

模型若要继续 scaling，就得参与制造下一轮经验：把自己的失败做成任务，把工具和 verifier 的反馈留在轨迹里，再把有用的部分训练回去。对我来说，这就是 synthetic data 与 RSI 接上的地方。

当然，这条路也很容易做偏。模型批量写出自己本来就会的答案，做一亿条也没多大用。同一个模型同时生产、打分和筛选，旧偏好还会越滚越大。后面谈 eval、Ladder 和轨迹产线，都是在处理这些麻烦。

1. Eval：在模糊目标下驱动自我迭代

Benchmark 都有保质期。模型逐渐接近满分，题型被反复研究，样本也可能混进训练集，它就越来越难区分最强的系统。不同题库老化的速度不一样，但趋势已经很清楚：一项覆盖 60 个常用文本 benchmark 的研究发现，近一半已经高度饱和（Akhtar et al., 2026）。Contamination 又让分数多了一层疑问（Singh et al., 2024）。

通常的办法是继续请人出更难的题，可这件事越来越贵。Humanity's Last Exam 从七万多道候选题中筛出 2,500 道，动员近千名专家，经过多轮审核（Center for AI Safety & Scale AI, 2025）。如果模型的迭代周期继续缩短，人类出题、维护答案和更新验证集的速度，迟早会跟不上。

所以 eval 也得跟着模型走。目标可以先写得宽一点，比如"数学研究能力更强"或"能完成更长的 coding task"。每轮测试以后，模型根据 failure map 去找新的边界：出新题、造反例、调难度，再补一批 benchmark 和 validation set。AutoBencher 已经尝试让模型搜索"重要、新颖、困难"的评测数据（Li et al., 2025）。

模型可以参与出题，却不能独自认定自己进步。平时找错用的 active benchmark 可以不断变化；确认提升的 private validation 要与训练产线隔开，也不能让被评 checkpoint 提前看到。题目和答案是否可靠、有没有泄漏，还得靠独立生成器、环境证据或人工抽查。

> *模型可以自己出下一套题，进步则由独立证据确认。*

一套设计良好的 eval，本来就是基础模型团队迭代时最重要的基础设施之一。到了 RSI 阶段，它还要承担新的工作：从"数学更强""coding 更强"这类模糊目标出发，把模型当前的边界变成下一轮能够执行的任务和验证集。

Self-evaluation 的用处，是不断移动能力边界。Self-validation 则让模型参与设计检验，最后把裁决交给它无法提前迎合的证据。

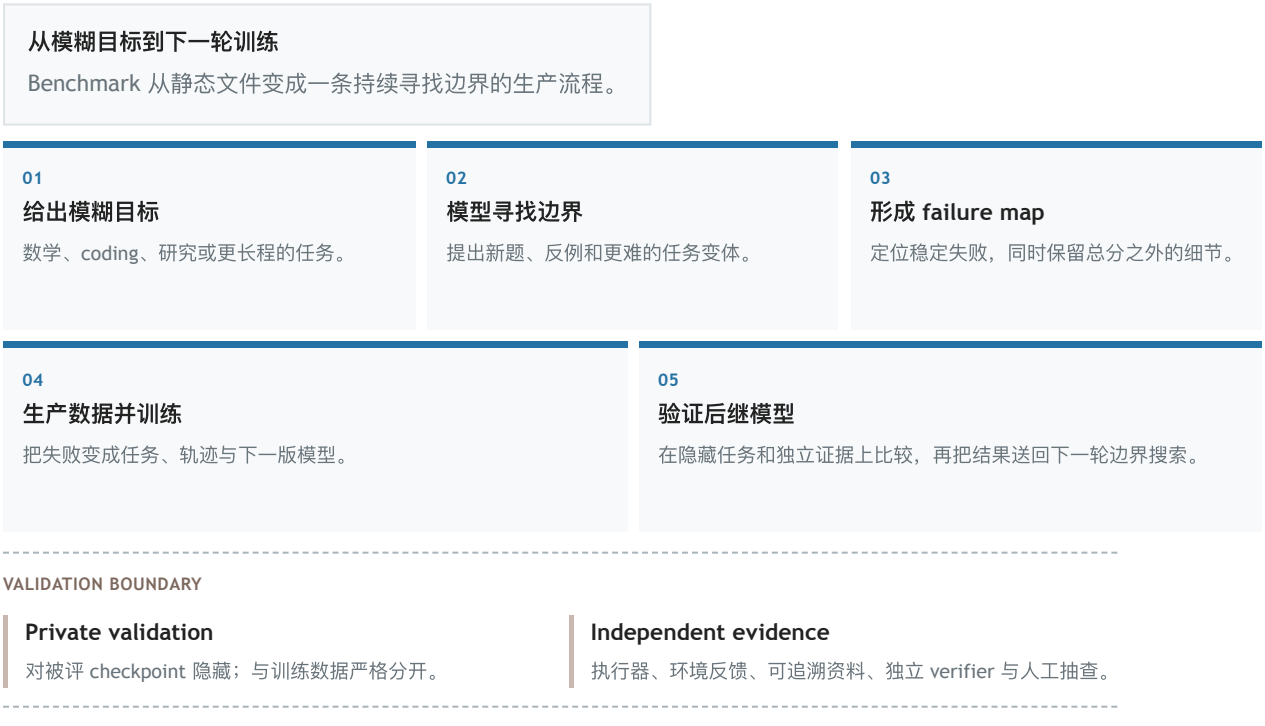

图 1. 01–05 构成一轮迭代；验证结果回到下一轮边界搜索，validation boundary 则阻止系统"自己出题、自己训练、再自己宣布胜利"。

每轮 eval 最有用的产物是一份 **failure record**。总分可以留着，但还要记下任务切片、最终错误、轨迹里最早出问题的位置、可能的原因，以及下一批数据准备补什么。同样是 timeout，背后可能是规划错了、工具没调通，也可能是上下文管理失效。只看最后的标签，会把这些问题混在一起。

对这类评估，我会看五层证据：

1. **Outcome**：任务最终是否完成，能否由执行器或明确 verifier 判断。
2. **Process**：失败最早从哪里开始，后面的错误是否只是连锁反应。
3. **Slice**：问题是否在一类新任务上重复出现；单个 bad case 提供的证据还不够。
4. **Prescription**：它能否导出下一批任务、反馈形式或 Harness 变化。
5. **Transfer**：干预以后，提升是否出现在未参与数据生产的 private validation 上。

2. Ladder：用小实验决定要不要爬下一阶

假设一个 coding Harness 一周能跑出一百万条轨迹，要不要全做？这个问题不能拍脑袋决定。合成数据最麻烦的地方就在这里：样本可以写得很漂亮，答案也可以是对的，训练到目标模型上却没有任何变化。

Ladder 说白了，就是先做便宜的小实验，再一档一档加钱。每次只放大一个东西，其余条件尽量不动。相对 control 的提升能够重复，才进入下一档。这样至少能在大规模生产之前看见收益和饱和。

Architecture ladder 常用来判断一个新结构能不能放大。做法是在几档递增的 compute budget 上，分别训练 candidate 和 matched baseline。数据、objective、评估方法，以及模型规模和 token 的分配规则都先固定。最后只看一件事：**candidate 的优势能否跨规模保持?**

Data ladder 把放大的对象换成数据。目标模型、训练配置和 eval 不动，只增加同一来源的数据量或覆盖，看每一批新数据还能换来多少 held-out gain。它要找的是边际回报开始消失的位置。

Synthetic-data ladder 又多了一层麻烦：数据是现场生产出来的。Producer、Harness、verifier、采样和 filter 一起决定数据长什么样；目标模型又决定这些轨迹能不能学进去。要让曲线可比，就得固定 **production recipe** 和 **target model**，每一档只增加通过验证、去重后的轨迹。产线或目标模型一变，旧曲线就只能作参考，实验要从便宜的档位重跑。

扩量还会反过来改变数据：通过率可能下降，重复越来越多，简单样本淹没长尾。同一批轨迹给两个模型训练，一个可能学会规划，另一个只记住措辞。SynthLLM 也观察到，synthetic data 的饱和区会随目标模型规模变化（Qin et al., 2025）。“能生成多少”和“这个模型该吃多少”，要分开问。

三种 Ladder 都从小实验开始。它们放大的对象不同，曲线失效的条件也不同：

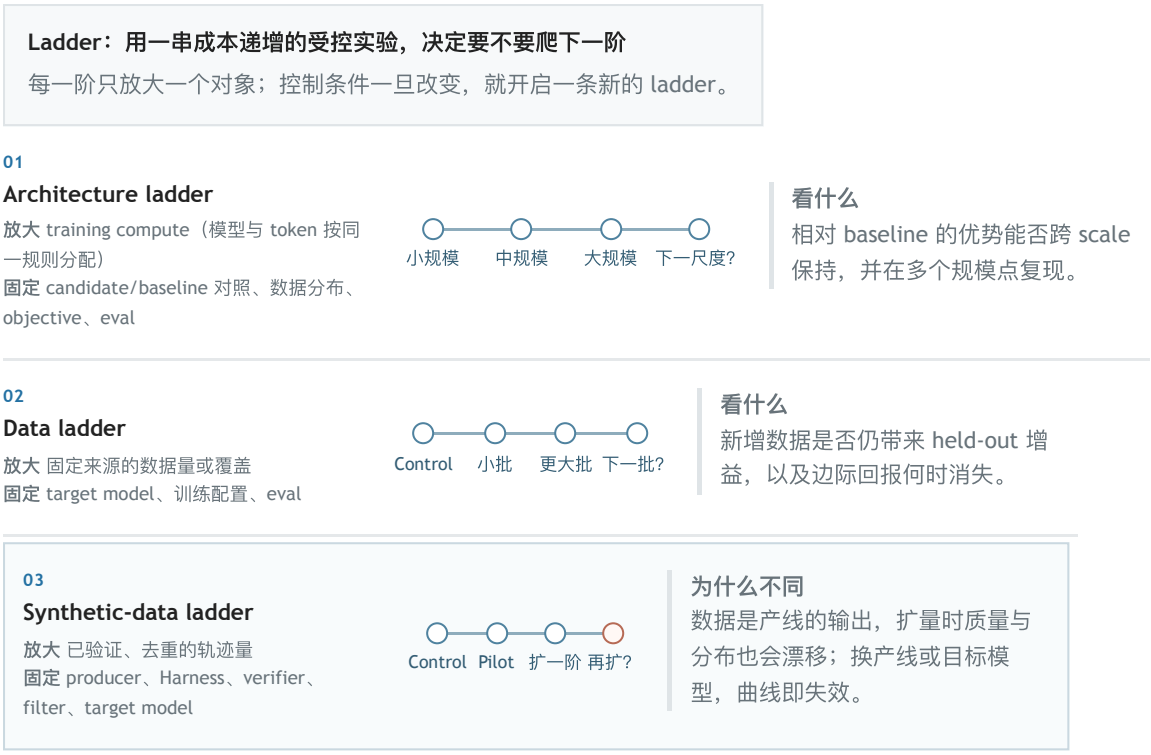

图 2. 每条 Synthetic-data ladder 都对应一条 production recipe 和一个 target model。

实际操作时，ladder 就是一串越来越贵的小赌注。先 audit 少量任务和 verifier，再从同一个 base checkpoint 做 pilot training。Held-out signal 能重复，就多做一档；信号接近噪声或成本阈值，就先复跑；跌到阈值以下，停下来改产线。

一个最小可用的实验设计

最小实验固定 base checkpoint、优化器、训练轮数和 held-out eval，只改变通过筛选、去重后的 trajectory 数量。数据变多，总训练 token 也跟着增加。如果 training-token budget 固定，测到的是等算力下的 coverage 或 mixture，不能和纯数据量 scaling 混在一起。每一档至少重复两次，顺便估计训练噪声。

每个点都记三本账。生产侧看成本、通过率和去重率；训练侧看有效 token、稳定性和行为变化；评估侧看目标切片、跨切片迁移和回归。相邻两档的增益能复现，才值得继续花钱。

在多个规模、模型和产线上复现之前，它只是 **ladder experiment**，还算不上可以外推的 scaling law。先用它管住投入、找到饱和点，已经很有价值。

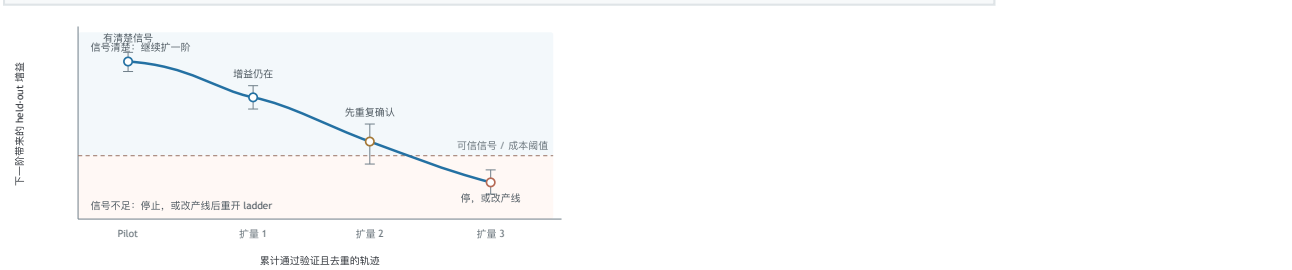

图 3. 决策规则示意。每个点都应从同一 base checkpoint 训练，并在未参与数据生产的 held-out 任务上重复测量；新的 production recipe 或 target model 需要重开一条曲线。

先看哪几个数

我会先看三个数：通过验证的比例，去重以后剩下的数量，以及每加一批不同轨迹，独立 eval 上涨了多少。前两个数算成本，最后一个数看收益。总生成量本身说明不了太多。

小批量训练有信号，再逐级增加合格且不重复的轨迹。每一级都从同样的模型起点出发，训练配置和评估集尽量不动，上一档留作对照。数据翻了几倍，独立任务却不再变好，或者通过率一路下降，产线大概已经接近饱和。

产量做到饱和点以前就够了。没有 Ladder，团队很容易先做完一批昂贵数据，再回来解释它为什么没用。

Ladder 属于产线与目标模型

同一批 synthetic data 放到两个模型上，结果可能差很多。因为每条数据都带着生产模型和 Harness 的习惯：怎么拆题、爱用什么工具、什么时候回退、会犯什么错。目标模型能不能吸收这些习惯，并没有通用答案。

换目标模型，或者改 producer、Harness、verifier、sampling policy、filter，都要先回到低成本档位。上一条曲线提供经验，不提供保证。Synthetic data 的 scaling law 难做，难就难在这里。

从 pilot 推到目标规模

Ladder 最终希望得到一组可以预测的曲线。先在小批数据和较小模型上跑低成本实验，再用少量更大规模的点做校准，估计同一条 production recipe 在目标模型上的收益、饱和位置和所需数据量。这样，大规模生产开始以前，团队已经知道大概要做多少数据，也能判断这条产线是否值得继续投。

跨模型预测需要一组经过校准的 ladder，不能拿一个小模型上的单点直接外推。模型规模变化以后，吸收效率和饱和位置都可能移动；较大模型上的校准点仍然要保留。

3. 轨迹需要一条靠谱的产线

数据一直有人在背后设计。去哪里找、留下什么、怎么标注、先学什么，都是人的选择。原料即使来自互联网，进模型以前也走过一条人为的管线。

公开网页不再提供主要增量以后，数据生产会从“收集文档”转向“记录工作”。专家完成一件事，可能要用私有工具，来回沟通，做很多局部判断，也会失败和恢复。如果只保存输入和最终答案，中间最值得学的部分就丢了。新的数据基础设施得把环境、行动、反馈和结果一起记下来。

现在不少 synthetic data pipeline 还是 prompt + model + filter。问答做得很快，过程却很薄。模型怎么搜、怎么试、看到报错后怎么改、什么时候发现走错了，往往更值得留下。

从 prompt 到产线

写代码要跑测试；做数学会检查证明、找反例、逐步加难；跑 agent 不能丢掉环境状态和工具返回。这些常识散在 prompt、脚本和研究者脑子里。Harness 把它们写进模型每次工作都要面对的环境。

这里的 Harness 包括 prompt、工具、上下文管理、工作流、持久状态、权限和验证逻辑。模型看见什么、能做什么、怎么保存结果、如何检查自己，都由它决定。最后收集到什么样的轨迹，也由它决定（Weng, 2026）。

能追踪
任务、生产模型、Harness、工具和 verifier 都有清楚的版本。

看得到过程
留下必要的状态、行动、工具反馈、失败和修改。

能验证
成功尽量来自执行结果或独立判断，也能解释为什么丢掉一条轨迹。

有差异
控制不同 Harness 和模型的比例，去掉重复轨迹与同一种表达。

能回测
每批数据都回到 Ladder 和独立 eval，测量目标模型学到了多少。

Harness–Evolve 之后，再 SFT

Harness–Evolve 的做法很直接。让模型在有工具、有反馈的环境里多试几次，允许它走不同路径，也允许失败以后回来改。跑完以后，再挑出正确、有代表性、彼此不同的完整过程。

这些 trajectory 最后送进 SFT。希望模型下次碰到类似问题时，更容易走上靠谱的路径：该验证就验证，失败以后会改，需要工具时知道怎么用。

STaR 提供了一个直接的先例：生成 rationale、保留能得到正确答案的路径，再把它们训练回模型（Zelikman et al., 2022）。Harness–Evolve 把候选空间从“文本推理过程”扩到带状态的 agent trajectory，工具返回、文件变化、执行错误、回退和重新规划都成为数据的一部分。DGM 和 Self-Harness 主要优化 agent 或 harness；本文的产物是训练数据，目标模型在最后的 SFT 阶段更新（Zhang et al., 2025; Zhang et al., 2026）。

Harness
工具、上下文、反馈、记忆、验证

Evolve
多次尝试、变体、回退、修改、选择

Trajectory data
留下完整、可靠而且有差异的过程

SFT
把这些过程训练回目标模型

图 4. 四个方框是同一条数据产线的四个阶段。Evolve 改善候选轨迹，最后的 SFT 再把这些过程写回目标模型。

为什么需要很多种 Harness

一个 Harness 会长出一种固定的轨迹。强调测试驱动的 coding Harness，会留下“运行—报错—修改”的路径；强调批评与重写的 Harness，会留下更多自我检查；强调搜索的 Harness，则会留下分支比较。它们都可能有用，也都可能用久了变成套路。

可以同时跑几种 Harness，再用 Ladder 看目标模型更能吸收哪一种，什么混合比例最好。实验会多一些，但训练集不容易被一种 pattern 填满。

Harness 也有自己的 horizon

长线任务会逼着 Harness 改造。今天的很多 harness code 默认任务能在一次运行、一个 context 里结束；工具同步返回，verifier 很快给结果，最后有一个状态表示成功。这套假设应付几十分钟的 coding task 还行。实验跑上几天、数据工程跨过多轮训练，或者研究反馈很晚才来时，它就撑不住了。

METR 用“人类专家完成同一任务所需的时间”来描述 agent 的 task-completion horizon。它衡量的是任务难度，与 agent 实际跑了多久分开计算。这个指标抓住了一个常见问题：模型会很多局部技能，不代表它能把这些技能可靠地串成一条长链（METR, 2026）。任务越来越长以后，状态丢失、上下文膨胀、错误累积和恢复失败，都会变成瓶颈。

面向 RSI 的 Harness 还缺几样基础设施：

- **持久状态**：实验、代码、数据版本和未完成事项需要保存到 context 之外；中断以后要能从 checkpoint 恢复。
- **分层目标**：长任务要拆成可验证的阶段，同时保留阶段之间的依赖，避免只优化眼前的小分数。
- **异步执行**：训练、评测和数据生成可能跑数小时或数天，Harness 要能启动、监控、取消和重新接管后台任务。
- **延迟反馈**：最终 reward 很晚才出现时，需要保存中间证据，并把失败追溯到最早产生偏差的步骤。
- **可演化但有边界**：模型可以修改 workflow、context policy 和工具组合，但 verifier、权限与审计日志不应被同一个改进回路随意改写。

最近的长程 agent 工作开始重视 compact state、checkpoint、verifier-backed state transition 和 targeted recovery，不再把完整交互历史反复塞回 prompt（Wu et al., 2026）。Harness–Evolve 也会碰到同一件事：轨迹在变长，承载轨迹的 Harness 也得升级。否则，产线永远只能生产当前 horizon 以内的数据。

人已经积累了哪些合成经验

改写与变换	从已有材料出发，做重写、翻译、格式转换、难度调整和任务化。
教师生成	从少量 seed 出发，由更强模型生成题目、答案、反馈或偏好数据，再蒸馏给目标模型。
搜索与验证	一次生成多条候选，用代码执行、证明检查、规则或 verifier 留下可靠样本。
环境交互	让模型在工具和环境中行动，记录状态、反馈、失败、回退与恢复，得到完整 trajectory。
课程与配比	控制难度、题型、工具、解法和生产模型的比例，让数据跟着目标模型的能力边界移动。

过去的 synthetic data 大多落在这几类里。Harness 轨迹属于“环境交互”：数据里保存的不只是题目和答案，还有模型看到的状态、调用过的工具、得到的反馈以及中途如何改路。Harness–Evolve 又在此基础上加入多次尝试和筛选，把一条交互过程做成可训练的轨迹。

真正有用的先验，是面对一个新目标时知道该选哪种生产方法。知识覆盖不足，可以先改写和扩展真实材料；答案容易验证，可以多采样再筛选；能力藏在工作过程里，就要搭 Harness 记录轨迹。模型以后也要学会做这种选择：先读目标和 failure map，再设计任务、环境、verifier 与数据配比。

生产模型很容易复制自己。行动、判断和筛选都交给同一个模型，Evolve 可能只会把原来的 pattern 越放越大。执行工具、独立 verifier、不同来源的生产模型、真实数据锚点和少量人工抽查，可以让产线少一点回音。

筛选本身也会制造 shortcut。Verifier 只看最终答案，可能留下碰巧答对的短路径；一味偏好长轨迹，又会奖励空转。筛选时最好同时看 outcome、过程是否完整、轨迹是否新鲜以及成本。被拒绝的轨迹和原因也要留下，它们正好能告诉下一轮该怎么改 Harness、怎么出题。

这条 loop 怎么转起来

把前面的东西连起来，就是一条普通的工作流：eval 找问题，Harness–Evolve 做轨迹，SFT 把轨迹训练回去，Ladder 决定要不要继续扩量。

上一轮的失败决定下一轮做什么数据。模型变强以后，旧题会失效，旧 Harness 也会饱和，题目、环境和数据都要跟着换。

01
先说想补什么
不必一开始就有精确分数，但要说清大致方向。

02
出一批新题
不断换题，找到当前模型会稳定卡住的地方。

03
设计 Harness
把人的领域经验写进工具、反馈、验证和探索规则。

04
跑出轨迹
让模型尝试、失败、修改，留下几种不同的完整过程。

05
筛选，再 SFT
只把可靠且有差异的轨迹训练回目标模型。

06
用 Ladder 验证
从小到大看吸收、迁移和饱和，再决定要不要继续。

这条 loop 有没有用，最后看下一版模型是否稳定变强。其它指标都只是过程证据。

目标怎么定、Harness 怎么写、证据是否可信、什么时候停，现在仍离不开人的判断。先把其中一段做成能测、能复现的实验，已经足够开始。

References

1. Villalobos et al. (2024), *Will we run out of data? Limits of LLM scaling based on human-generated data*.
2. Singh et al. (2024), *Evaluation data contamination in LLMs: how do we measure it and (when) does it matter?*.
3. Zelikman et al. (2022), *STaR: Bootstrapping Reasoning With Reasoning*.
4. Qin et al. (2025), *Scaling Laws of Synthetic Data for Language Models*.
5. Zhang et al. (2025), *Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents*.
6. Zhang et al. (2026), *Self-Harness: Harnesses That Improve Themselves*.
7. Weng (2026), *Harness Engineering for Self-Improvement*.
8. DePue (2026), *A Stargate for Data*.
9. METR (2026), *Task-Completion Time Horizons of Frontier AI Models*.
10. Wu et al. (2026), *StructAgent: Harness Long-horizon Digital Agents with Unified Causal Structure*.
11. Akhtar et al. (2026), *When AI Benchmarks Plateau*.
12. Center for AI Safety & Scale AI (2025), *Humanity's Last Exam*.
13. Li et al. (2025), *AutoBencher: Creating Salient, Novel, Difficult Datasets for Language Models*.