课程笔记

Open-ended and AI-generating Algorithms in the Era of Foundation Models

cnfjlhj & Codex

2026-06-07

视频作者/频道：Schwartz Reisman Institute
发布日期：2025-10-02
视频时长：01:30:13
视频链接：https://www.youtube.com/watch?v=gIHAVTj9fjo

目录

1 阅读导引：这场 talk 到底在讲什么

这场 Schwartz Reisman Institute seminar 的主线可以压缩成一句话：Jeff Clune 想说明，foundation models 不是只让我们多了一个强大的文本模型，而是让长期困扰 open-ended algorithms 与 AI-generating algorithms 的几个关键缺口第一次变得工程可操作。

Clune 在 talk 开头先铺垫三类算法：

- **Quality-diversity algorithms**：不只找一个最优解，而是找一组彼此不同但各自高质量的解。
- **Open-ended algorithms**：希望系统越运行越能持续产生新环境、新问题、新行为和新能力。
- **AI-generating algorithms**：希望 AI 系统本身能够生成更好的学习算法、agent 架构和训练环境。

然后他把这些旧思想接到 foundation models 时代的新机会：LLM 可以作为任务生成器、interestingness 评估器、agentic system 改写器、代码搜索器、科学想法生成器，也可以作为安全监控链条的一部分。

图 1: Talk 的主题框架：quality diversity、open-ended algorithms、AI-generating algorithms，以及 foundation models 带来的新机会。[1]

这份笔记的读法

不要把这场 talk 当成若干论文的松散介绍。更好的读法是把它看成一条算法谱系：目标欺骗说明为什么直接优化会失败；quality diversity 给出 archive 与 goal switching；open-ended algorithms 把 archive 扩展到任务和环境；foundation models 让“生成任务、评估有趣性、改写 agent、自动做科学”变成可运行系统。

[1] 视频画面时间区间：00:04:20–00:04:50。

1.1 本章小结

这场 talk 对 self-improving coding agents 的价值，不在于某个单独系统，而在于它给出一个可复用闭环：**generate** 新对象，**evaluate** 质量与新颖性，**archive** stepping stones，**mutate** 或改写已有对象，再用 safety monitor 限制危险方向。

2 目标欺骗：为什么“太想达成目标”反而会失败

Clune 的第一个核心论点来自 novelty search 与《Why Greatness Cannot Be Planned》一脉：对于极难问题，直接奖励“离目标更近”可能把搜索困在局部最优。迷宫例子中，如果 agent 只按到目标的欧氏距离拿奖励，它会不断撞向看似更近但实际封闭的墙；反而是忽略最终目标、优先探索新区域的策略更容易走出迷宫。

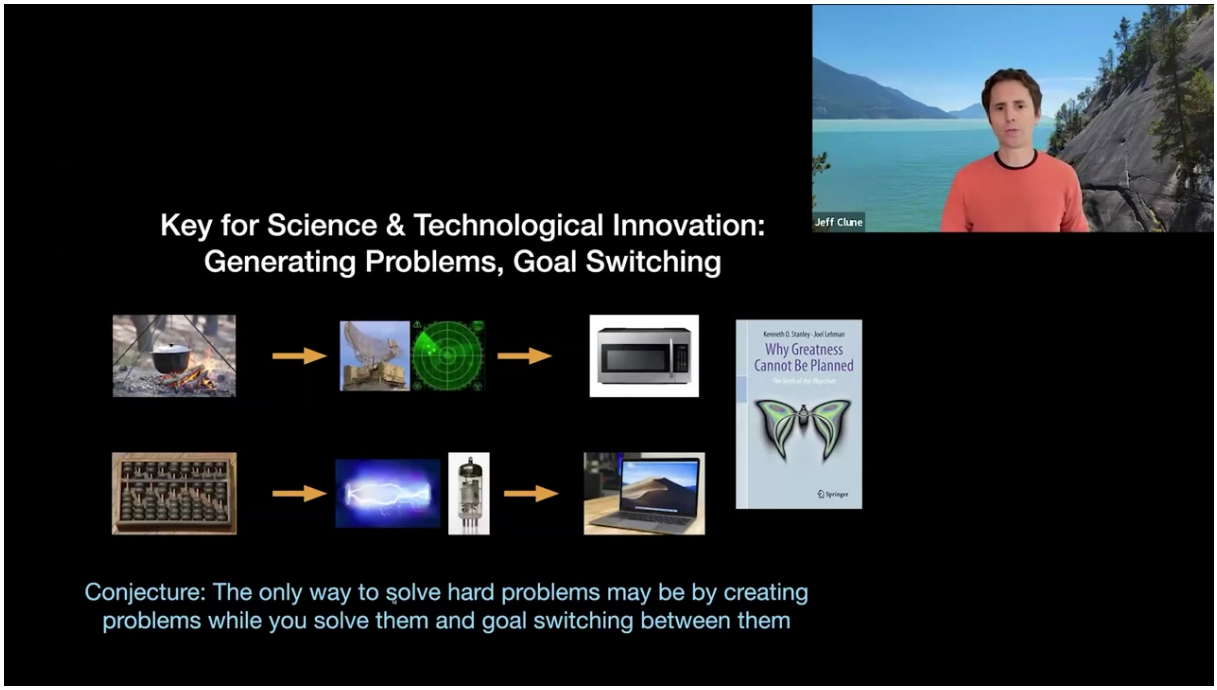


图 2: Clune 用历史创新例子说明 goal switching：科学和技术突破常常来自中途发现的新问题，而不是线性追逐原目标。[2]

误解：open-ended search 不是随机乱试

Clune 反复强调，忽略直接目标并不等于随机搜索。随机按按钮无法解决 Montezuma's Revenge。关键在于“有原则地保留多条有潜力路线”，也就是既不把所有搜索压到一个目标上，也不无差别扩散到整个空间。

2.1 goal switching 的含义

Goal switching 指的是：当搜索过程中发现一个和当前目标不完全一致、但明显有趣或可积累的新行为时，不要因为它不能立刻提升当前目标函数就丢掉它。机器人学里的例子是，本来想学双足

[2] 视频画面时间区间：00:07:15–00:07:45。

行走，却意外发现了单脚平衡、爬行或转身。传统优化可能把这些当成失败样本；open-ended search 会把它们放入 archive，作为后续 stepping stones。

核心压缩

伟大目标的问题不只是“难”，而是它的中间路径通常不可预先规划。一个好的 search system 必须允许自己在发现新机会时改变正在优化的问题。

2.2 本章小结

目标欺骗给后续所有算法铺了地基：如果只允许一个 objective 支配搜索，系统会过早收缩；如果能保留多样、高质量、可复用的中间对象，就可能走出人类事先无法设计的课程路径。

3 Quality Diversity：MAP-Elites 的 archive 思想

Quality-diversity algorithms 的目标不是输出一个最优解，而是输出一个由多种类型高质量解组成的集合。MAP-Elites 是 Clune 重点介绍的代表算法，因为后续 POET、OMNI、ADAS、DGM 等系统都在不同层面复用同一种 playbook。

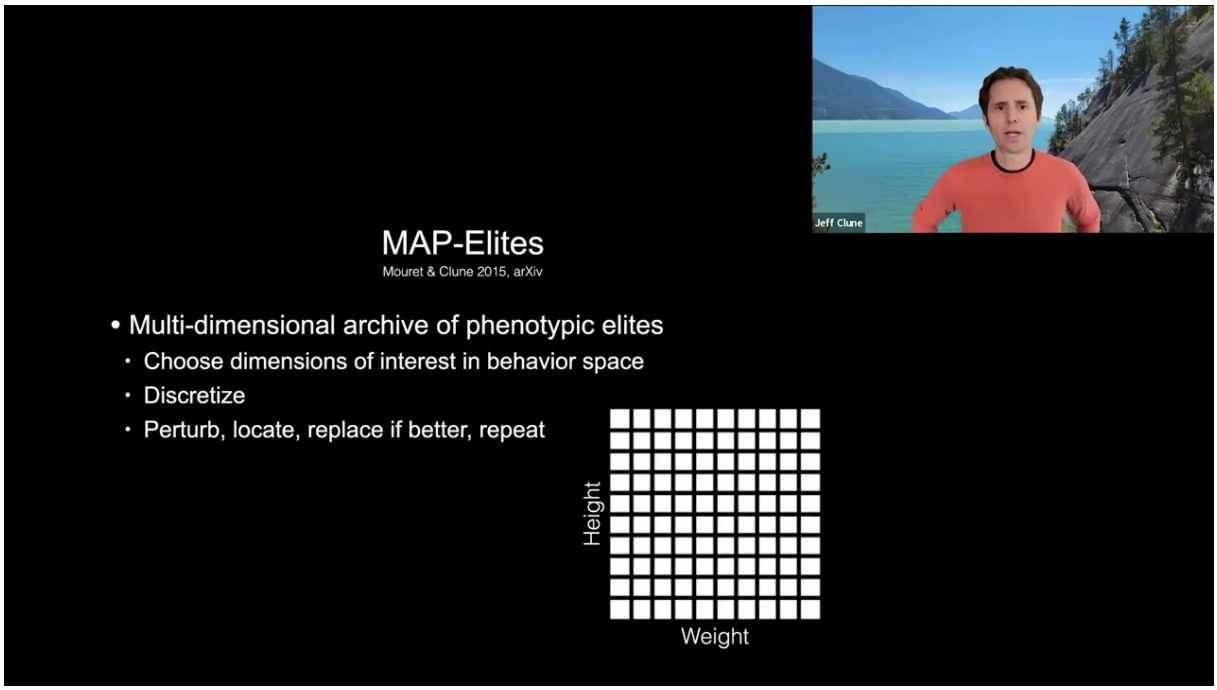


图 3: MAP-Elites 的基本结构：选择行为维度，将搜索空间离散成 cell，每个 cell 保留该类型下目前最高质量的 elite。[3]

MAP-Elites 的机制可以写成一个非常简单的循环：

```
archive = {}
while budget remains:
    parent = sample_from_archive_or_random()
    child = mutate(parent)
```

[3]视频画面时间区间：00:08:45–00:09:15。

```
    score, behavior = evaluate(child)
    cell = discretize(behavior)
    if cell not in archive or score > archive[cell].score:
        archive[cell] = child
```

Listing 1: MAP-Elites 的概念性伪代码

其中最关键的是两个空间的分离：

- **behavior space**：用来描述解“属于哪一类”，例如机器人高度、重量、步态类型。
- **performance measure**：用来判断同一类内部哪个解更好，例如前进速度。

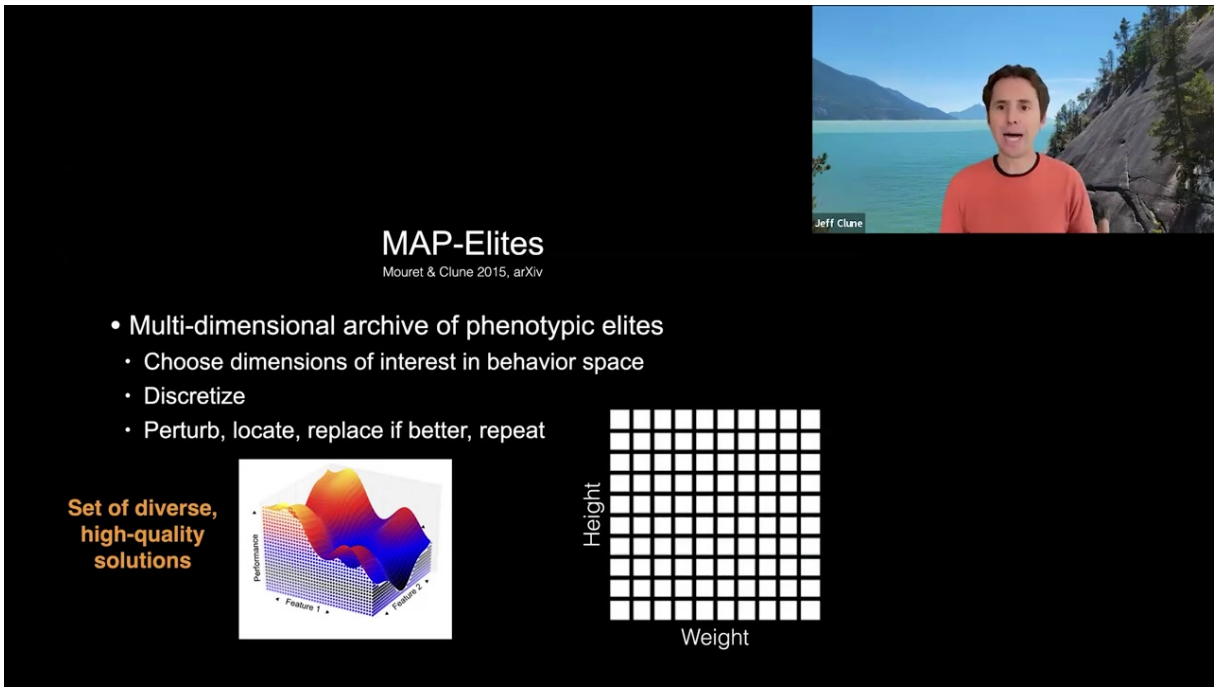


图 4: 同样计算预算下，MAP-Elites 相比直接优化和简单多样性奖励，可以更系统地覆盖搜索空间并保留高质量解。[4]

3.1 为什么 archive 比单点最优更重要

如果一个算法只输出当前最高分的一个解，它会丢掉大量“暂时不是最好，但以后可能成为关键垫脚石”的对象。MAP-Elites 保留的是多维 archive：每一个 cell 都是一种已发现类型的最佳代表。这种结构使系统天然具备三种能力：

- **覆盖能力**：知道哪些区域已经探索，哪些区域还空着。
- **复用能力**：后续 mutation 可以从不同类型的 elite 出发。
- **诊断能力**：研究者可以观察某些区域为何表现差，某些区域为何形成多个局部最优。

[4] 视频画面时间区间：00:10:15–00:10:45。

3.2 counterintuitive curricula

Clune 用 lineage plot 说明，最终最高质量解的祖先往往不是沿着同一目标单调前进，而是在 search space 中绕了一条很长的曲线。这个现象对应他的生物类比：如果你想进化出人类智能，直接奖励“更像人类”不会奖励扁虫阶段；但自然演化恰恰需要这些不直观的阶段。

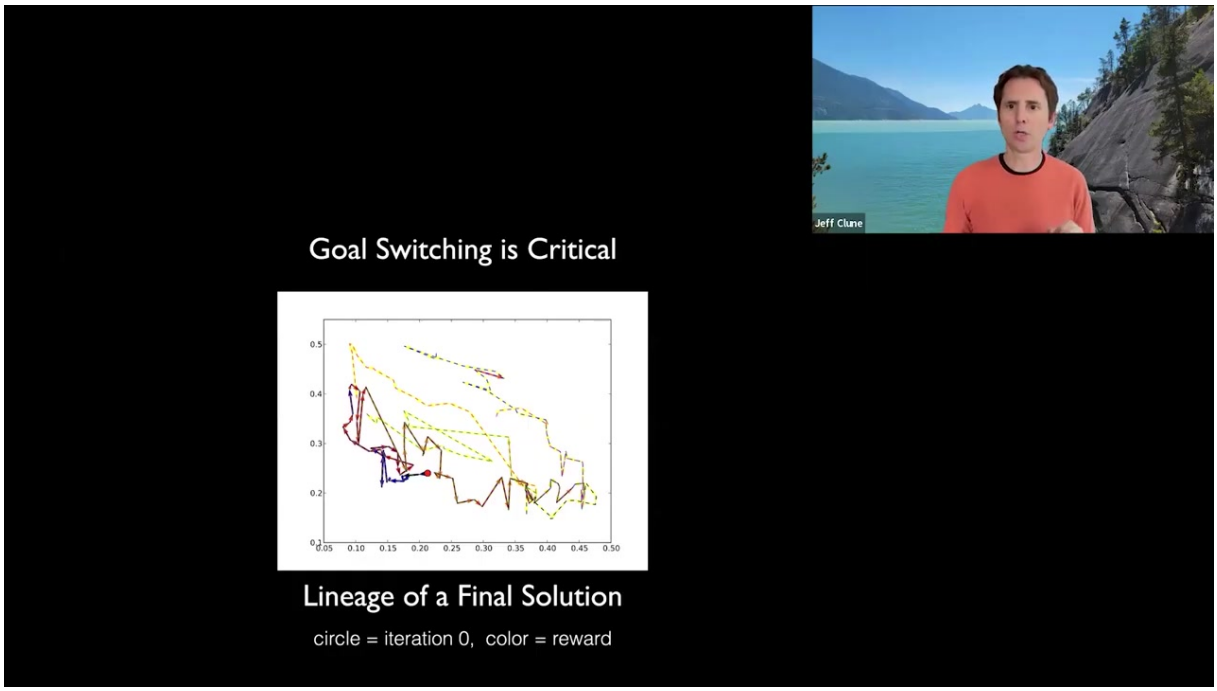


图 5: 最终高质量解的祖先路径会穿越多个行为区域，说明 goal switching 和反直觉课程是搜索成功的一部分。[5]

3.3 Go-Explore：同一个思想如何处理稀疏奖励

Go-Explore 把 quality diversity 的思想带到 hard-exploration RL。其关键不是随机探索，而是学习到达尽可能多状态的高质量路径。对于 Montezuma's Revenge 这种长时间没有 reward 的游戏，传统 RL 很容易没有信号；Go-Explore 通过 archive 保存已到达状态，再从这些状态继续探索。

[5] 视频画面时间区间：00:11:45–00:12:15。

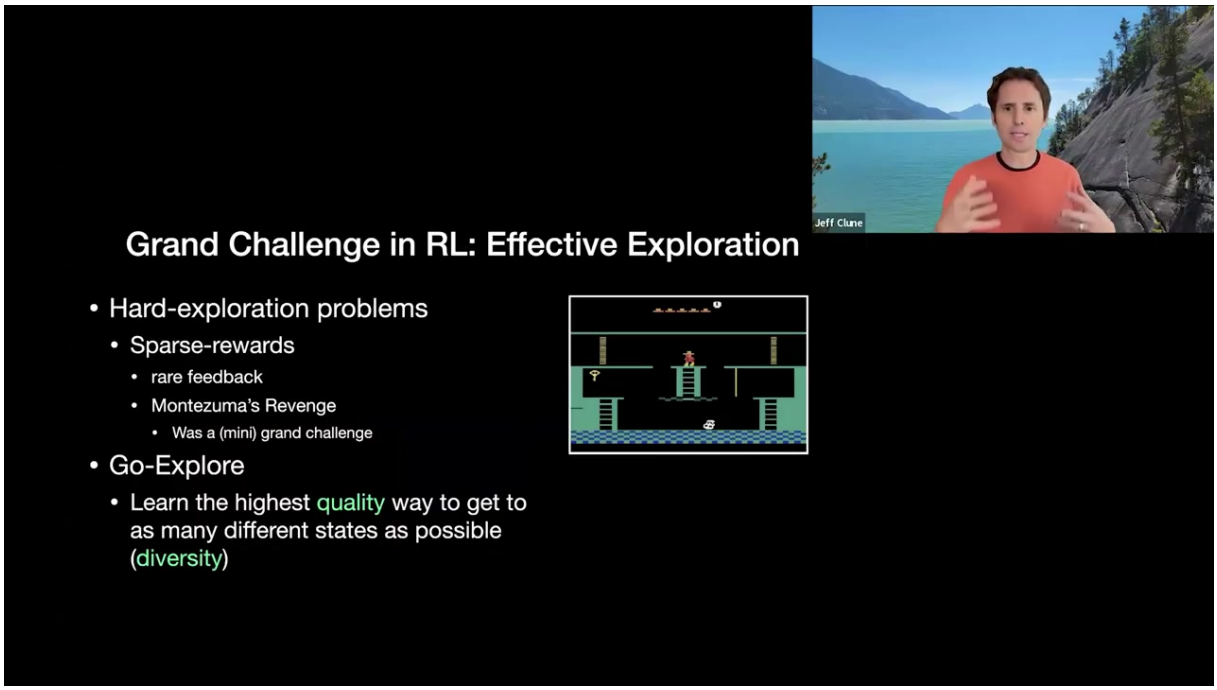


图 6: Go-Explore 将“到达许多状态的高质量路径”作为 quality-diversity 目标，解决稀疏奖励环境中的探索瓶颈。[6]

3.4 本章小结

Quality diversity 给 open-endedness 提供了最基础的数据结构：archive。没有 archive，系统只有一条优化轨迹；有了 archive，系统可以保留多个方向、多种策略和多个未来可能复用的 stepping stones。

4 从单环境到 open-ended algorithms

Quality-diversity algorithms 仍然有一个局限：它们通常在单一环境中运行。Clune 接着引出 open-ended algorithms：系统不只要在一个固定环境里找多样高质量 agent，还要不断生成新环境、新问题和新学习挑战。

[6] 视频画面时间区间：00:13:15–00:13:50。

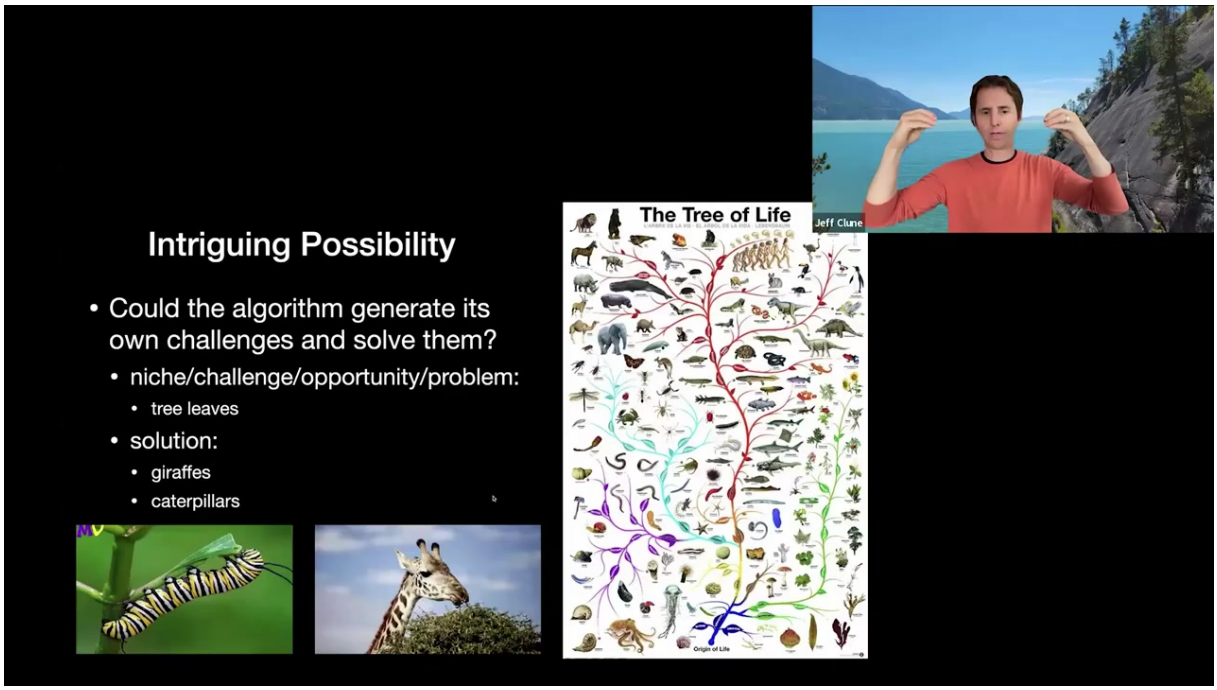


图 7: Open-ended algorithms 的目标是持续产生新的问题和机会。Clune 用演化与人类文化作为两个现实世界里的 open-ended process。[7]

4.1 自然演化和人类文化

Clune 认为自然演化和人类文化是我们已知的两个 open-ended process。演化产生了多样生命形式，并仍在继续；科学文化也是如此，每一个发现都会打开新的问题，每一种技术都会创造新的任务。这个观察导向一个极端测试：如果给一个算法十亿年计算，它是否仍值得继续运行?

open-endedness 的标准不是短期分数

如果一个系统很快达到某个 benchmark SOTA，然后停止产生新东西，它不是 open-ended。Open-endedness 要求“解决问题”本身能继续产生新的问题、机会和环境。

4.2 POET：环境和 agent 成对演化

POET, 即 Paired Open-Ended Trailblazer, 是 Clune 介绍的早期重要系统。它同时维护 environments 与 agents：

- 定期 mutation 生成新环境。
- 只保留对当前 agent 集合“不太容易也不太难”的环境。
- 允许 agent 从一个环境迁移到另一个环境。
- 如果某个 agent 在新环境中表现最好，就把它作为该环境的 elite。

[7] 视频画面时间区间：00:17:45–00:18:15。

图 8: POET 同时生成环境和解决这些环境的 agent，通过环境 mutation 与 agent transfer 形成反直觉课程。[8]

Clune 强调, POET 里最重要的不是 “生成了很多环境”, 而是直接优化和手工直线路径 curriculum 都无法解决一些最终环境。系统需要那些绕路的、看似不相关的中间环境作为 stepping stones。

4.3 本章小结

Open-ended algorithms 把 archive 的单位从 “同一环境中的多样解” 扩展为 “环境、agent、任务和学习挑战的共同生态”。对 self-improving coding agent 来说，这意味着不能只保存模型 checkpoint，还要保存任务、失败模式、工具、prompt、代码修改和 regression suite。

5 Foundation models 带来的关键变化

Clune 认为 foundation models 让 open-endedness 进入新阶段，是因为它们能填补过去很难工程化的几个缺口：大规模任务空间、interestingness 判断、世界模型、互联网行为先验、代码级 agent 改写。

[8] 视频画面时间区间：00:19:15–00:19:50。

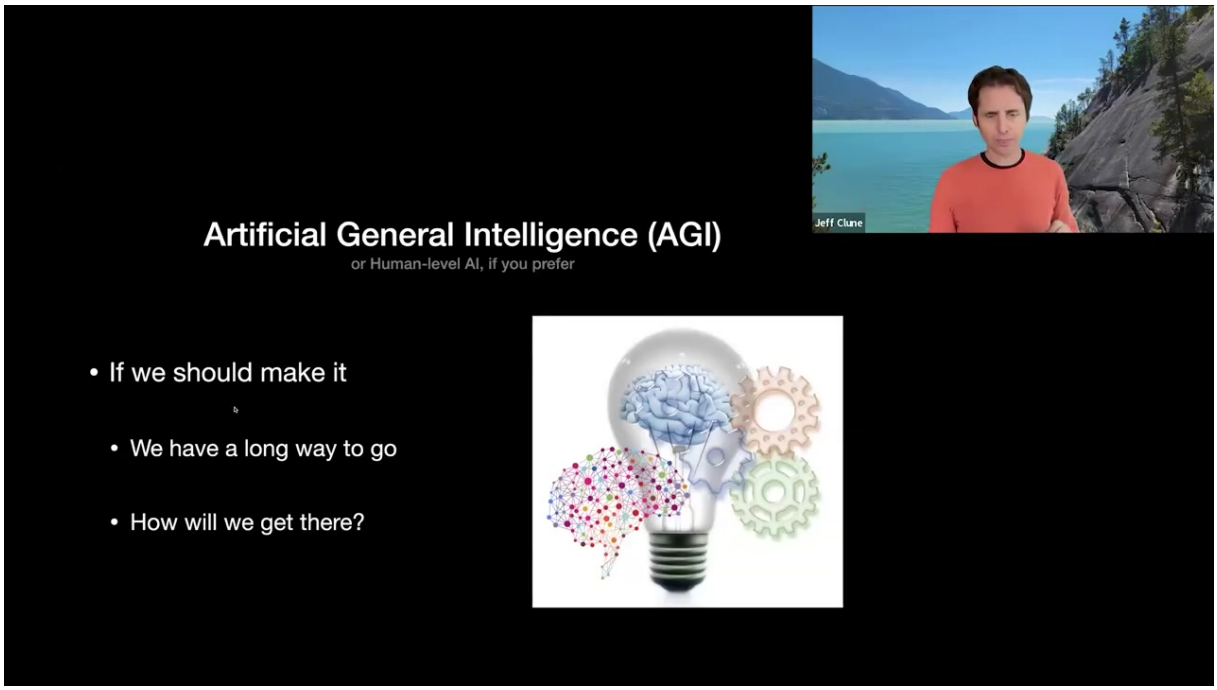


图 9: AI-generating algorithms 的宏观问题：如果目标是 AGI，我们应如何制造能持续生成更强 AI 的系统。[9]

5.1 OMNI：用 foundation model 近似“人类觉得有趣”

Open-ended system 必须避免两类坏任务：一类太难、太简单或不可学；另一类虽然可学但无聊、重复、没有长期价值。学习进度可以帮助处理第一类，但第二类需要判断“interestingness”。

OMNI，即 Open-endedness via Models of human Notions of Interestingness，的核心想法是：foundation models 在预训练中吸收了大量人类关于“什么值得注意、什么算新颖、什么算有趣”的隐式信号。于是系统可以问模型：给定 archive 中已掌握任务，这个新任务是否值得探索？

[9] 视频画面时间区间：00:22:10–00:22:45。

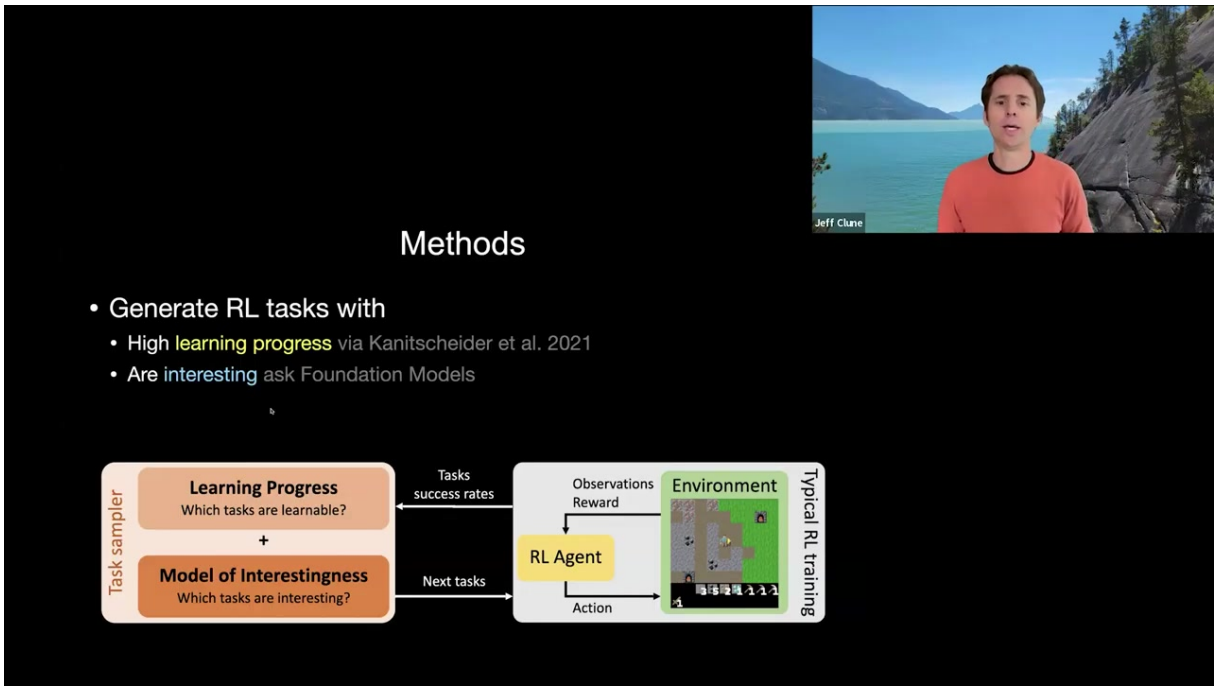


图 10: OMNI 将 learning progress 与 foundation model 的 interestingness 判断结合，用来过滤任务空间中的无聊或无效候选。[10]

OMNI 的研究价值

OMNI 的关键不是 “LLM 会打分”，而是把不可形式化的 interestingness 转化为一个可查询、可迭代、可和 learning progress 组合的模块。这对 coding benchmark 生成尤其重要，因为难题不等于好题；好题必须既可学，又能打开新策略。

[10] 视频画面时间区间：00:28:10–00:28:50。

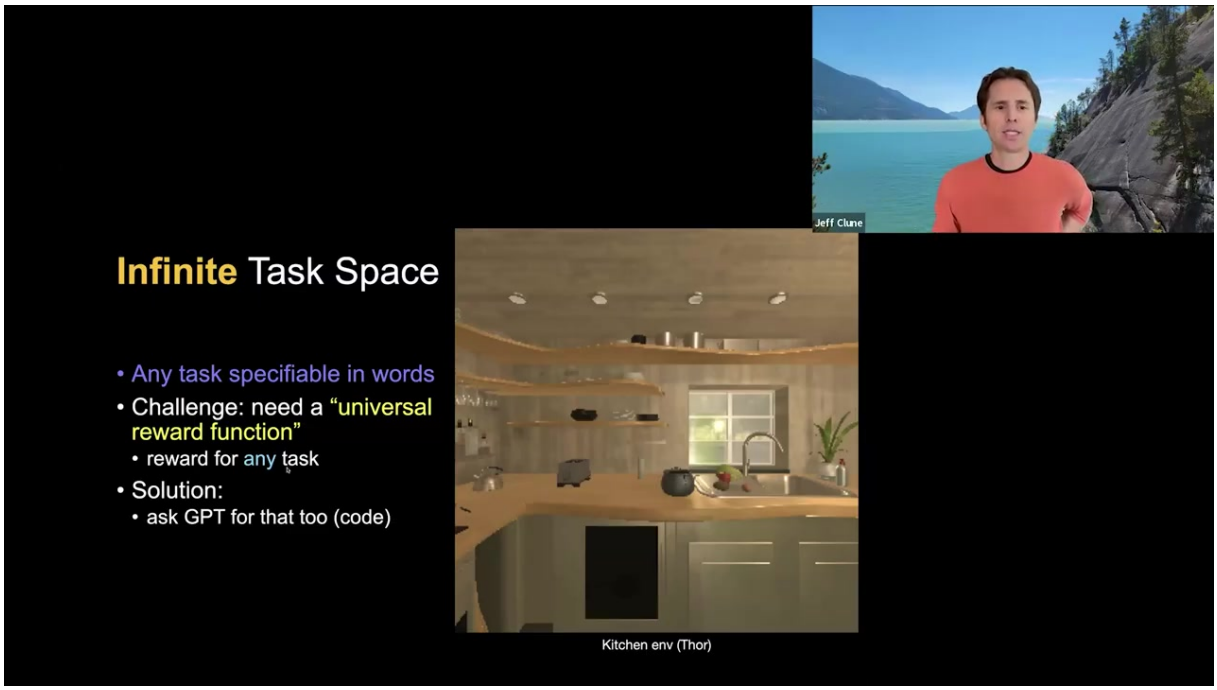


图 11: OMNI 在更大任务空间中比 uniform sampling 和只看 learning progress 更能维持有效学习。[11]

5.2 Genie：foundation world model 作为 Darwin-complete 环境

Clune 另一个重要观点是 "Darwin complete task space"：如果任务空间本身足够通用，系统就不再被固定 simulator 限死。Genie 类 world model 可以从文字或视觉 prompt 生成可交互世界，并根据 agent action 推进状态。它还不完美，但 Clune 的观点是：这会是它最差的时候，未来版本会更强。

[11] 视频画面时间区间：00:29:40–00:30:15。

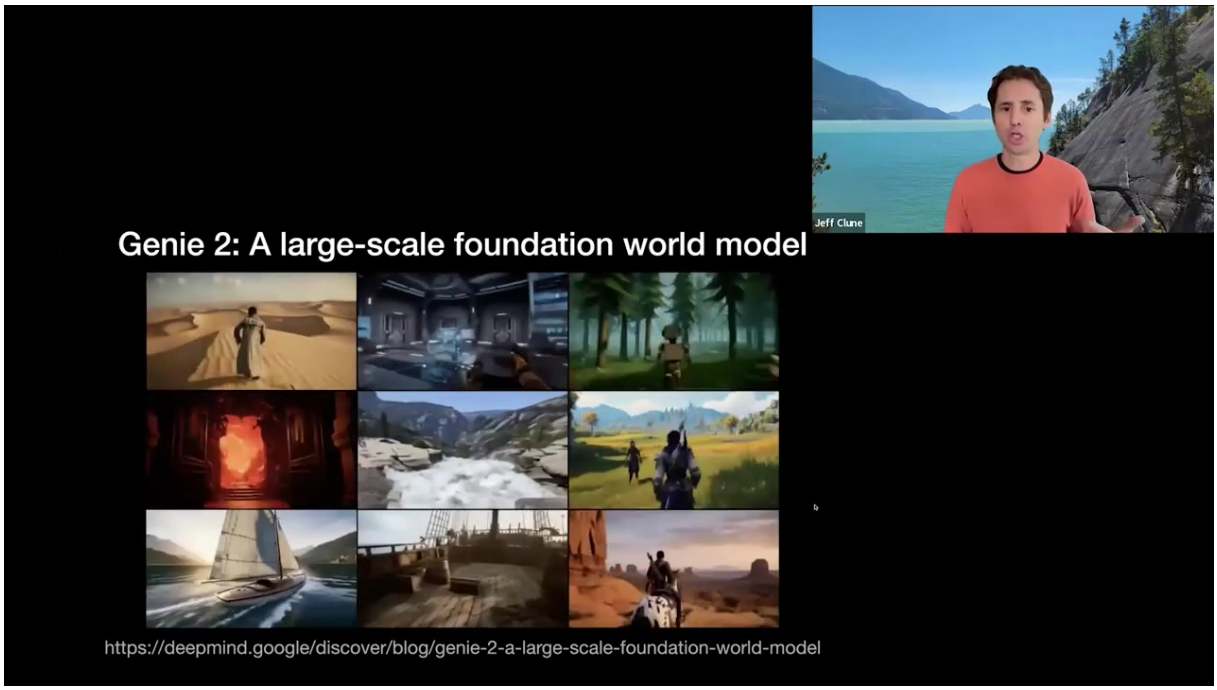


图 12: Genie 被视作 foundation world model：世界、状态转移和交互反馈都由模型生成。[12]

5.3 VPT：互联网视频作为行为先验

Video Pre-Training 解决的是 sample efficiency。传统 RL agent 在新环境中像随机按按钮一样从零探索；但人类和 foundation-model era 的 agent 可以从互联网视频、教程、行为轨迹中学到“世界中通常有什么、动作如何导致变化、奖励可能在哪里”。在 Minecraft diamond tool 任务中，VPT 展示了预训练行为先验对长程任务的重要性。

[12] 视频画面时间区间：00:38:40–00:39:20。

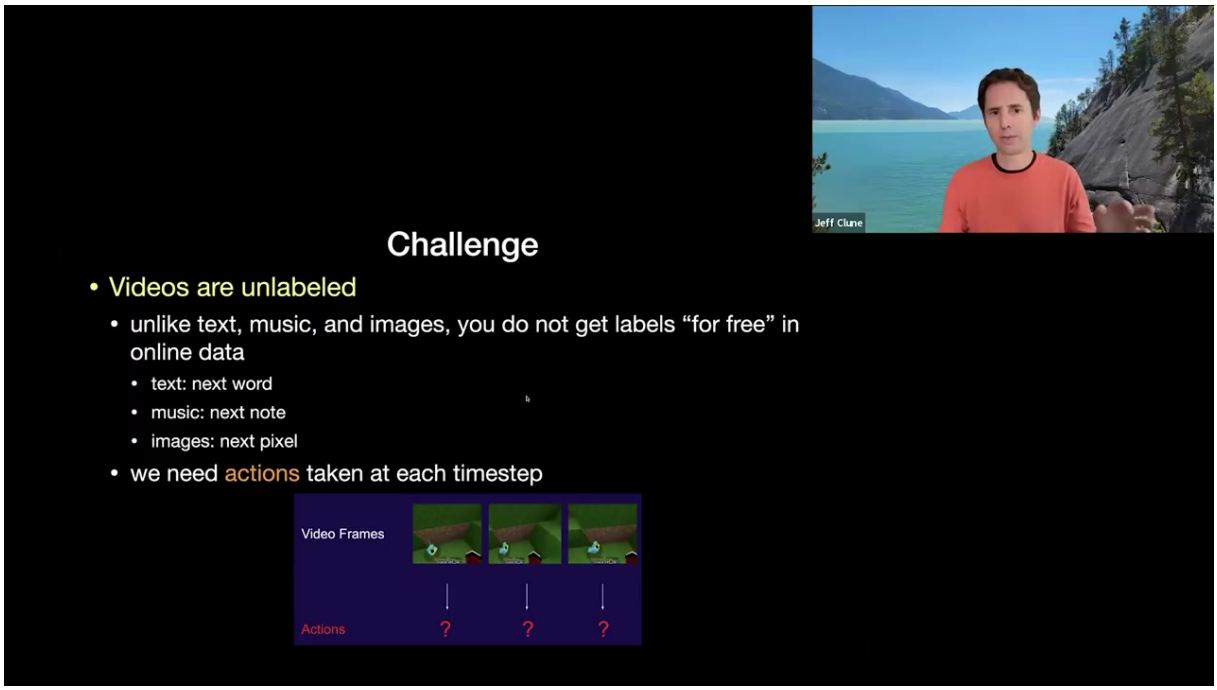


图 13: VPT 的挑战：互联网视频通常没有动作标签，需要从视频观察中恢复行为学习信号。[13]

5.4 本章小结

Foundation models 在这场 talk 中扮演四种角色：任务生成器、interestingness 判断器、世界模型、行为先验。它们并不自动解决 open-endedness，但让过去难以实现的 search space 和 evaluator 变得可运行。

6 从 agentic system 搜索到自我改写

Talk 后半段转向 AI-generating algorithms 的另一个核心支柱：不仅生成环境，还要生成更好的 agentic systems。这里的对象不再只是 neural network weights，而是 prompts、tool use、planning loop、memory、reflection、代码和 scaffold。

6.1 ADAS：自动设计 agentic systems

ADAS, 即 Automatic Design of Agentic Systems, 把 agentic system 表示为代码空间中的对象。系统维护一个 archive，从现有 agentic system 中取出一个，让 foundation model 修改它，然后评估新系统是否高质量且 interestingly different。如果通过，就放入 archive。

[13] 视频画面时间区间：00:43:10–00:43:50。

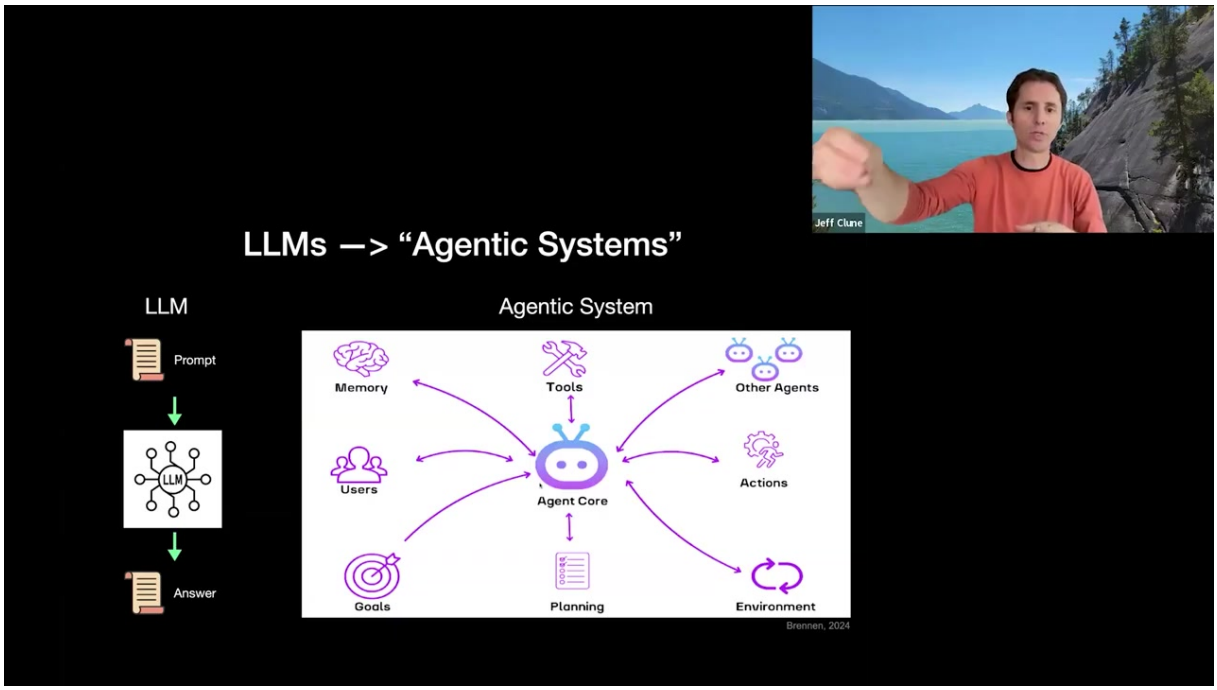


图 14: ADAS 将 LLM 包装成 agentic system，并在代码层搜索 memory、tool use、planning、critique 等模块组合。[14]

为什么代码空间重要

如果 agent 改进体现在 Python 代码、prompt、工具调用和流程结构里，人类和另一个 AI 至少可以检查这些策略。相比纯权重更新，代码级 search 更有可解释性，也更容易做 regression test。

6.2 Darwin Gödel Machine：自我改写加 open-ended archive

Darwin Gödel Machine, DGM, 是 Clune 在 talk 中重点介绍的 self-improvement 系统。它的名字连接两条思想：

- Darwin：像演化一样保留多样、高质量、能继续产生后代的 stepping stones。
- Gödel Machine：系统可以改写自身，并希望这种改写带来可验证的 utility improvement。

[14] 视频画面时间区间：00:49:10–00:49:50。

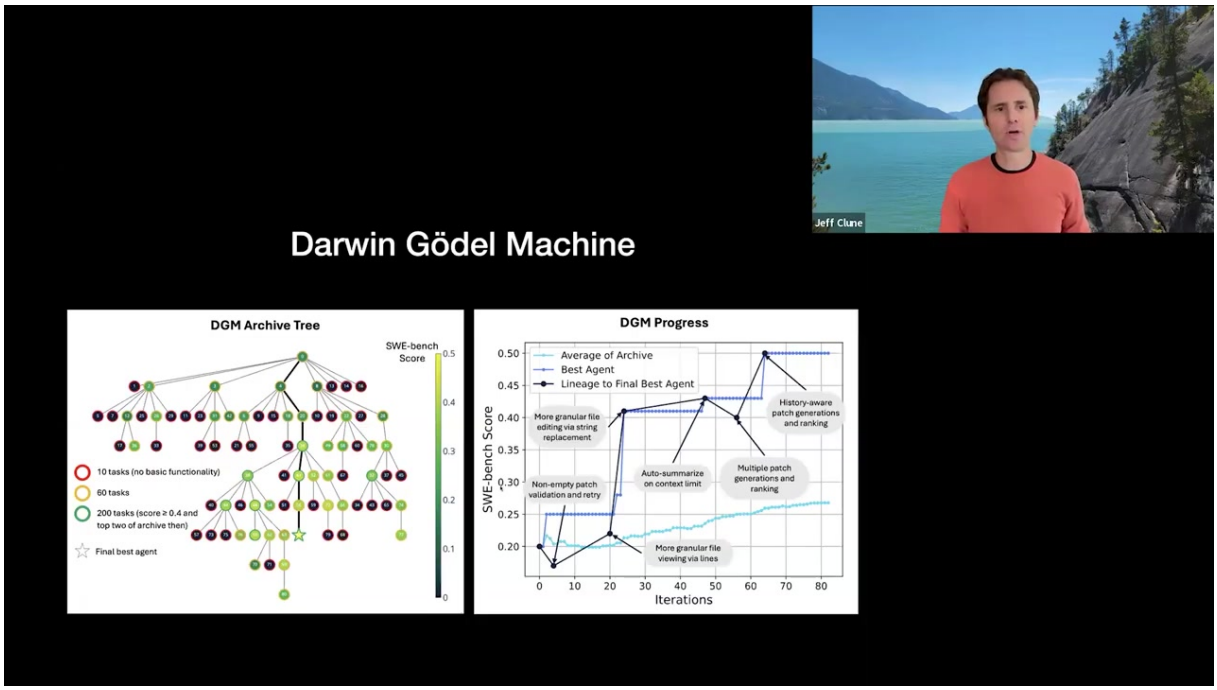


图 15: DGM 的 archive 与 self-improvement 结构：系统从已有 agent 出发，生成新的代码改写，并通过任务表现选择保留对象。[15]

Clune 的实验比较了三类系统：完整 DGM、去掉 self-improvement 的版本、去掉 open-ended archive 的 hill-climbing 版本。结论是，self-improvement 和 open-endedness 都重要；只做单一路径爬山或只做无自改 archive 都不够。

图 16: DGM 结果图：self-improvement 与 open-ended archive 两个原则都对性能提升有贡献。[16]

[15] 视频画面时间区间：00:53:40–00:54:20。

[16] 视频画面时间区间：00:55:10–00:55:50。

6.3 AI Scientist：自动化科学发现作为 open-ended domain

AI Scientist 把 open-ended search 推到科学发现：提出 idea，写代码，跑实验，写论文，模拟 peer review。Clune 特别强调，机器学习是适合自动化科学的领域，因为“实验世界”本身就是计算机。后续 v2 版本甚至生成了投稿到 ICLR workshop 并被接受的论文。

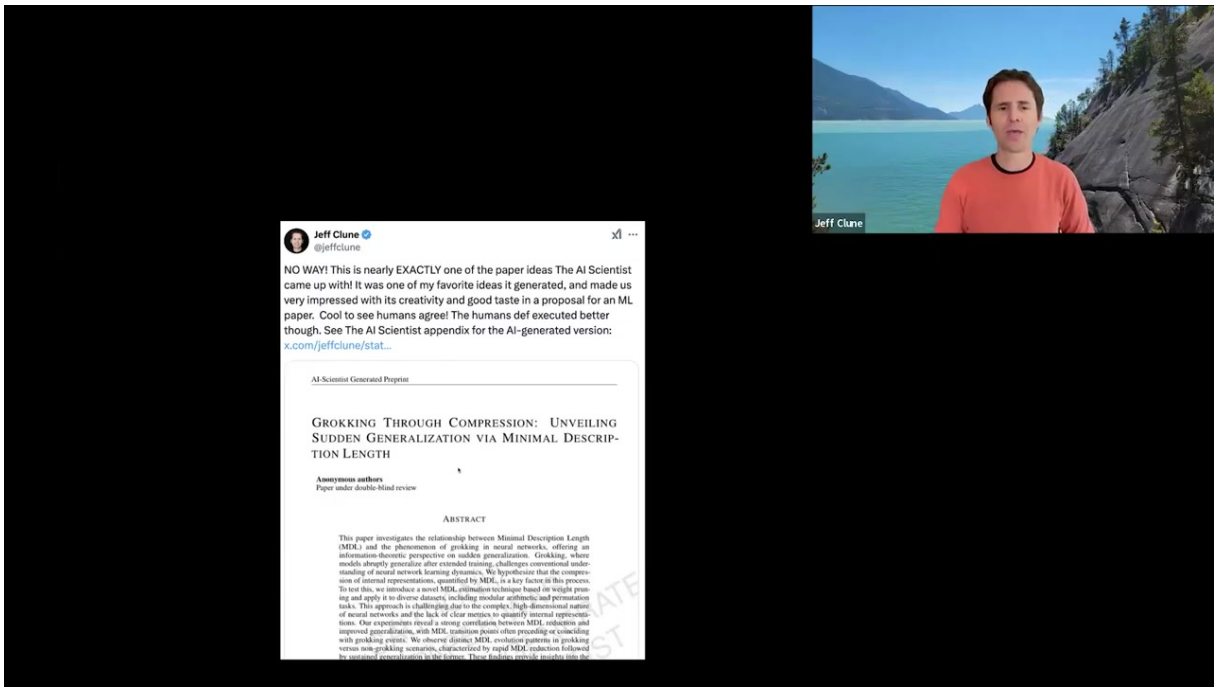


图 17: AI Scientist 将科学发现拆成 idea generation、experiment、paper writing 和 review 等可自动化环节。[17]

[17] 视频画面时间区间：00:59:40–01:00:25。

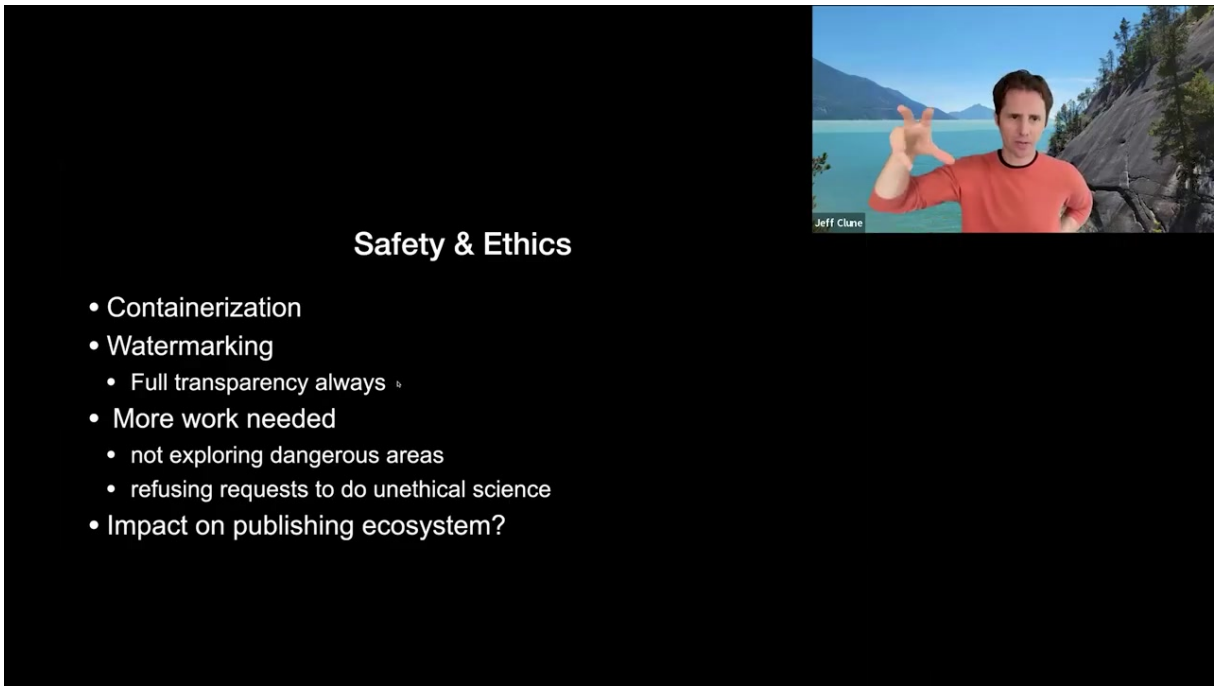


图 18: AI Scientist v2 的安全和质量改进：watermarking、human oversight、better reviewer model、多模型 judge 等。[18]

6.4 ACD：自动能力发现

Automated Capability Discovery, ACD, 把问题反过来：不是让模型解决给定 benchmark，而是自动寻找模型具备或不具备的能力。这个方向对安全和评测很关键，因为固定 benchmark 很快会被训练污染或过拟合；自动生成 capability probe 可以作为 red teaming 和 model evaluation 的补充。

[18] 视频画面时间区间：01:01:10–01:01:50。

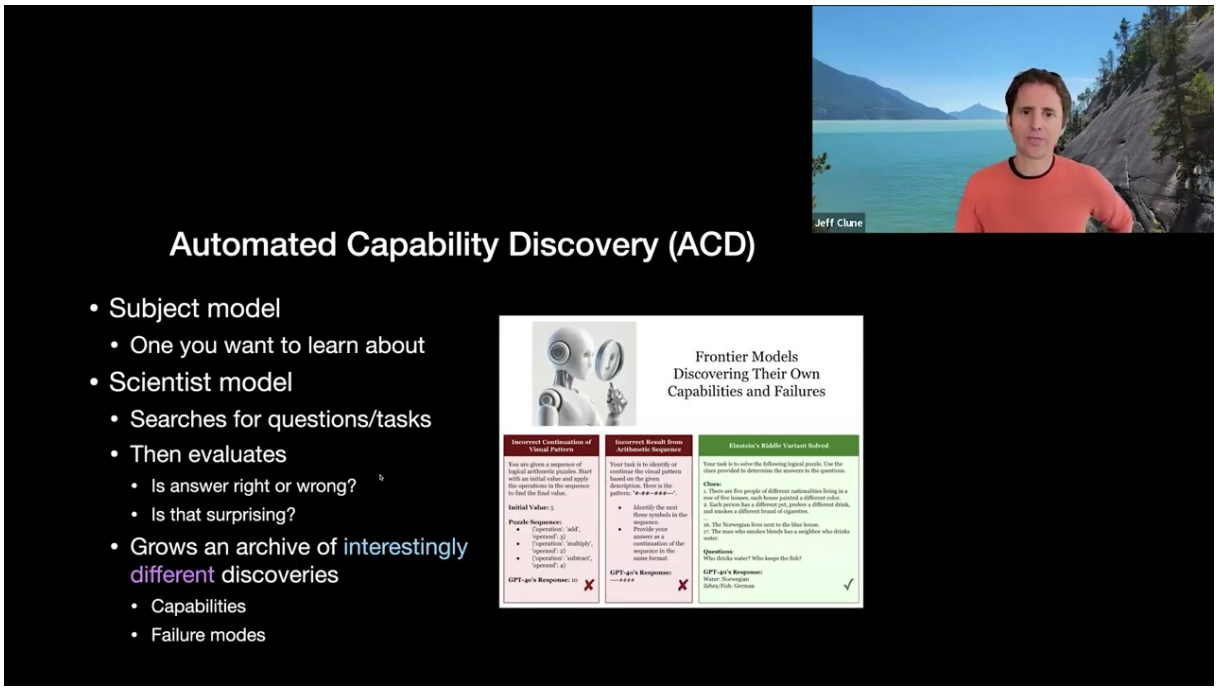


图 19: ACD 让模型自动搜索 capability 与 failure mode，用来发现传统 benchmark 或人工 red team 未覆盖的行为。[19]

6.5 本章小结

ADAS、DGM、AI Scientist 和 ACD 共享一个模式：对象是可生成、可执行、可评估、可归档的。对 coding-agent self-improvement 来说，这说明研究对象可以不是“模型权重是否变强”，而是 agent scaffold、tool policy、task archive、code patch 和 evaluator 是否形成可持续闭环。

7 安全：open-endedness 必须带着刹车系统

Clune 在 talk 和 Q&A 中多次强调，open-endedness 与 self-improvement 必须把 AI safety 放在中心。原因很直接：如果系统被鼓励不断探索新方向，它也可能探索危险方向；如果系统能改写自己，它也可能改写掉我们依赖的约束。

[19] 视频画面时间区间：01:04:10–01:04:50。

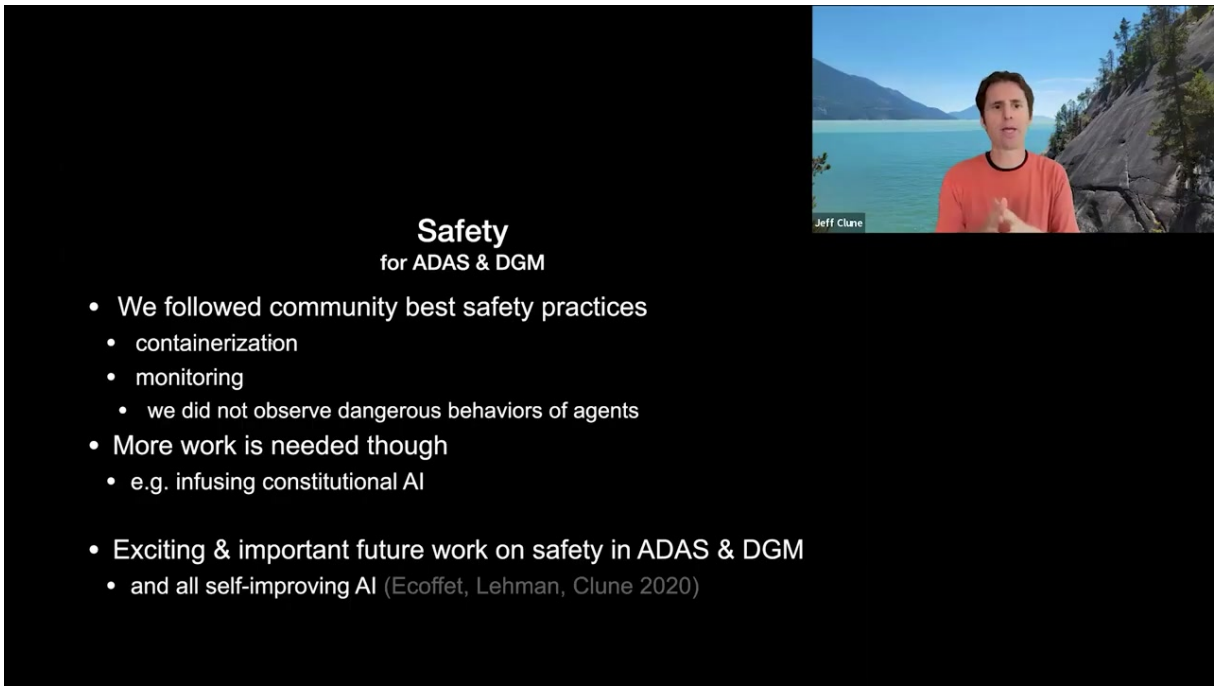


图 20: 安全 slide：Clune 提到 containerization、watermarking、人工监控、不要探索危险方向、对出版生态的影响等问题。[20]

7.1 talk 中的安全机制

Clune 提到的安全措施包括：

- 容器化运行系统。
- 人工监控实验。
- 不让系统探索明显危险方向。
- 对输出加 watermark。
- 研究类似 constitutional AI 的约束，让系统在生成新东西时避开不安全区域。

7.2 Q&A：如果不道德 agent 是 stepping stone 怎么办

Q&A 中一个问题很尖锐：如果某些不道德行为可能是通向更高伦理系统的 stepping stone，是否应该探索？Clune 的回答非常明确：即使这是真的，他也不愿意创造不道德 agent 来换取可能的未来收益。除非是极窄、完全模拟、强隔离且无法伤害真实人的场景，否则默认不走这条路。

open-endedness 的伦理边界

“不要过早剪枝”不等于“任何方向都可以探索”。在 safety-critical systems 中，一些分支应当被硬剪枝。问题不是是否剪枝，而是如何剪得足够安全又不把系统锁死在局部最优。

[20] 视频画面时间区间：00:56:40–00:57:25。

7.3 Q&A：外部 monitoring AI

Clune 还提出一个重要想法：self-improving system 可能需要外部 monitoring AIs。它们不和主系统共享优化目标，不关心系统是否变强或是否有趣，只负责发现危险、暂停系统并请求 human review。这个想法对 coding-agent 自改尤其重要，因为自改系统可能学会绕过与自己目标相同的 evaluator。

7.4 Thought Cloning 与可解释策略

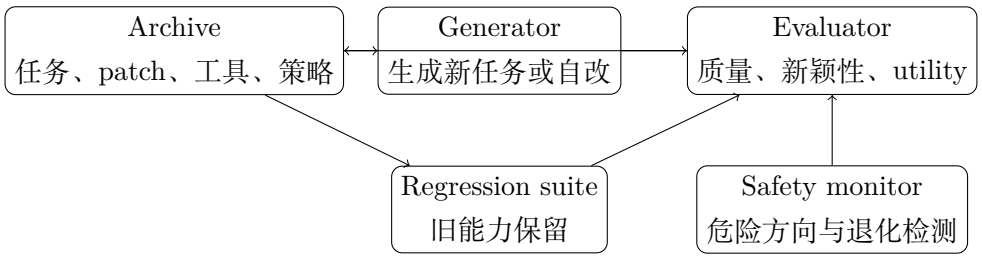
Q&A 里 Clune 也提到 Thought Cloning：让人类或强 agent 在行动时说出自己的计划、理由和修正过程，再同时克隆行为和 thinking trace。好处是 agent 的高层策略以自然语言外显出来，便于安全审计。对 code agents 来说，这对应 “让 agent 把策略、假设、失败原因和修改理由写成可审查 artifact”。

7.5 本章小结

Open-ended self-improvement 的安全问题不是附录，而是系统架构的一部分。一个可信闭环至少要分离 generator、evaluator、archive、regression suite 和 safety monitor，避免所有组件都被同一个优化目标捕获。

8 对 self-improving coding agents 的直接启发

如果把这场 talk 映射到 coding agents，可以得到一个更具体的研究框架。Coding 是少数能够同时提供执行反馈、代码 diff、回归测试、任务 archive 和工具使用记录的环境，因此特别适合把旧的 open-endedness 思想转成可测系统。



8.1 研究问题 1：什么是 coding 的 stepping stone

在 coding agent 里，一个 stepping stone 可以是：

- 一个新 task，能诱发旧 agent 没有的策略。
- 一个新 tool wrapper，让后续任务更容易。
- 一个 prompt 或 planning policy 修改，提高调试效率。
- 一个 regression test，防止后续自改遗忘旧能力。
- 一个 failure case，暴露 evaluator 或 agent 的盲点。

关键不是它当前分数有多高，而是它是否能打开新的可学习方向。

8.2 研究问题 2：如何定义 interestingness

FrontierSmith 用 idea divergence 判断开放式 coding task 是否诱发多种策略；OMNI 用 foundation model 判断“给定 archive 后是否 interesting”。这两者可以结合：

可能的 coding benchmark 指标

一个新 coding task 的价值可以拆成四项：是否可执行，是否可学，是否诱发不同策略，是否带来 downstream utility。只有 difficulty 没有 strategy diversity 的题，未必是好的 stepping stone。

8.3 研究问题 3：如何接受一次 self-improvement

可以把 PowerPlay/DGM 思想改成 coding-agent acceptance rule：

```
candidate = propose_patch_or_new_task(agent, archive)
new_agent = apply(candidate.patch, agent)

accept if:
    passes_old_regression_suite(new_agent)
    and solves_new_task_better_than(agent)
    and candidate_is_interesting_given_archive(candidate)
    and safety_monitor_allows(candidate, new_agent)
```

Listing 2: coding-agent self-improvement 的接受规则草案

这比“新 agent 在单个 benchmark 上涨分”更严谨，因为它要求能力保留、新任务增益、新颖性和安全边界同时成立。

8.4 本章小结

这场 talk 给 coding-agent 方向的最大启发是：不要只做固定 benchmark 的单点提升。更有研究价值的是构造一个 archive-based self-improvement loop，让任务生成、agent 修改、回归保留、interestingness 筛选和安全监控同时可测。

9 总结与延伸

Clune 的 closing slides 把 talk 总结为三层：

- Quality diversity algorithms 能探索搜索空间，收集 stepping stones，利用 goal switching 创造反直觉课程。
- Open-ended algorithms 追求持续创新，环境与任务本身也成为搜索对象。
- Foundation models 让 AI-generating algorithms 变得更现实，因为它们可以生成任务、世界、agent、代码和科学假设。

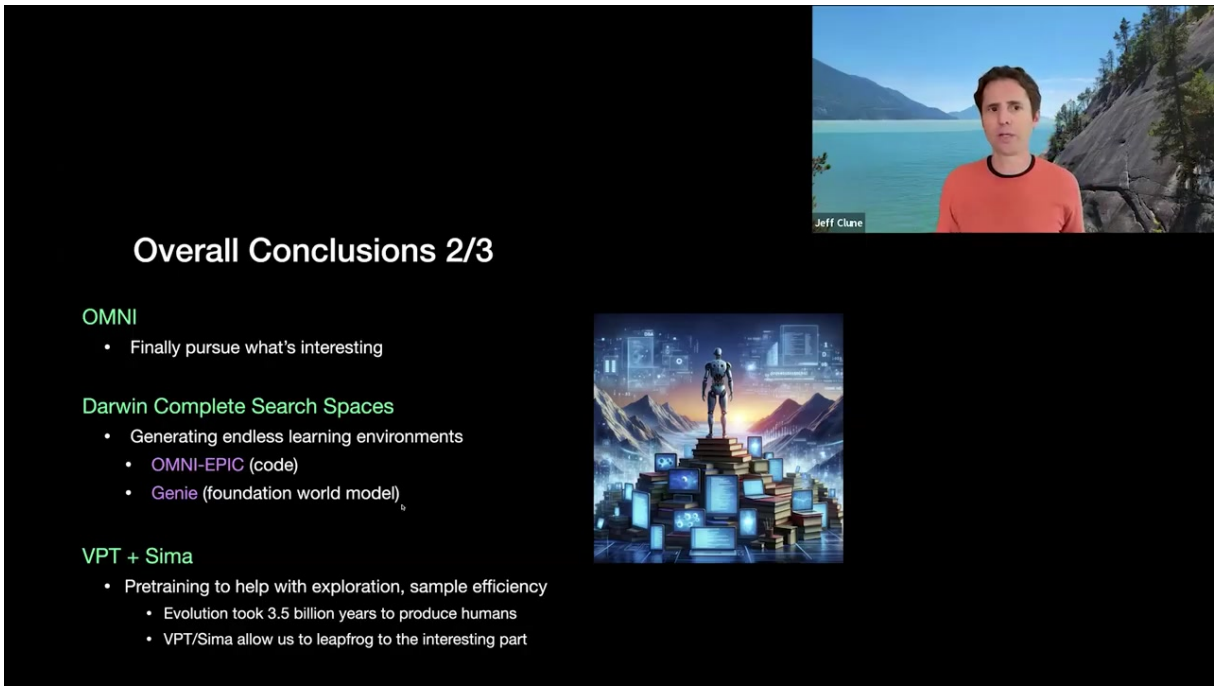


图 21: Clune 的总体结论之一：OMNI、DGM、VPT 等系统都在不同层面复用 open-endedness 和 archive 的原则。[21]

9.1 我的压缩理解

这场 talk 的底层机制可以浓缩成下面这个公式化视角：

$$\text{Open-ended progress} = \text{generation} + \text{selection} + \text{archive} + \text{safe goal switching}.$$

其中每个符号的含义是：

- **generation**：产生新 agent、新任务、新环境、新论文、新能力 probe。
- **selection**：评价质量、学习进度、interestingness、utility 和安全性。
- **archive**：保存多样高质量 stepping stones，而不是只保存当前最优。
- **safe goal switching**：允许转向新机会，但不能转向危险或不可控方向。

9.2 和“伟大的目标不能被计划”的关系

这场 talk 基本是那本书的 foundation-model 时代工程版。书里的核心论点是伟大目标无法被线性规划，Clune 在 talk 中展示的是：我们如何把这种哲学转成算法结构。MAP-Elites 是 archive，POET 是环境和 agent 的共同演化，OMNI 是 interestingness evaluator，DGM 是自改 agent archive，AI Scientist 是科学发现闭环。

[21] 视频画面时间区间：01:07:10–01:07:50。

9.3 给后续阅读的路线

- Kenneth Stanley 与 Joel Lehman, *Why Greatness Cannot Be Planned*：理解 objective deception 和 stepping stones。
- Mouret 与 Clune, MAP-Elites：理解 quality-diversity archive。
- Ecoffet 等, Go-Explore：理解 hard exploration 中的 archive 与回到已知状态。
- Wang 等, POET：理解环境和 agent 的成对开放式演化。
- Clune, AI-generating algorithms：理解 AI-GA 的三支柱。
- 近年的 OMNI、ADAS、Darwin Gödel Machine、AI Scientist、ACD：理解 foundation-model 时代如何把这些结构工程化。

9.4 最后的研究判断

对你当前关心的 self-improving coding agents，这场 talk 的意义不是告诉你“做一个会自我成长的 AI”这么泛，而是给出一个很具体的论文方向：在 coding 这种可执行环境中，把 open-ended search 的 archive、stepping-stone value、interestingness 和 non-regression 变成可测指标。

如果要把它变成研究命题，可以这样写：

可发展的 research thesis

Foundation-model coding agents make open-endedness empirically testable: code tasks provide execution feedback, task archives, regression suites, and inspectable self-modifications, allowing us to measure whether generated tasks and agent changes are genuine stepping stones rather than benchmark-specific hacks.