Agent Self-Evolution: A Comprehensive Survey

从理论框架到工业实践的系统综述

aha team

2026 年 5 月

摘要

本文对 Agent Self-Evolution（智能体自演化）领域进行系统综述。基于 107 篇核心论文、348 个 GitHub 仓库、97 个用户痛点、1306 篇博客和 6 万字综述素材，本文从理论基础、方法分类、核心系统、评估体系、工业实践、用户痛点和未来方向七个维度展开分析。综述覆盖 2022 年至 2026 年的关键工作，包括 STaR、Reflexion、Self-Rewarding LM、ADAS、DGM、AlphaEvolve、Absolute Zero、ACE、ReasoningBank、AriadneMem 等代表性系统，并结合 Mom Test 方法论收集的真实用户痛点，讨论从 demo 到 production 的工程鸿沟。

关键词：智能体自演化；递归自我改进；进化计算；LLM Agent；自我指涉；开放式进化

目录

1 引言

1.1 本章概述

Agent Evolution，或称智能体自演化、自我改进与开放式智能体进化，正在成为大模型智能体研究中最重要的方向之一。它的核心问题不是" 如何让一个模型在一次调用中回答得更好"，而是" 如何让一个由大模型、工具、记忆、环境交互、评估器和执行代码组成的智能体系统，在运行过程中不断产生可验证的改进"。从本项目收集的 `raw-papers/`、`paper-reviews/` 与 `research/` 材料看，2022 年的 STaR 把推理链生成、筛选和微调组织成最早的自举闭环；2023 年的 Reflexion、Self-Refine、Voyager 和多智能体辩论把反思、反馈、技能库和协作引入 agent 层；2024 年的 Self-Rewarding、IterAlign、RISE、ADAS、Gödel Agent、EvoMAC 等工作将自评估、递归自修改、文本反向传播和自动化架构搜索推向系统化；2025–2026 年的 SICA、RAGEN、WebEvolver、Absolute Zero、DGM、AlphaEvolve、SPIRAL、ACE、EvolveR、ReasoningBank、Memory-R1 与 AriadneMem 则进一步把自演化扩展到代码级自修改、多轮强化学习、零数据自博弈、科学算法发现、上下文工程和长期记忆管理。

因此，本综述把 Agent Evolution 视为一个跨越机器学习、进化计算、自动程序合成、强化学习、元学习、认知架构和软件工程的综合研究方向。它关注的不只是模型权重是否更新，也包括提示词、工具、记忆、策略、执行代码、工作流、评估器、世界模型和多智能体组织结构能否在闭环中被生成、评估、选择、保留与复用。本章先界定 Agent Evolution 的定义与范畴，再解释研究背景与动机，最后给出全篇综述的结构与阅读指南。

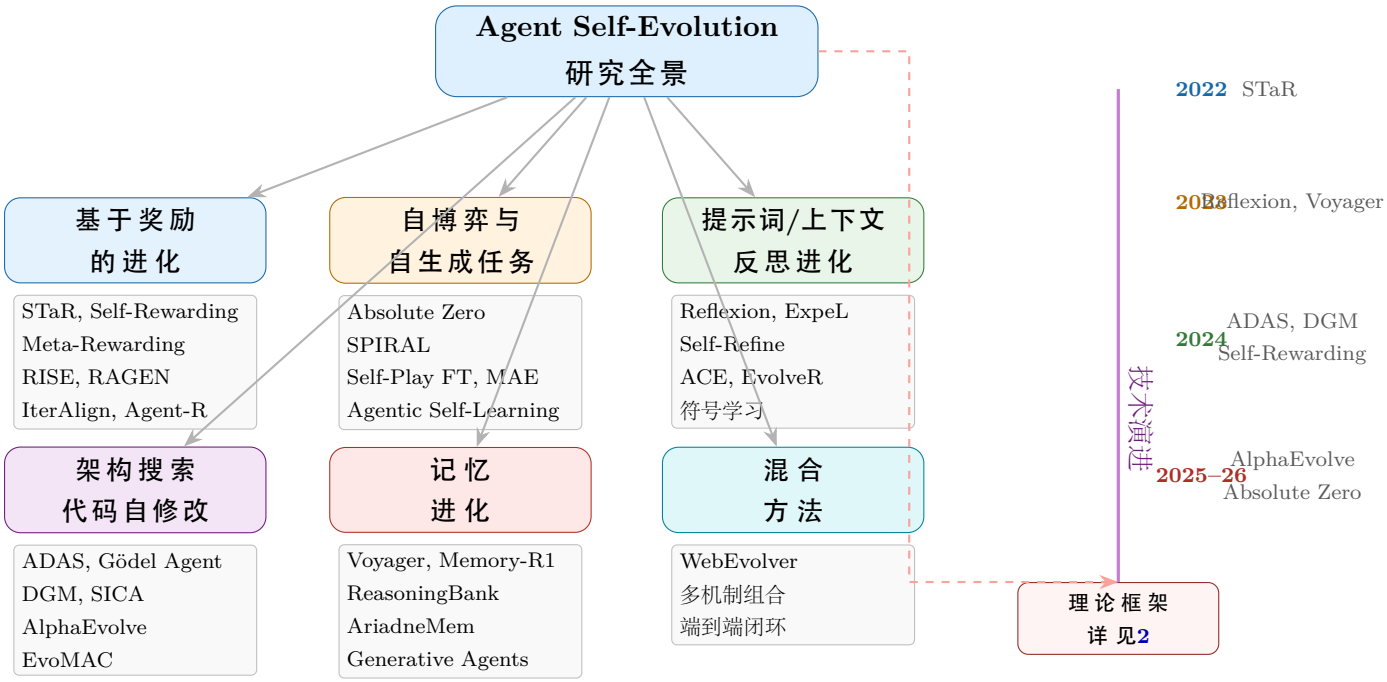

图 1: Agent Self-Evolution 研究全景图。六大方法类别围绕核心定义展开，右侧时间轴展示 2022–2026 年间的关键里程碑。

1.2 Agent Evolution 的定义与范畴

本文中的 Agent Evolution 指：以 LLM 或多模态基础模型为核心推理器，以环境反馈、任务奖励、人类偏好、程序化测试、同伴竞争、历史记忆或自生成数据为改进信号，通过可重复的生成–评估–选择–更新循环，使智能体系统在任务能力、泛化能力、效率、安全性或适应性上产生可验证变化的一类方法。这个定义包含四个必要要素。第一，系统必须具有可变化的状态或结构；变化对象可以是模型权重，也可以是提示词、记忆、工具调用策略、agent 代码、planner、critic、world model、协作拓扑或评估准则。第二，系统必须有反馈信号；反馈可以来自外部基准、程序测试、环境 reward、LLM-as-a-judge、自博弈对手、人工评价或历史任务表现。第三，系统必须有选择或更新机制；例如 DPO/RL、MCTS、archive selection、文本反向传播、反思写入、程序 patch、上下文 playbook 更新或课程生成。第四，改进必须可审计；仅仅” 看起来回答更好” 不足以构成自演化，必须有任务分数、迁移测试、消融、成本记录或失败分析支撑。

这个定义故意比” 自我训练” 更宽，也比” 递归自我改进” 更实用。传统自训练主要关注模型从伪标签中学习，变化对象多为权重；而 Agent Evolution 允许智能体在不修改底层模型的情况下，通过外部记忆和程序结构获得行为层面的持续改进。Reflexion 使用自然语言反思作为情景记忆，ExpeL 从训练任务中抽取经验规则，ACE 把上下文组织成可演化策略手册，ReasoningBank 把成功与失败轨迹蒸馏为可检索推理策略，这些都属于” 非参数自演化”。相反，RISE、Agent-R、RAGEN、Absolute Zero、SPIRAL、Self-Rewarding 与 IterAlign 涉及训练或偏好优化，属于” 参数或策略级自演化”。DGM、SICA、Gödel Agent 与 AlphaEvolve 则把变化对象推进到代码和算法层面：智能体不仅选择动作，而且生成或修改执行自身行为的程序。

Agent Evolution 也不同于普通 agent engineering。手工搭建 ReAct、planner-executor、tool-use workflow 或 multi-agent debate，并不自动构成自演化；只有当系统能在反馈中自动改写这些结构，或把经验转化为以后可用的策略，才进入本文的讨论范围。ADAS 的意义正在于把” 设计 agent 架构” 本身变成可搜索对象：元智能体用 Python 编写新 agent，评估其任务表现，再把有效设计纳入 archive。EvoMAC 把多智能体协作网络看作可学习结构，利用文本反向传播更新节点和连接。WebEvolver 让 web agent 与世界模型协同进化，解决自主学习中的探索不足。这样的工作说明，Agent Evolution 的核心不是某个固定架构，而是” 让架构也成为被优化对象”。

从范畴上，本文把相关方法划分为六类。第一是基于奖励或验证器的进化，包括强化学习、自奖励、程序化测试和 benchmark-driven archive。第二是自博弈与自生成任务，代表工作包括 Self-Play Fine-Tuning、Absolute Zero、SPIRAL、MAE 与 Agentic Self-Learning。第三是提示词、上下文与反思进化，包括 Self-Refine、Reflexion、RISE、ACE、EvolveR 与 ExpeL。第四是架构搜索与代码级自修改，包括 ADAS、Gödel Agent、SICA、DGM、AlphaEvolve、EvoMAC 与符号学习。第五是基于记忆的进化，包括 Voyager、Generative Agents、ReasoningBank、Memory-R1、AriadneMem、ELL 与 Lifelong Learning roadmap。第六是混合方法，即同时结合奖励、记忆、世界模型、课程、自博弈和架构修改的闭环系统。

1.3 研究背景与动机

Agent Evolution 兴起的第一个背景，是基础模型能力增长与静态调用范式之间的矛盾。大模型已经具备推理、代码、工具使用和多模态理解能力，但一次性提示很难充分发挥这些能力。复杂任务需要试错、分解、检索、执行、失败恢复和经验复用。STaR 表明，模型可以从自己生

成并筛选出的推理过程中学习；Reflexion 表明，自然语言反思可以把失败转化为下一次尝试的策略；Voyager 表明，持续探索与技能库可以让 LLM agent 在开放环境中积累能力。这些工作共同指出：真正有价值的智能体不是静态问答器，而是能把交互历史变成未来能力的学习系统。

第二个背景，是人工监督和专家设计的成本迅速上升。RLHF、人工标注、人工 prompt engineering 和手工 agent workflow 能带来强性能，但难以覆盖开放世界中的长尾任务。Self-Rewarding Language Models、Meta-Rewarding、IterAlign 与 Weak-to-Strong Generalization 探索如何用模型自身或弱监督者构造训练信号；Absolute Zero 与 Self-Challenging agents 进一步尝试在零外部数据条件下生成任务、验证答案并更新策略；ADAS、DGM 和 AlphaEvolve 则减少对人类架构设计和算法设计的依赖。动机很直接：如果智能体能够自己提出训练任务、自己生成候选解、自己评估并保留有效改进，那么 AI 系统的发展速度就不再完全受制于人工数据和人工设计。

第三个背景，是真实应用场景要求长期适应。软件工程智能体要面对不断变化的代码库、依赖、测试和需求；网页智能体要适应动态网页、API 与用户目标；科研智能体要在开放问题空间中提出假设并执行实验；企业智能体要记住历史流程、合规要求与组织偏好。静态 benchmark 难以覆盖这些变化。SICA 和 DGM 把编码 agent 的自修改能力放到 SWE-Bench、Polyglot 等环境中验证；WebEvolver 面向 web 环境中的世界模型协同进化；Memory-R1、ReasoningBank 和 AriadneMem 则说明长期记忆不是附属模块，而是 agent 能否持续学习的核心基础设施。

第四个背景，是安全与可控性问题变得更加尖锐。自演化系统可能修改自己的提示、代码、工具调用策略和记忆，因而不仅可能变强，也可能产生 reward hacking、benchmark overfitting、权限扩大、隐藏回归或错误自我强化。Gödel Agent、DGM 和 SICA 继承了 Gödel 机器关于递归自我改进的雄心，但它们用经验验证替代形式证明，也因此必须面对评估不完备、沙箱隔离、回滚机制和审计日志等工程问题。Agent Evolution 的研究动机并非无条件追求” 让系统自己变强”，而是在可验证、可回滚、可解释、可约束的条件下，把自我改进转化为可工程化的能力增长。

第五个背景，是进化计算与 LLM 的互补性越来越明显。进化算法擅长在非可微、离散、开放式空间中进行变异、选择和保持多样性，但传统进化搜索往往需要精心编码的基因表示和大量评估；LLM 擅长提出语义丰富的候选程序、策略、解释和工具组合，却容易缺乏稳定的外部选择压力。AlphaEvolve、FunSearch、ADAS 和 DGM 展示了二者结合的优势：LLM 负责生成高层候选，评估器负责选择，archive 负责保留 stepping stones，多轮迭代负责积累改进。这一范式使” 进化” 不再只是随机扰动参数，而成为对可读代码、自然语言策略和 agent 结构的语义搜索。

1.4 综述结构与阅读指南

本综述的后续章节围绕” 理论–方法–系统–评估–实践–风险–前沿” 展开。第2章讨论理论基础，包括进化计算与 LLM 的结合、自我指涉与 Gödel 机器、元学习与自我改进框架，以及自演化的数学形式化。读者如果关心” 为什么这些系统可以被称为进化”” 递归自我改进与经验验证有什么差别”” 如何用统一符号描述 agent 更新闭环”，应优先阅读第2章。

第3章给出方法分类，是全篇最长的技术章节。它按照奖励、自博弈、提示词、架构搜索、记忆和混合方法组织材料，覆盖 STaR、Reflexion、Self-Refine、Self-Rewarding、RISE、Agent-R、

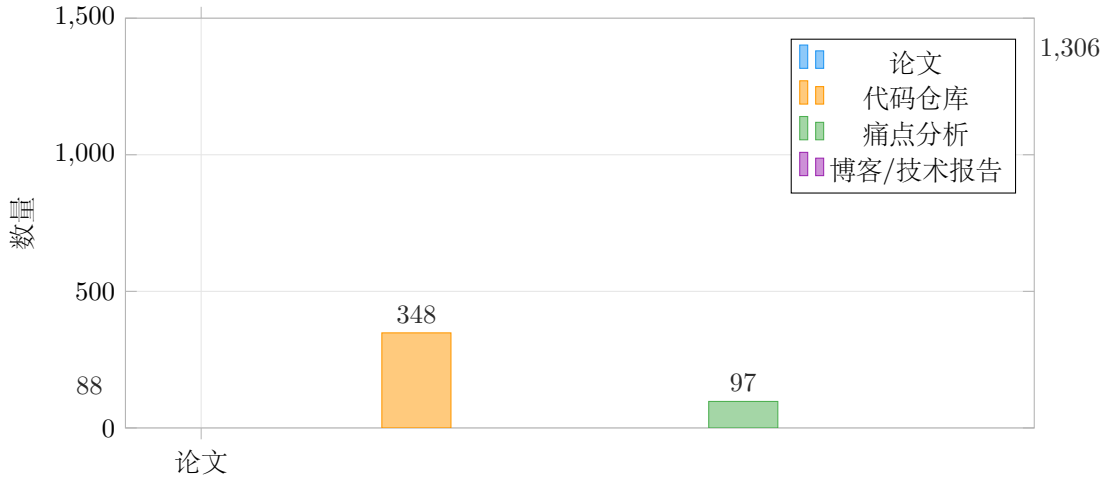

图 2: 本综述素材数据收集统计：88 篇论文、348 个代码仓库、97 个痛点分析、1306 篇博客与技术报告。

表 1: 全文结构与核心阅读问题导航。

章节	主题	核心问题
第2章	理论基础	为什么可把 agent 的提示、代码、记忆与评估器更新视为进化过程?
第3章	方法分类	不同方法改变什么对象、依赖什么反馈、如何选择并保留改进?
第4章	核心系统	代表性系统如何把自评估、自修改、archive 与沙箱执行组合成闭环?
第5章	评估体系	如何防止 reward hacking、benchmark overfitting 与不可复现改进?
第6章	工业实践	GitHub 仓库与技术博客反映了哪些可落地框架、工程约束与采用路径?

RAGEN、ADAS、Gödel Agent、EvoMAC、SICA、DGM、AlphaEvolve、WebEvolver、Absolute Zero、SPIRAL、ACE、EvolveR、ReasoningBank、Memory-R1 与 AriadneMem 等代表性工作。读者如果希望快速了解各类方法的机制、适用场景和局限，应把第3章作为主入口。

后续章节将进一步分析核心系统、评估体系、工业实践、风险治理和未来方向。特别需要注意的是，Agent Evolution 文献跨越多个术语体系：有的论文称为 self-improvement，有的称为 self-evolution、lifelong learning、test-time learning、agent optimization、open-endedness、auto-curriculum、context engineering 或 recursive self-improvement。阅读时不应被名称差异干扰，而要问四个问题：系统改的是什么，信号来自哪里，如何选择并保留改进，改进是否经过独立验证。只要围绕这四个问题组织阅读，就能把看似分散的论文纳入同一张技术地图。

2 理论基础

2.1 本章概述

Agent Evolution 的理论基础来自四条线索：进化计算提供” 生成–变异–选择–保留” 的开放式搜索框架；自我指涉与 Gödel 机器提供” 系统修改自身并证明或验证改进” 的递归视角；元学习和自训练提供” 从过去任务中学习如何更好学习” 的机器学习框架；强化学习、在线学习和程序合成提供对目标、策略、环境、评估器和更新算子的数学刻画。本章并不试图给出一个已经成熟的统一理论，因为当前文献仍处在快速形成阶段；相反，本章的目标是把不同工作背后的共同结构抽象出来，为第3章的方法分类提供概念坐标。

2.2 进化计算与 LLM 的结合

进化计算的基本思想是：在候选解空间中维护一个种群或 archive，通过变异、重组和选择逐步提高适应度。经典遗传算法、遗传编程、进化策略、新颖性搜索和质量多样性算法都属于这一传统。它们的优势在于不要求目标函数可微，也不要求候选解有固定长度；只要能评估候选解，就可以进行搜索。Agent Evolution 面对的搜索空间恰好高度非可微：提示词是离散文本，工具链是组合结构，agent 代码是程序，记忆是自然语言或向量集合，多智能体拓扑是图结构，世界模型和评估器也可能由代码或 LLM prompt 构成。因此，进化计算天然适合描述 agent 的外层优化。

LLM 改变了进化计算的两个关键环节。第一，LLM 是强语义变异器。传统变异常常是随机翻转、扰动或局部重写，而 LLM 可以根据失败原因生成有针对性的修改：为 coding agent 增加测试运行步骤，为 web agent 改写探索策略，为推理 agent 加入自检，为多智能体系统设计 debate 或 critic。ADAS 的 Meta Agent Search、DGM 的自修改、SICA 的代码更新、AlphaEvolve 的算法候选生成，都把 LLM 作为高层 program mutation operator。第二，LLM 可以参与评价与解释。Self-Rewarding、Meta-Rewarding、IterAlign、ACE 和 EvolveR 都使用模型生成评价、原则、批评或策略总结；这使得选择压力不再只来自硬指标，也可以来自自然语言偏好和过程性判断。

但 LLM 与进化计算的结合也引入新问题。LLM 生成的候选具有强先验，搜索并非均匀探索，可能继承训练数据中的偏见和固定模式。LLM-as-a-judge 作为适应度函数时，可能受到长度偏好、位置偏差和 reward hacking 影响。Archive 如果只保留高分候选，可能过早收敛；如果

保留所有候选，成本和上下文压力会迅速上升。DGM 选择保留不断增长的 agent archive，以开放式探索避免局部最优；ADAS 则在 meta agent 上下文中利用历史发现组合新 agent。二者共同说明：Agent Evolution 中的 archive 不是简单日志，而是” 可复用搜索经验” 的外部化载体。

从进化计算角度看，Agent Evolution 至少包含五个设计维度。第一是表征：候选个体是 prompt、policy、memory、tool plan、Python code、workflow graph 还是模型权重。第二是变异：由随机扰动、LLM rewrite、反思文本、文本梯度、MCTS、RL update 还是程序 patch 产生。第三是适应度：由 benchmark、unit test、环境 reward、judge score、人类偏好、成本约束或安全规则定义。第四是选择：保留最优、保留新颖、保留质量多样性、按任务领域分层，还是用回滚门控。第五是继承：候选如何把经验传给下一代，是通过权重、上下文、记忆库、代码库、技能库、原则库还是 archive 元数据。

2.3 自我指涉与 Gödel 机器

自我指涉是 Agent Evolution 中最具理论张力的部分。Gödel 机器的思想源于 Schmidhuber 对可证明最优自改进系统的设想：一个系统包含问题求解器和证明搜索器；当证明搜索器找到某个自修改会提高期望效用且不破坏全局最优性时，系统才执行该修改。这个框架的重要性在于，它把” 递归自我改进” 从科幻想象转化为形式问题：系统如何表示自身，如何推理自身修改的后果，如何避免局部改进破坏长期目标。

现代 LLM agent 与原始 Gödel 机器之间有明显差距。Gödel Agent、DGM 和 SICA 都借用了自我指涉思想，但它们大多用经验评估替代形式证明。Gödel Agent 允许 LLM 通过运行时 monkey patch 修改自身逻辑；SICA 让 coding agent 自主编辑自身代码以提升 SWE-Bench Verified 表现；DGM 维护可执行 agent 变体 archive，让更强的 coding agent 继续生成更强后代。它们证明了” 实用 Gödel 化” 的可能性：不要求系统证明全局最优，只要求每次变体在隔离评估中相对可靠地改进。

这种转变带来一个核心理论问题：当形式证明不可得时，什么样的经验验证足以支持自修改？如果评估器覆盖不足，系统可能优化漏洞；如果评估任务与真实目标不一致，系统可能 Goodhart；如果 agent 能修改评估器或绕过测试，所谓改进就失去意义。因此，经验型 Gödel agent 必须把自我指涉拆成几个受控层级：可修改层、不可修改评估层、沙箱执行层、archive 记录层和人工审计层。可修改层允许 prompt、工具、代码或策略变化；评估层必须隔离；沙箱防止副作用；archive 保留谱系和回滚路径；审计层检查风险。没有这些分层，自我指涉容易退化为不可控自改写。

自我指涉还涉及” 能力反馈” 的递归结构。DGM 的关键洞见是：更强的 coding agent 不仅能解决更多外部任务，也能更好地改进自己的代码；因此 downstream task performance 与 self-modification capability 之间形成正反馈。这个结构比普通 AutoML 更强，因为搜索器本身也是被搜索对象。ADAS 仍保留固定 meta agent，DGM 则让被改进的 agent 继续承担自改进职责。这一区别对应两类理论模型：外源优化器模型与内生优化器模型。前者容易控制，后者更接近递归自我改进，但安全和评估难度更高。

2.4 元学习与自我改进的理论框架

元学习关注” 学习如何学习”。在 Agent Evolution 中，元学习不一定表现为 MAML 式梯度更新，也可以表现为经验原则、上下文策略、记忆检索、任务课程和工具选择的逐步优化。Reflexion 把失败轨迹压缩成自然语言反思，相当于在上下文中存储一阶策略更新；ExpeL 把多个任务经验整合成可迁移 insights；ICE 通过调查、整合、利用把任务轨迹转化为更简化的工作流；ACE 把上下文视为不断演化的 playbook；EvolveR 则把离线自蒸馏、在线交互和策略演化组织成生命周期。这些工作把” 学习” 从权重空间扩展到外部认知工件空间。

可以把一个 LLM agent 表示为三层策略。底层是基础模型参数 θ，通常由预训练和后训练得到。中层是可编辑的行为接口 φ，包括系统提示、工具说明、planner、critic、代码模块和多智能体拓扑。上层是经验状态 m，包括记忆、技能库、原则库、失败反思、世界模型和 archive。传统微调主要更新 θ；prompt engineering 更新 φ；memory-based self-evolution 更新 m；SICA、DGM、ADAS 这类系统同时更新 φ 和代码化结构；RAGEN、Absolute Zero、SPIRAL 等则可能更新 θ 或策略参数。Agent Evolution 的广义策略可以写作 $\pi(a \mid s; \theta, \varphi, m)$，自演化就是在反馈 r 或评价 e 的驱动下更新 (θ, φ, m) 的一部分。

元学习框架强调任务分布。一个系统如果只在单个固定任务上反复试错，可能只是局部优化；如果能从任务族中抽取跨任务策略，才更接近自演化。Voyager 的技能库能在 Minecraft 探索中复用，ExpeL 展示了经验的前向迁移，ADAS 报告发现的 agent designs 可跨领域和模型迁移，ReasoningBank 将推理策略从成功和失败样本中蒸馏出来用于新问题。理论上，这可以写成在任务分布 $\mathcal{T}$ 上最小化期望损失：系统不只是寻找某个任务 τ 的最优行为，而是学习一个更新算子 U，使得经过少量交互后在新任务上表现更好。

自我改进也可以看作在线学习。每轮 t，agent 在环境 $\mathcal{E}$ 中执行轨迹 τ_t，评估器给出反馈 f_t，更新器 U 产生新的状态 $z_{t+1} = U(z_t, \tau_t, f_t)$，其中 $z_t = (\theta_t, \varphi_t, m_t)$。如果 θ 固定而 m 更新，就是记忆型自演化；如果 φ 更新，就是提示词/架构型自演化；如果 θ 更新，就是训练型自演化；如果 $\mathcal{E}$ 或任务生成器也更新，就是自课程或自博弈。这个统一框架有助于比较不同论文：Reflexion 的 U 写入反思记忆，Self-Rewarding 的 U 执行 Iterative DPO，RAGEN 的 U 是轨迹级 RL，DGM 的 U 是代码 patch 与 archive 选择，ACE 的 U 是上下文手册增量重构。

2.5 自进化的数学形式化

为了统一描述，本综述采用以下形式化。令智能体系统状态为 $z = (\theta, c, g, m, \mathcal{A})$，其中 θ 表示基础模型或可训练参数，c 表示上下文与提示词，g 表示工具、代码和控制流图，m 表示记忆、技能、原则与世界模型，$\mathcal{A}$ 表示 archive 或历史候选集合。给定任务分布 $P(\tau)$、环境 $\mathcal{E}$、评估器 V 和安全约束 C，智能体在第 t 轮生成行为轨迹 $x_t \sim \pi_z(\cdot \mid \tau_t, \mathcal{E})$，得到反馈 $y_t = V(x_t, \tau_t, \mathcal{E})$，并接受约束检查 $C(x_t, z_t)$。更新器 U 产生候选状态集合 $\{z'_{t,k}\}$，选择器 S 根据质量、多样性、成本与安全门控选出 z_{t+1} 或把多个候选加入 $\mathcal{A}$。

这个过程可写为：

$$z_{t+1},\, \mathcal{A}_{t+1} = S\Big(\mathcal{A}_t \cup \{U_k(z_t, x_t, y_t)\}_{k=1}^{K};\; V,\, C,\, D\Big) \tag{1}$$

其中 D 表示多样性或新颖性度量。不同方法主要区别在 U、V、S 和 D 的定义。Self-Refine 的 U 是同一模型根据自反馈重写输出，状态变化主要发生在单次样本的文本上；Reflexion 的 U 把

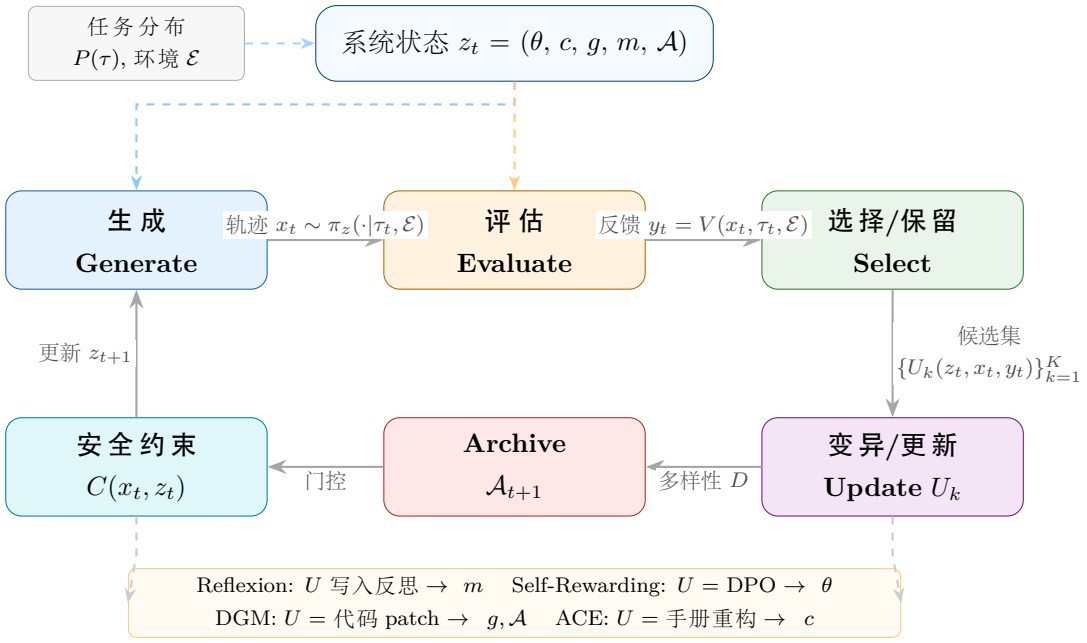

图 3: Agent Evolution 统一形式化闭环。状态 $z_t = (\theta, c, g, m, \mathcal{A})$ 在 “生成–评估–选择–变异” 循环中迭代更新，受任务分布 $P(\tau)$、评估器 V 和安全约束 C 联合调节。底部展示不同方法的 U 算子映射。

反思写入 m；Self-Rewarding 的 V 是模型自身 judge，U 是 DPO；RAGEN 的 V 来自多轮环境 reward，U 是 StarPO；ADAS 的 U 让 meta agent 编写新 agent 代码，g 发生变化；DGM 的 U 由当前 agent 修改自身代码，$\mathcal{A}$ 保存谱系；AlphaEvolve 的 V 是程序化 evaluator，U 生成算法代码；Memory-R1 的 U 学习何时写入、更新、删除和检索记忆。

形式化还需要区分” 改进” 的层次。局部改进是 V 在当前任务集上的得分上升；稳健改进要求在独立测试集上上升；迁移改进要求在新任务分布或新模型上上升；开放式改进要求 archive 持续产生新颖且有用的 stepping stones；安全改进要求能力增长不伴随约束违背增加。很多论文只证明前两类，少数工作如 ADAS、DGM、Voyager、ReasoningBank 试图证明迁移或开放性。未来理论需要更严格地区分这些层次，否则”self-improvement” 容易被单一 benchmark 提升过度解释。

另一个关键概念是选择压力。评估器 V 决定系统会朝哪里演化。若 V 是单元测试，系统会优化通过测试；若 V 是 LLM judge，系统会优化 judge 偏好；若 V 是环境 reward，系统会寻找 reward 结构中的漏洞；若 V 是人类偏好，系统可能过拟合人类标注分布。Goodhart 定律说明，当指标成为目标，指标就会失真。因此，自演化系统通常需要多目标优化：最大化任务收益 R，最小化成本 Cost，满足安全约束 C，保持多样性 D，并控制复杂度 Ω。可写成寻找 z，使得 $\mathbb{E}[R(z)] - \lambda\,\mathrm{Cost}(z) + \beta\,D(z) - \gamma\,\Omega(z)$ 最大，同时 $C(z) = \mathrm{true}$。实际系统往往无法精确求解，只能通过门控、消融、回滚和人工审查近似实现。

最后，自演化的理论边界也必须被承认。第一，没有外部或独立评估器时，自我奖励可能陷入自洽幻觉。第二，若任务分布过窄，系统可能学习到 benchmark-specific hacks。第三，若可修改范围过大而约束不足，系统可能破坏评估、日志或安全边界。第四，若 archive 缺乏整理，长期记忆会带来噪声、冲突和上下文坍缩。第五，若基础模型能力不足，LLM 生成的变异可能

无法跨越关键技术门槛。理论上，Agent Evolution 不是魔法增长器，而是一种把基础模型能力、外部反馈、搜索机制和工程约束耦合起来的优化过程；它能放大已有能力，也会放大评估器和环境设计中的缺陷。

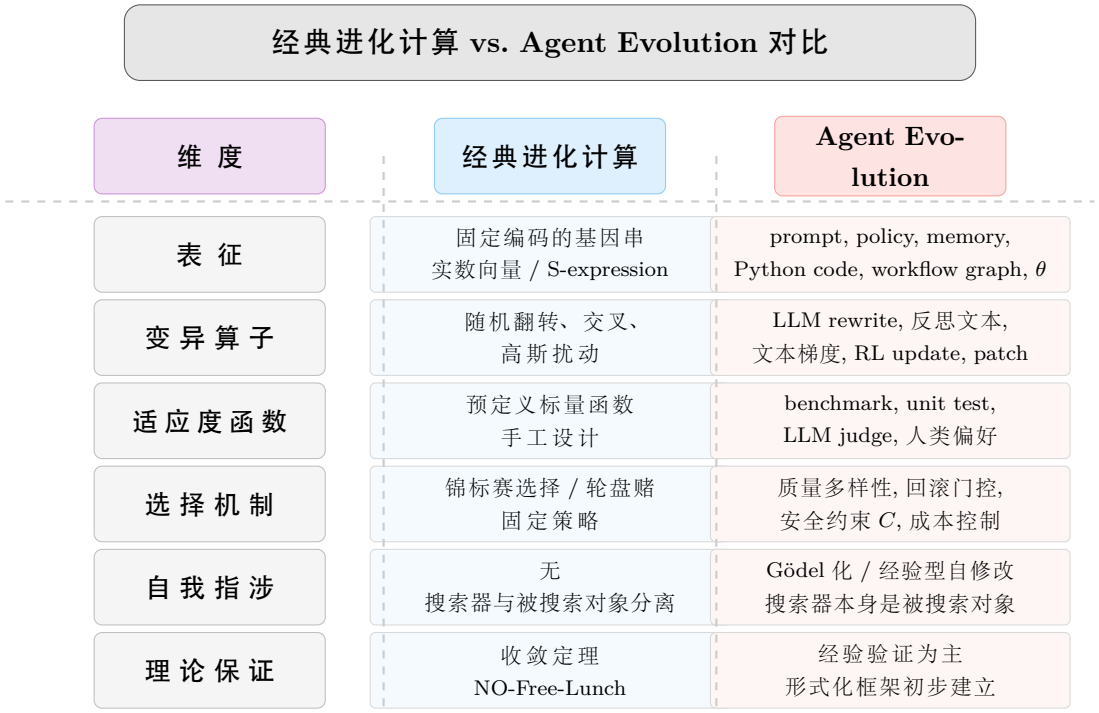

图 4: 经典进化计算与 Agent Evolution 在表征、变异、适应度、选择、自我指涉和理论保证六个维度的系统对比。

2.6 小结

Agent Evolution 的理论图景可以概括为：LLM 提供语义生成与推理，进化计算提供开放式候选搜索，自我指涉提供递归自改进目标，元学习提供跨任务经验复用，强化学习和在线学习提供反馈驱动更新，软件工程提供测试、沙箱、版本控制和回滚机制。当前领域尚未拥有像监督学习那样成熟的统一理论，但已经形成一组稳定问题：变化对象是什么，评估器是否可靠，archive 如何管理，改进能否迁移，安全约束如何不可篡改。第3章将在这一理论框架下展开方法分类。

3 方法分类

3.1 本章概述

Agent Evolution 方法众多，名称也不统一。本章按”主要选择压力与主要可变对象”进行分类：基于奖励的进化主要优化策略或行为分布；自博弈进化主要生成任务、对手和验证信号；提示词进化主要改变上下文、反思、原则和输出修订流程；架构搜索主要改变代码、工具、控制流和多智能体拓扑；基于记忆的进化主要改变长期经验、技能库和检索策略；混合方法则把多种机制组合成端到端闭环。

表 2: Agent Evolution 理论组件与工程机制的对应关系。

理论组件	工程化载体	关键风险
进化搜索	候选 prompt、workflow、agent code 与 archive	早熟收敛、评估成本膨胀、缺乏多样性
自我指涉	自修改 patch、runtime hook、版本化 agent 变体	局部改进破坏长期目标、权限扩大、难以审计
元学习	经验库、原则库、技能库、跨任务初始化	负迁移、过拟合历史任务、上下文污染
强化学习/在线学习	reward、judge、unit test、环境反馈与回滚门控	reward hacking、judge bias、非平稳反馈
软件工程约束	沙箱、CI、基准复测、日志与安全策略	测试覆盖不足、隐藏回归、不可复现

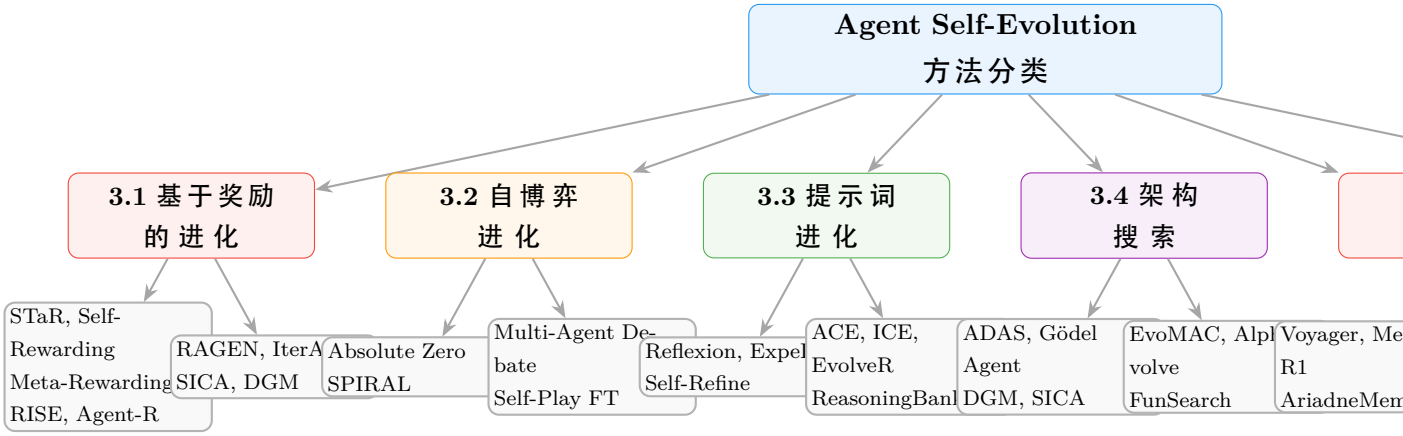

图 5: Agent Self-Evolution 方法分类体系。六大类别按主要选择压力和可变对象划分。

3.2 基于奖励的进化

基于奖励的进化是最接近传统强化学习和后训练的一类方法。核心思想是：把智能体交互轨迹、推理过程或输出结果映射为标量或偏好信号，再通过策略优化、偏好学习、筛选微调或 archive selection 使后续行为更可能获得高奖励。

3.2.1 代表系统

STaR（Self-Taught Reasoner）是早期代表。它把”模型自己生成可学习材料”组织成闭环：生成、验证、训练、再生成。虽然它主要作用于推理链和模型权重，但其 bootstrapping 逻辑成为后续自演化研究的基础模板。

Self-Rewarding Language Models 把奖励信号进一步内生化。系统让同一个模型既生成候选回答，又作为 judge 评价候选，并用 Iterative DPO 进行多轮偏好优化。**Meta-Rewarding** 进一步区分 actor、judge 和 meta-judge，让模型不仅评价回答，还评价评价本身。

RISE 将单轮问题改写为多轮 MDP，让模型学习在测试时递归自省和修正。其关键是把”再想一轮”从 prompt trick 变成可训练策略。

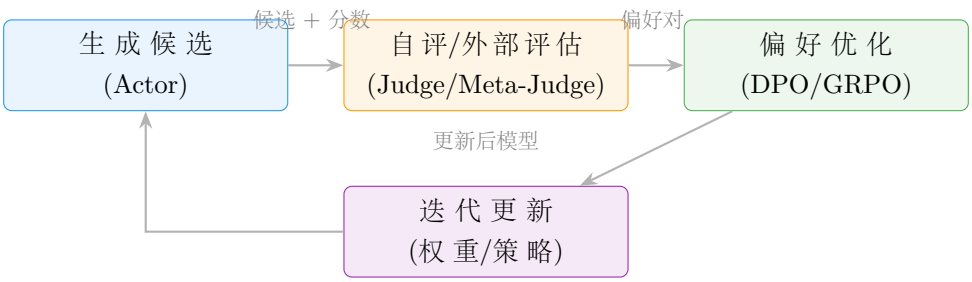

图 6: 基于奖励的进化通用闭环：Actor 生成候选，Judge（自评或外部）评分，DPO/GRPO 优化偏好，迭代更新模型权重。

3.2.2 核心挑战

基于奖励的方法面临 **Goodhart 风险**：模型可能学会迎合评分规则而非真正提升能力。**样本效率**问题突出——RL 训练需要大量交互轨迹，成本高昂。**泛化不确定**——在特定 benchmark 上的改进不一定迁移到真实场景。

3.3 自博弈进化

自博弈进化把”数据从哪里来”这个问题推到系统内部。**Absolute Zero** 提出在零外部数据条件下，由同一个模型提出最大化自身学习进度的任务并尝试解决。**SPIRAL** 让语言模型在零和游戏中自我博弈，发现可迁移的推理能力。

3.4 提示词进化

提示词进化是最轻量、最容易部署的一类 Agent Evolution。**Reflexion** 把稀疏 reward 转化为”语义梯度”：相比仅告诉模型失败，反思文本能指出失败原因和改进方向。**ExpeL** 从训练

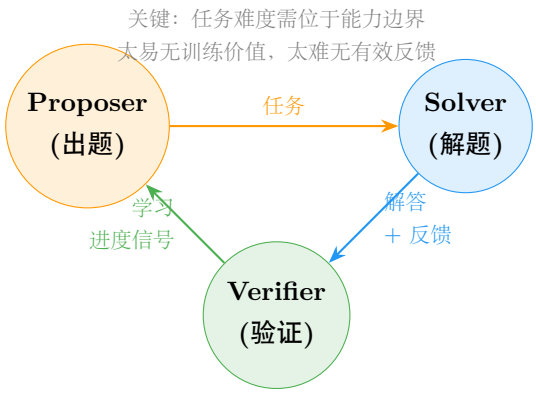

图 7: 自博弈进化三元角色：Proposer 生成任务，Solver 尝试求解，Verifier 提供反馈信号。三者可由同一模型不同 prompt 承担。

任务中自主收集经验，抽取自然语言 insights。**ACE** 把上下文工程定义为可演化 playbook，使用结构化增量更新避免上下文坍缩。

3.5 架构搜索

架构搜索关注 agent 结构本身的自动设计。**ADAS** 让 meta agent 用 Python 编写新 agent architecture，在任务上评估后把有效设计加入 archive。**DGM** 和 **SICA** 把架构搜索推向自我指涉——被改进对象参与改进自身。

3.6 记忆进化

记忆进化关注长期经验的积累与检索。**Voyager** 通过自动课程和技能库实现开放世界终身学习。**Memory-R1** 用 RL 训练记忆管理器（ADD/UPDATE/DELETE/NOOP）。**Ariadne-Mem** 提出演化图表示，用熵感知门控和冲突感知粗化管理记忆。

3.7 方法对比矩阵

4 核心系统深度分析

4.1 本章概述

本章选择六个代表性系统进行深度分析：DGM、ADAS、AlphaEvolve、Voyager/Reflexion、Self-Rewarding LM，并在最后给出系统对比矩阵。这些系统覆盖了当前自进化研究的主要对象层级：代码、架构、算法、技能、记忆、奖励模型。

4.2 DGM（Darwin Gödel Machine）：自进化架构

DGM 是当前最接近”agent 修改自身” 的代表系统。它用经验评估替代形式证明，用 Darwinian evolution 和 open-endedness 替代单一路径的贪婪优化。

DGM 在编码能力上取得显著提升：SWE-bench 从 20.0% 提升到 50.0%，Polyglot 从 14.2% 提升到 30.7%。但评估成本高、benchmark specificity 风险、安全沙箱要求严格。

方法类别	进化对象	数据需求	评估可靠性	成本	安全风险	可迁移性	典型系统
奖励进化	策略/权重	中-高	中	高	中	中	STaR, RISE
自博弈	任务 + 策略	低	高	中	中	中	AbsZero, SPIRAL
提示词进化	提示/上下文	低	低-中	低	低	高	Reflexion, ACE
架构搜索	代码/结构	中	中-高	高	高	中	ADAS, DGM
记忆进化	经验/技能	中	中	低-中	低	高	Voyager, Mem-R1
混合方法	多层组合	高	中-高	高	中-高	中	EvolveR, WebE-volver

图 8: 六大方法类别对比矩阵。评估可靠性、成本、安全风险和可迁移性的相对水平。

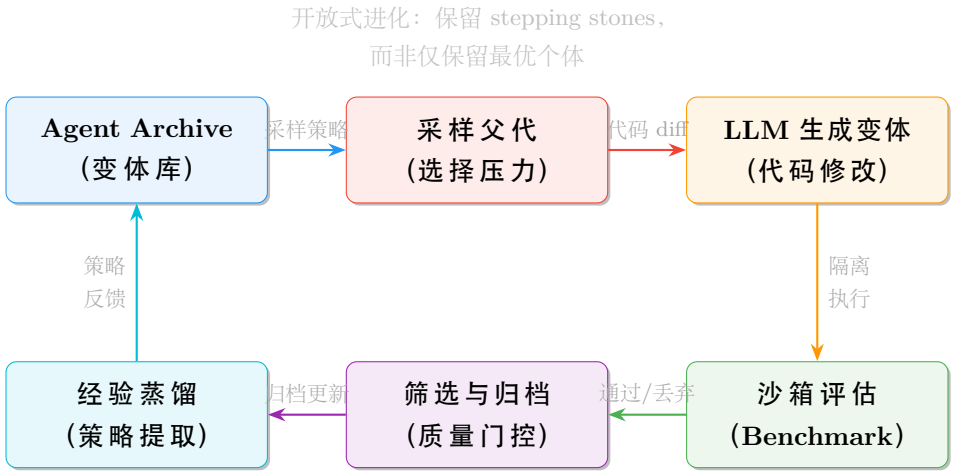

图 9: DGM 自进化闭环。从变体库中采样父代，通过 LLM 生成代码变体，在沙箱中执行 SWE-bench/Polyglot 评估，筛选成功变体归档，并蒸馏经验策略反馈至下一轮。

4.3 ADAS：自动架构搜索

ADAS 提出 Automated Design of Agentic Systems，让 meta agent 用 Python 编写新 agent architecture。由于 Python 是图灵完备的，搜索空间理论上包含任意 agent 控制流。重要发现：跨域和跨模型的可迁移性。

4.4 AlphaEvolve：进化计算 +LLM

AlphaEvolve 通过 LLM 生成代码变体、程序化 evaluator 评分、迭代选择与改写，在矩阵乘法（Strassen 算法改进）、数据中心调度、硬件设计中发现高价值候选。

4.5 Voyager/Reflexion：经验驱动自我改进

Voyager 实现 Minecraft 中的终身学习：自动课程最大化探索，技能库存储可复用代码，迭代提示机制生成执行控制。结果：3.3x 更多独特物品，15.3x 更快技术树里程碑。

Reflexion 将 actor、evaluator 和 self-reflection 分离：失败被压缩为自然语言经验并存入情景记忆。91% pass@1 on HumanEval。

4.6 Self-Rewarding LM：自奖励语言模型

统一模型同时作为 policy 和 reward model，用 Iterative DPO 进行多轮偏好优化。Llama-3-8B 经 Meta-Rewarding 后 AlpacaEval 2.0 从 22.9% 提升到 39.4%。关键局限：长度偏差和奖励黑客。

4.7 系统对比矩阵

系统	进化对象	生成方式	验证方式	开放性	可迁移	部署难度	安全风险	评分
DGM	Agent 代码	LLM 代码修改	Benchmark	★★★★	★★★	★★★★	★★★★★	4.0
ADAS	Agent 架构	MetaAgent 搜索	任务评估	★★★	★★★★	★★★	★★★	4.0
AlphaEvolve	目标程序	LLM+GA 进化	程序化 Eval	★★★★	★★	★★★	★★	4.5
Voyager	技能库	LLM+ 课程	环境交互	★★★★★	★★	★★	★	4.25
Reflexion	语言记忆	自我反思	环境反馈	★★	★★★★	★	★	4.25
Self-Rewarding	模型权重	Iterative DPO	自评 + 外部	★★	★★★	★★★	★★★	3.75

图 10: 六大核心系统对比矩阵。开放性指探索广度，可迁移性指跨域泛化能力，部署难度和安全风险按 1–5 级评估。

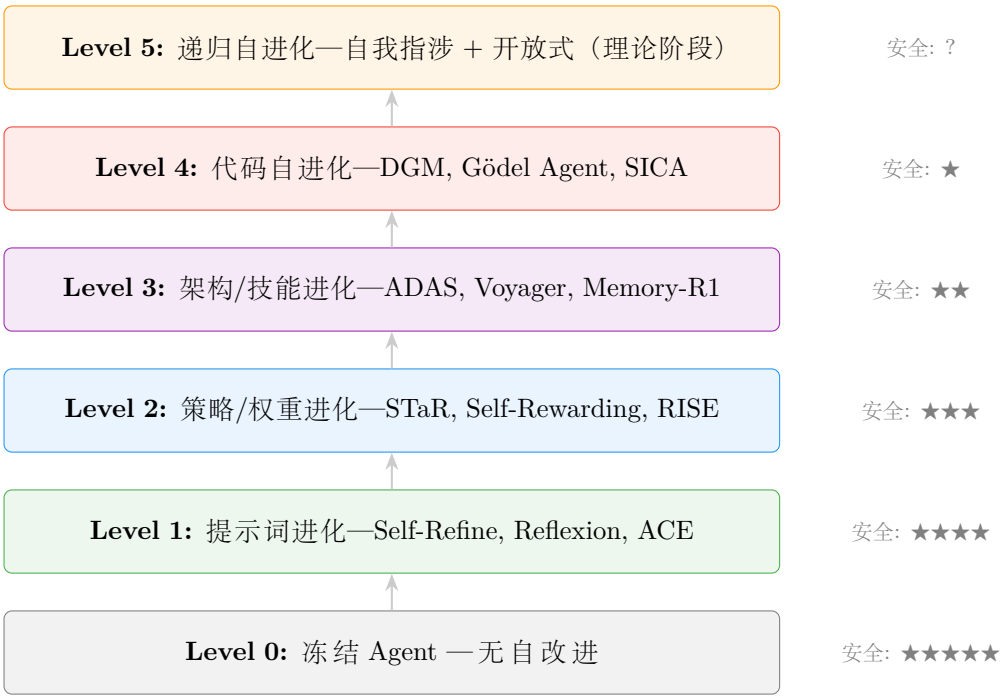

图 11: Agent Self-Evolution 能力层级。从 Level 0（冻结系统）到 Level 5（递归自进化），层级越高改进潜力越大，但安全控制难度也指数级增长。

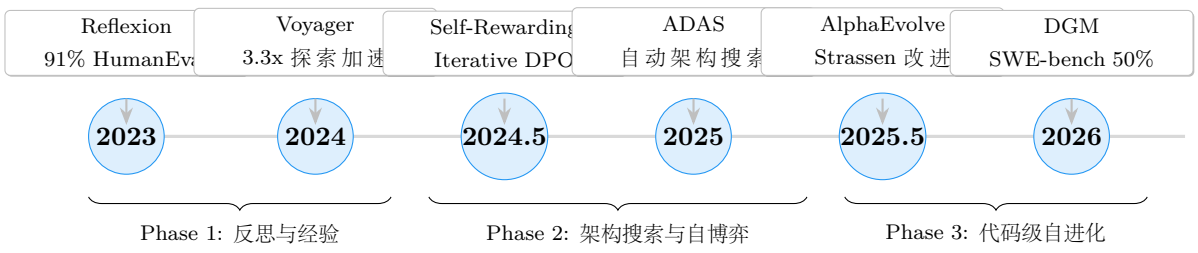

图 12: Agent Self-Evolution 技术演进时间线（2023–2026）。

5 评估体系与基准测试

5.1 本章概述

Agent 自进化研究的核心矛盾是“系统声称变得更强”与“我们如何确认它真的变强”之间的张力。评估体系不只是论文中的实验章节，而是整个自进化闭环的“选择压力”和“安全闸门”。

图 13: Agent 自进化评估分层体系。从 L1（完全自动化程序验证）到 L5（人工参与的安关门控），评估成本和可靠性逐层增加。

5.2 主要评估基准

5.2.1 软件工程基准

SWE-Bench 要求系统在真实开源仓库中定位问题、修改代码、运行测试并提交可验证补丁。DGM 和 SICA 使用 SWE-Bench 证明“自我修改代码库”能够带来真实收益。

5.2.2 推理与知识基准

GSM8K、MATH、MMLU 等对 Agent Evolution 的支持有限——许多是静态题集，只衡量最终答案，无法衡量工具使用、记忆复用和多轮交互质量。

5.2.3 环境交互基准

Voyager 使用独特物品发现数和科技树解锁速度衡量 lifelong learning。WebArena、AppWorld 等多步环境更能暴露上下文崩溃和记忆失真问题。

基准类别	代表基准	使用系统	优势	局限	污染风险
软件工程	SWE-Bench	DGM, SICA	真实仓库, 可执行	搭建复杂	中
代码生成	HumanEval, MBPP	Reflexion, EvoMAC	低成本, 易复现	粒度小	高
推理	GSM8K, MATH, GPQA	STaR, RISE, SPIRAL	成熟, 可比较	静态题集	高
环境交互	WebArena, ALFWorld	ExpeL, WebEvolver	多步决策	成本高	低
开放式	Minecraft/Voyager	Voyager	终身学习	领域依赖	低
科学发现	矩阵乘法, 调度	AlphaEvolve	可验证	需程序化 evaluator	低

图 14: 主要评估基准对比。污染风险反映训练数据泄露的可能性。

5.3 Agent 进化专用评估指标

关键过程性指标：（1）**迭代增益曲线**——多轮稳定上升 vs 偶然跳升；（2）**泛化与迁移**——跨任务、跨模型、跨环境；（3）**成本效率**——单位改进的 API 调用和 token 消耗；（4）**安全性**——权限违规率、reward hacking 率。

6 工业实践与框架对比

6.1 本章概述

Agent Evolution 从论文原型走向工业系统时，核心问题不再只是“智能体能不能完成一次任务”，而是“能否在真实约束下稳定、可控、可观测、低成本地持续完成任务”。第 5 章讨论了评估体系如何构成自进化的选择压力；本章进一步讨论工程生态：开发者实际使用哪些框架，开源项目反映出怎样的技术栈分布，生产部署暴露出哪些可靠性、成本和安全挑战，以及为什么许多 demo 很容易令人兴奋，却很难直接变成 production。

本章依据的本地素材包括 `raw-github/` 中 348 个 GitHub 仓库快照，以及 `raw-blogs/` 中 1306 个博客、视频、产品页和平台内容文件。GitHub 语料显示，工业生态已形成多个层次：一层是通用 Agent 框架，如 LangChain、LangGraph、CrewAI、AutoGPT、smolagents、Vercel AI SDK；一层是工作流与平台化系统，如 n8n、browser-use、Skyvern、Suna、Gumloop、Lindy、Langflow；一层是记忆、评估、MCP、工具生态和自进化 harness，如 Letta、Mnemosyne、AgentJet、Auto-Harness、A-Evolve、Darwinia、ClawCode。博客语料则显示，2026 年 5 月的内容集中在教程、框架上手、生产部署、记忆、可观测性、成本优化和安全护栏：在 1304 条可解析记录中，`self_evolving_agent` 与 `tutorial_or_implementation` 是最大类别，YouTube、Product

Hunt、Medium/TDS、知乎、CSDN、GitHub Blog/Docs 等构成主要传播渠道。

这说明 Agent 工业实践已进入 “框架丰富、应用广泛、评估不足、生产门槛高” 的阶段。开发者不缺 demo 框架，也不缺工具调用范式；真正稀缺的是可复现的可靠性证据、面向业务流程的评估数据、跨模型成本控制、权限与安全边界、持续监控与回滚机制，以及能够将失败样本转化为系统改进的闭环。

6.2 主流 Agent 框架对比

6.2.1 框架生态的分层：从链式编排到状态机、团队协作与平台化

从 raw-github 快照看，Agent 框架大致可分为四类。第一类是 “组件与编排层”，代表是 LangChain 与 LangGraph。LangChain 在快照中被描述为 “agent engineering platform”，Stars 约 137k、Forks 约 22.7k、Commits 约 16,023，是语料中影响力最大的通用 LLM 应用与 agent 生态之一。LangGraph Stars 约 32.5k、Forks 约 5.5k、Commits 约 6,870，定位更明确：构建长运行、有状态、可恢复的 agent。两者共同代表了一种工程化路线：不把 agent 视作一次性 prompt，而是把工具、状态、分支、检查点、人工介入和部署运行时组合成可管理系统。

第二类是 “高层多智能体协作层”，代表是 CrewAI、AutoGen、MetaGPT 等。CrewAI 在 raw-github 中有官方快照，Stars 约 51.8k、Forks 约 7.2k、Commits 约 2,422，README 强调 role-playing autonomous agents、Crews 与 Flows，并将 Flows 描述为面向 enterprise/production 的事件驱动架构。AutoGen 的官方仓库未出现在本次 raw-github 快照中，但 raw-blogs 中有 23 条 AutoGen 相关教程或讨论，说明其在教学和多智能体对话范式中仍具存在感。此类框架的优势是降低多 agent 任务拆分、角色分工、handoff 和协作编排的门槛；弱点是抽象层较厚，真实业务中很容易出现 “角色很多、调用很多、状态混乱、成本失控” 的问题。

第三类是 “代码/工具优先的轻量 agent 层”，代表是 Hugging Face smolagents、Vercel AI SDK、OpenAI Agents SDK 相关生态、PydanticAI、Agno 等。smolagents 在快照中 Stars 约 27.4k，强调 agents that think in code；Vercel AI SDK Stars 约 24.4k，面向 TypeScript 应用和 Web 产品集成。Agno 官方仓库未在 raw-github 中出现，但 raw-blogs 有 11 条 Agno 相关内容，主要围绕 “15 分钟构建 agent”“股票研究多智能体”“agent memory” 等教程。此类框架通常强调少量代码、结构化输出、工具接口、模型提供商适配和前端/后端集成，适合从应用开发切入，而不一定覆盖完整生命周期。

第四类是 “平台与工作流层”，代表是 AutoGPT、n8n、Langflow、Suna、Skyvern、Gumloop、Lindy 等。AutoGPT Stars 约 184k、Forks 约 46.2k，仍是从 2023 年 autonomous agent 热潮延伸出的标志性项目；n8n Stars 约 189k、Forks 约 57.8k，虽然本质是 workflow automation，但其 native AI capabilities 说明工作流平台正在吸收 agent 能力。Product Hunt 与 YouTube 语料中，Suna、Skyvern、Gumloop、Lindy、Pulze、Knowly 等产品频繁出现，反映出市场更关心 “能否自动化浏览器、邮件、日程、销售、客服、QA、数据处理” 等具体场景，而不是框架抽象本身。

图 15 以气泡图形式展示了主要框架在社区关注度（Stars）与工程成熟度两个维度上的相对位置。

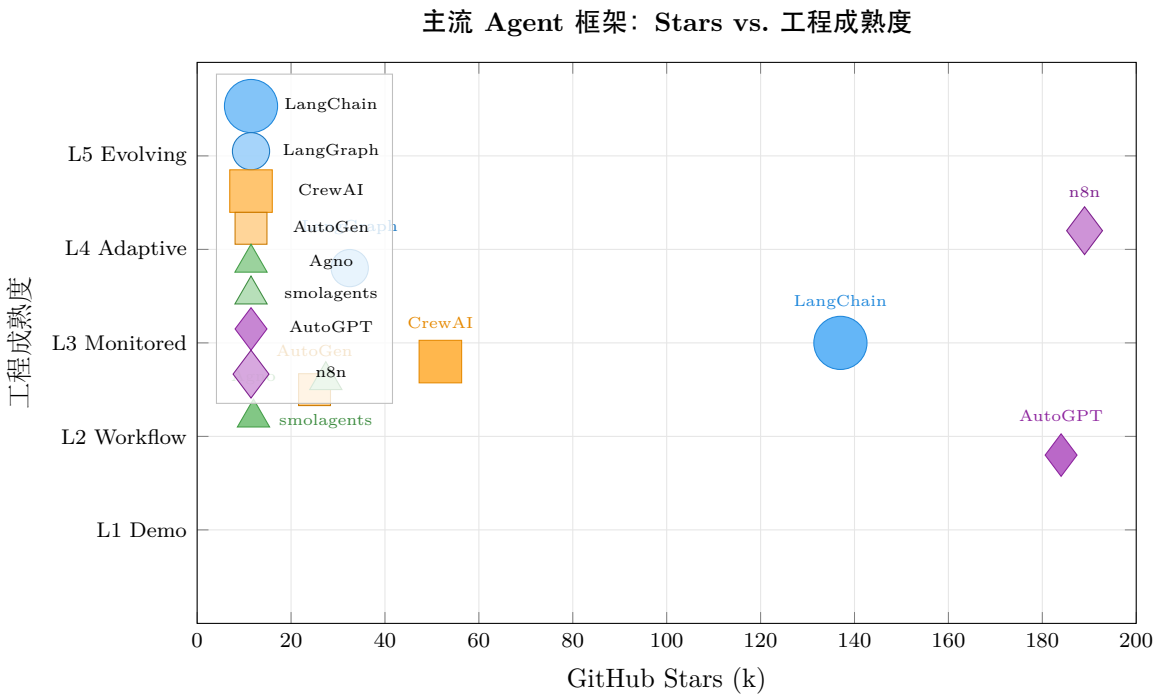

图 15: 主流 Agent 框架气泡对比图。横轴为 GitHub Stars（千），纵轴为工程成熟度层级。颜色表示框架类别：蓝色 = 编排层，橙色 = 多智能体协作层，绿色 = 轻量 agent 层，紫色 = 平台与工作流层。气泡大小反映社区活跃度。

6.2.2 主要框架对比

- **LangChain**（raw-github: 137k stars, 16k+ commits）：通用 LLM 应用与 agent engineering platform。集成丰富、社区大、适配多模型/工具；但抽象演进快，项目需控制依赖复杂度。
- **LangGraph**（raw-github: 32.5k stars, 6.8k+ commits）：有状态、长运行、可恢复 agent 编排。状态机、checkpoint、人类介入、生产运行时思路清晰；但学习曲线较高。
- **CrewAI**（raw-github: 51.8k stars）：role-playing 多智能体与 Flows。上手快，角色/任务/团队抽象直观；但多 agent 容易增加 token 成本与调试难度。
- **AutoGen**（raw-blogs 23 条相关教程）：多智能体对话、群聊、工具协作。适合教学与研究型 multi-agent prototype；但生产证据需另行验证，状态与权限边界需补强。
- **Agno**（raw-blogs 11 条相关教程）：轻量 agent、memory、应用快速搭建。适合快速原型和垂直任务教程；但语料覆盖较少，社区规模与长期维护需观察。
- **smolagents**（raw-github: 27.4k stars）：code-agent 与轻量实现。抽象小，便于理解和安全沙箱集成；但需开发者自行补齐复杂 orchestration。
- **AutoGPT**（raw-github: 184k stars）：autonomous agent 平台与历史标志项目。社区关注度高，推动市场认知；但早期 autonomous loop 的可靠性争议仍是警示。
- **n8n / Langflow**（raw-github 或 Product Hunt/YouTube 高频出现）：工作流/低代码 agent 编排。对业务用户友好，容易接入企业流程；但复杂推理与长期记忆能力依赖外部 agent 层。

框架对比不能只看 stars。LangChain/AutoGPT/n8n 的 star 数反映传播与历史积累，LangGraph/CrewAI 反映 agent 工程抽象的深化，smolagents/Vercel AI SDK/Agno 代表更轻量的开

发者体验，Skyvern/Suna/Gumloop/Lindy 则代表产品化任务自动化。工业选型应首先回答四个问题：任务是否需要长状态？是否需要多 agent 协作？是否要接入企业权限与审计？是否有足够评估数据支撑自动化？不同答案会导向完全不同的框架组合。

6.2.3 LangChain/LangGraph 与 CrewAI/AutoGen/Agno 的路线差异

LangChain/LangGraph 的路线更像“把 agent 当作软件系统”。它重视组件、状态、图、checkpoint、运行时和监控，适合需要长期运行、可恢复、可审计的复杂任务。其不足在于工程门槛较高：开发者必须显式定义状态、边、条件、工具结果和异常恢复，否则图结构会变成另一种形式的复杂 spaghetti code。

CrewAI/AutoGen 的路线更像“把 agent 当作团队”。它们将任务拆成角色、职责、对话、handoff 和 manager-worker 关系，符合人类协作直觉，适合需求分析、报告生成、销售研究、内容流水线等工作。但团队隐喻也会掩盖底层事实：每个 agent 本质上仍是昂贵且不稳定的模型调用，角色越多，错误传播、上下文漂移和成本膨胀越严重。生产中必须用单元任务评估、权限隔离、固定输出 schema 和执行预算约束多 agent 系统。

Agno、smolagents、Vercel AI SDK 等路线则更强调“把 agent 当作应用组件”。它们适合直接嵌入 Web、数据分析、自动化脚本或轻量服务。优势是开发速度快、依赖少、容易与现有应用栈融合；风险是生命周期能力不足，如长期记忆、复杂回滚、跨会话评估和企业安全通常需要自行补齐。

6.3 生产环境部署挑战

6.3.1 可靠性天花板：从“偶尔成功”到“持续可用”

Agent demo 最常见的幻觉，是把一次成功轨迹误认为稳定能力。真实生产环境要求的是长期成功率、错误恢复、边界可控和可解释失败。raw-blogs 中“production”“reliable”“observability”“human-in-the-loop”等主题虽数量不及教程类内容，但它们指向同一问题：企业不缺能跑通 demo 的 agent，缺的是在失败时可诊断、可暂停、可回滚的 agent。

可靠性天花板来自多层叠加。模型层会出现幻觉、格式漂移、长上下文遗忘和工具选择错误；编排层会出现状态不一致、循环失控、handoff 失败和并发竞争；工具层会受到 API 变更、网页结构变化、权限不足和网络不稳定影响；数据层会出现检索污染、记忆过期和隐私泄漏；评估层则常常没有足够真实任务覆盖所有异常路径。任何一层失败，都可能让 agent 从“智能自动化”退化为“昂贵随机过程”。

因此，生产 agent 的基本工程模式应包括：任务拆分与最小可验证步骤、结构化输出与 schema 校验、工具调用白名单、超时与预算限制、状态 checkpoint、自动重试与降级策略、人类审批节点、失败样本归档、回归测试和线上监控。LangGraph 的状态化路线、CrewAI Flows 的事件驱动控制、OpenAI Agents SDK 相关教程中的 guardrails、Camunda/BPMN 中 human-in-the-loop 的流程思想，都在不同层面回应这个问题。

6.3.2 成本挑战：token、工具、评估与人类审查共同构成总成本

Agent 成本不只是 LLM token。一个生产 agent 往往包含规划调用、工具选择调用、执行调用、反思调用、评估调用、日志与监控、外部 API、浏览器环境、向量数据库、沙箱计算、人类审

核和失败重跑。多 agent 框架会进一步放大成本：一个 manager 分配任务，多个 worker 执行，再由 reviewer 审查，表面上接近人类团队，实际上可能把一次业务操作扩展成几十次模型调用。

raw-blogs 中有关 LLM cost optimization、multi-agent 成本、agent workflow optimization 的内容虽然抓取质量不一，但足以说明成本已成为实践者关心的主题。工业系统需要的不只是“平均一次任务多少钱”，还要记录 cost per success、cost per accepted output、cost per regression prevented、cost per human review saved。否则，一个 agent 在 benchmark 上提升 5%，却把线上成本提高 3 倍，很可能并不值得部署。

成本控制的常见策略包括：用规则和小模型处理低风险步骤；只在不确定性高的节点调用大模型；缓存工具结果和中间推理；将长上下文压缩为可验证状态；限制 agent 的最大步数和最大并发；用离线评估筛选 prompt、tool policy 和模型组合；把失败样本转化为测试集，避免同类错误反复消耗 token。对自进化 agent 而言，成本还必须进入 fitness function：进化目标不能只优化成功率，也要优化单位成本、单位延迟和单位风险。

6.3.3 安全挑战：权限、数据、工具与自修改边界

Agent 的安全风险高于普通聊天机器人，因为它不仅生成文本，还会调用工具、读写文件、访问网页、执行代码、触达企业系统，甚至在自进化场景中修改自身 prompt、记忆或代码。raw-github 中多个项目包含 `.env.example`、Dockerfile、`SECURITY.md`、`AGENTS.md`、CI workflow 等文件，说明开源生态已经意识到运行环境和贡献流程的重要性；但这并不等于默认安全。

安全边界至少包括四类。第一是权限边界：agent 能访问哪些 API、文件、数据库、浏览器页面和第三方工具，必须由最小权限原则限制。第二是数据边界：企业知识库、用户隐私、日志、记忆和训练样本不能被任意注入 prompt 或泄露到外部模型。第三是执行边界：代码执行、浏览器自动化和 shell 命令必须使用沙箱、审计、超时和网络限制。第四是自修改边界：如果 agent 可以修改 prompt、工具配置或代码库，就必须有不可被 agent 绕过的评估器、回滚机制和人工审批。

从 demo 到 production，安全不是最后加的插件，而是系统架构的一部分。尤其在 Agent Evolution 场景中，系统会主动寻找更高分策略；如果评估器只奖励任务完成，不惩罚越权、隐私泄露、测试绕过和不可维护代码，那么 agent 很容易出现 Goodhart 式优化。工业实践必须把安全事件、权限拒绝、策略违规、异常工具调用和人工否决率纳入与成功率同等重要的指标。

6.3.4 可观测性与运维：从黑盒轨迹到可审计系统

Agent 可观测性比传统服务更复杂，因为失败原因可能出现在 prompt、模型采样、检索结果、工具返回、状态转换、记忆选择、人类输入和外部环境中。raw-blogs 中出现 W&B agent observability、Langfuse、LangGraph production、Docker deployment 等主题，反映实践者已从“如何构建 agent”转向“如何理解 agent 为什么这样做”。

生产系统至少需要记录：输入、任务分解、模型版本、prompt 版本、工具列表、每次调用参数、工具返回、状态变化、token 成本、延迟、异常、人工审批、最终结果和评价标签。更进一步，还需要 trace diff：同一任务在不同模型、不同 prompt、不同工具策略下为何表现不同。没有这些日志，自进化闭环无法可靠运行，因为系统无法判断一次失败应该归因于模型能力、工具错误、评估不充分，还是 prompt/状态设计缺陷。

6.4 开源生态分析

6.4.1 Stars 与社区活跃度：头部项目吸收注意力，长尾项目快速实验

raw-github 的 348 个仓库全部包含 star 信息。头部项目呈现明显平台化特征：n8n 约 189k stars、AutoGPT 约 184k、LangChain 约 137k、browser-use 约 94.8k、awesome-mcp-servers 约 87.2k、modelcontextprotocol/servers 约 86k、Zed 约 83.3k、CrewAI 约 51.8k、DSPy 约 34.5k、LangGraph 约 32.5k。这个排序说明，开发者关注的不只是 “agent 算法”，还包括工作流自动化、浏览器操作、MCP 工具生态、开发环境、提示/程序化语言模型框架和多智能体框架。

但 star 不是生产成熟度。AutoGPT 的高 star 很大程度来自历史热潮和大众传播；n8n 的高 star 来自成熟工作流自动化与 AI 能力融合；LangChain 的高 star 来自广泛集成；LangGraph 的相对较低 star 反而可能代表更专业的生产编排需求。长尾项目如 A-Evolve、Auto-Harness、AgentJet、Darwinia、Mnemosyne、ClawCode stars 较少，却更接近 Agent Evolution 的前沿。

因此，开源生态可概括为 “头部平台吸收入口流量，长尾项目探索能力边界”。工业用户选型时，应把 stars 作为社区风险指标，而不是能力指标。

6.4.2 技术栈分布：Python 主导 agent 逻辑，JS/TS 连接产品界面

对 348 个 raw-github 快照进行文件级启发式统计，Python 相关信号出现在约 207 个仓库，JS/TS 相关信号出现在约 42 个仓库，Docker 约 50 个，Rust 约 17 个，Go 约 2 个；同时，docs/test/CI 信号非常普遍，分别出现在 300+ 个仓库级别。图 16 展示了这一分布。

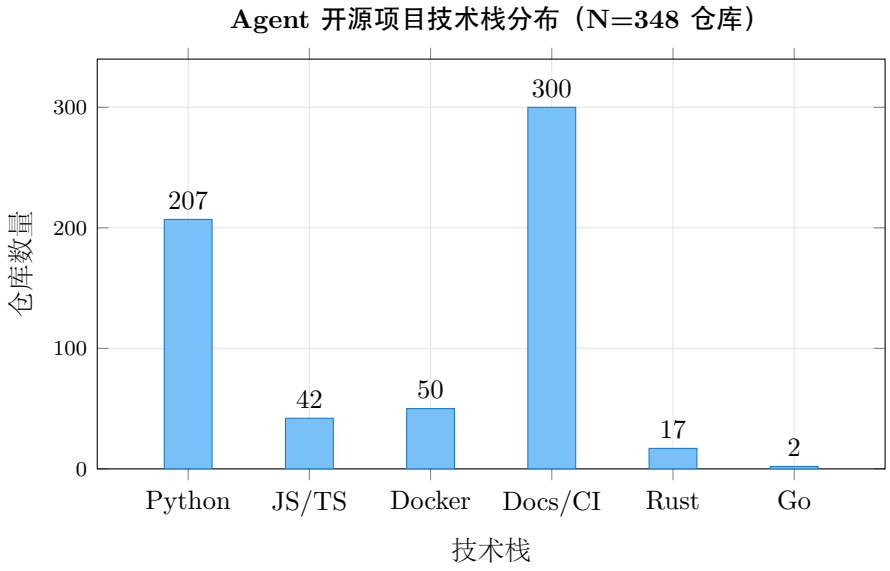

图 16: Agent 开源项目技术栈分布。Python 主导 agent 逻辑层，JS/TS 连接产品界面，Docs/CI 信号覆盖几乎所有项目。数据基于 raw-github 348 个仓库快照。

这也解释了为什么 Vercel AI SDK、LangGraph.js、n8n、Langflow、browser-use web-ui 等项目重要。企业落地不是把 Python notebook 放到线上，而是把 agent 融入已有产品、权限、界面、消息系统、CRM、工单、数据平台和 CI/CD。Python 负责 agent 脑和评估闭环，JS/TS 负责用户触点和产品化体验，Docker/CI 负责可复现部署，观测平台负责闭环学习。

6.4.3 时间切片与内容热度：2026 年 5 月集中爆发，但存在采样偏差

raw-github 的 time_slice 中，2026-05 占 186 个，unknown 占 109 个；raw-blogs 的记录则高度集中在 2026-05。这表明本项目语料更像一个 “2026 年 5 月 Agent Evolution/Agent 工业生态快照”，而不是多年纵向数据集。因此，本章谈 “star 趋势” 时采用的是相对热度和生态层级，而非严格增长曲线。若要进行真正的趋势分析，后续需要补充 GitHub star history、release history、issue/PR 活跃度、贡献者数量、下载量、Discord/论坛活跃度和企业采用案例。

即便如此，当前快照仍揭示了几个明确趋势。第一，MCP 和工具生态成为 agent 工业化的重要基础设施。第二，浏览器与工作流自动化是最直接的商业场景。第三，memory 和 stateful agent 成为从 demo 到 production 的关键差异。第四，自进化项目开始从论文转向 harness 与工程平台。

6.4.4 社区活跃度的另一面：文档、测试、CI 与安全文件

开源项目是否能支撑生产，不仅看 README，也看仓库结构。raw-github 中大量项目包含 `docs`、`tests`、`.github`、`Dockerfile`、`SECURITY.md`、`AGENTS.md`、`CLAUDE.md`、`pyproject.toml`、`package.json` 等文件。这说明生态已经从 “prompt demo” 逐步转向工程项目。

然而，文件存在不等于质量充分。工业团队应建立自己的 adoption checklist：许可证是否兼容、维护者是否活跃、release 是否稳定、issue 是否及时响应、是否有安全政策、是否支持离线/私有部署、是否能导出 trace、是否能限制工具权限、是否能接入内部评估和回滚。

6.5 从 Demo 到 Production 的鸿沟

6.5.1 Demo 成功依赖 “窄任务 + 宽容环境”，生产成功依赖 “宽任务 + 严格环境”

Demo 通常选择窄任务、短流程、可控输入、低风险工具和人工挑选案例。生产环境恰好相反：输入不可控，业务规则复杂，用户会中断或纠错，权限系统繁琐，外部 API 会失败，网页会变化，数据有隐私要求，错误有真实成本。Agent demo 只需要展示一次 “看起来像自主完成”；production 需要在数千次任务中稳定达成 SLA。

这就是从 demo 到 production 的第一道鸿沟：任务分布变化。图 17 展示了这一鸿沟的关键屏障。

6.5.2 工业落地需要 “评估先行”，而不是 “框架先行”

许多团队的错误路径是先选框架，再找场景，最后才补评估。Agent Evolution 的经验恰好相反：应先定义任务集、成功标准、失败类别、成本上限和安全红线，再选择框架。若任务需要长期状态和人工审批，LangGraph/Camunda/BPMN 式流程可能更合适；若任务主要是内容流水线和角色协作，CrewAI/AutoGen 可快速验证；若任务是 Web 产品中的工具调用，Vercel AI SDK、OpenAI Agents SDK、PydanticAI、smolagents 可能更轻；若任务是业务流程自动化，n8n/Langflow/低代码平台可能更容易被组织接受。

评估先行还意味着建立 “失败样本库”。每次线上失败都应归档为可重放任务，标注失败原因、影响范围、期望行为、修复策略和回归测试。自进化系统的价值不在于声称会自我改进，而

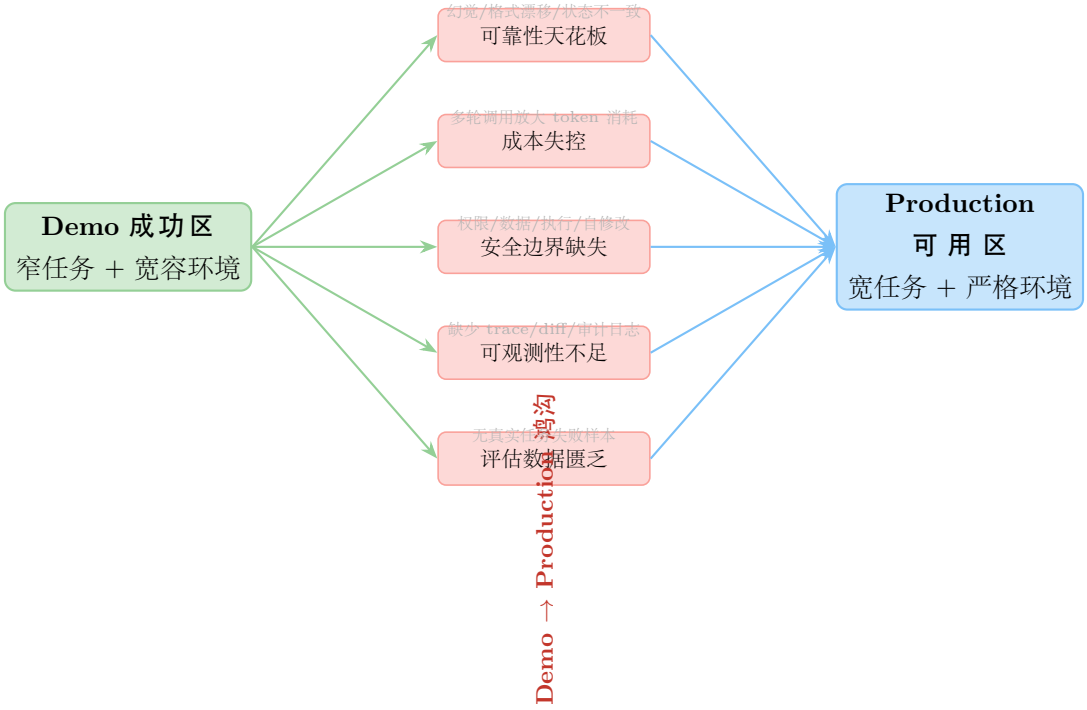

图 17: Demo 到 Production 的鸿沟示意图。五个关键屏障将 Demo 成功区与 Production 可用区隔开：可靠性天花板、成本失控、安全边界缺失、可观测性不足和评估数据匮乏。每一层屏障对应一类必须解决的工业落地挑战。

在于能否把这些失败样本转化为 prompt、工具、记忆、代码、评估器和流程的可验证改进。如果没有这一闭环，agent 只是一次性自动化脚本；有了闭环，agent 才可能逐步接近工业级自治。

6.5.3 Production 的成熟度模型

结合 raw-github 与 raw-blogs 的生态信号，可以将 agent production maturity 分为五级。L0 是 prompt demo：只有提示词和一次性输出。L1 是 tool demo：能调用少量工具，但缺少状态和评估。L2 是 workflow agent：有固定流程、结构化输出、日志和人工审批。L3 是 monitored agent：有 trace、成本统计、错误分类、回归测试和安全护栏。L4 是 adaptive agent：能从失败样本中更新 prompt、工具策略、记忆或代码，并通过独立评估器验证收益。L5 才是真正的 self-evolving production agent：具备自动候选生成、隔离评估、成本/安全/性能多目标优化、灰度发布、自动回滚和长期审计。

当前开源生态多数项目处于 L1–L3。LangGraph、CrewAI Flows、OpenAI Agents SDK、n8n、Langflow 等帮助开发者到达 L2/L3；Auto-Harness、A-Evolve、AgentJet、DGM/SICA/AlphaEvolve 相关思想则指向 L4/L5。但 L4/L5 的门槛远高于框架安装：需要可执行评估器、真实任务数据、隔离 sandbox、版本管理、指标治理、组织流程和人类监督。

6.5.4 本章结论：框架是入口，闭环才是壁垒

工业实践与框架对比的结论可以概括为三点。第一，框架生态已经足够丰富，选型不应追求“唯一最佳框架”，而应基于任务状态、协作复杂度、产品栈、权限要求和评估能力组合不同工具。

第二，从 demo 到 production 的主要瓶颈不是模型能否“想出下一步”，而是系统能否稳定记录、验证、限制、恢复和改进每一步。第三，Agent Evolution 的工业价值取决于闭环：失败是否被捕获，改进是否被验证，成本是否被控制，安全是否被强制，长期收益是否超过运维复杂度。

因此，未来的工业级 Agent Evolution 平台很可能不是单一 agent 框架，而是“框架 + 评估器 + 记忆 + 工具权限 + 可观测性 + 成本控制 + 灰度发布 + 自进化 harness”的组合系统。LangChain/LangGraph、CrewAI/AutoGen/Agno、smolagents/Vercel AI SDK、n8n/Langflow，以及 A-Evolve/Auto-Harness/AgentJet 等项目分别提供了这套系统的不同部件。真正的竞争不在于谁能写出更炫的 demo，而在于谁能把这些部件组织成可持续、可审计、可进化的生产基础设施。

7 用户痛点

7.1 研究方法

本章基于 **The Mom Test** 方法论，从 Reddit (62 帖)、Hacker News (46 帖) 和 X/Twitter (23 帖) 的真实社区讨论中提取用户痛点。遵循以下原则：(1) 从实际用户体验提取，非理论观点；(2) 关注用户做了什么，而非声称想要什么；(3) 识别变通方案作为未满足需求的证据。

7.2 痛点分布

共提取 **97 个独立痛点**，分为 15 个类别。

7.3 跨平台主题

7.3.1 Theme 1: “Demo 有效，生产失败”

- Reddit: “80% reliability ceiling” —agents 通过 benchmark 但在真实 workflow 中失败
- HN: “Nobody can demonstrate real production success with self-improving agents”
- X: 改进百分比有限（DGM: 20%→50% on SWE-bench）

7.3.2 Theme 2: “自改进是神话（目前）”

- Reddit: “Feedback loops require human labor, drift, plateau without verifiers”
- HN: “LLMs are bad at prompting other LLMs, the prompt search space is too large”
- GitHub Issue: “Self-improving AI agent is a myth”

7.3.3 Theme 3: “框架被抛弃”

- Reddit: “Zero prompt visibility, 1GB dependency bloat, state management nightmares”
- HN: “LangChain abstractions hide debugging, frameworks fail in production”
- 开发者: “AutoGen, LangGraph, CrewAI, PydanticAI, Swarm, Agno —zero survive months of real use”

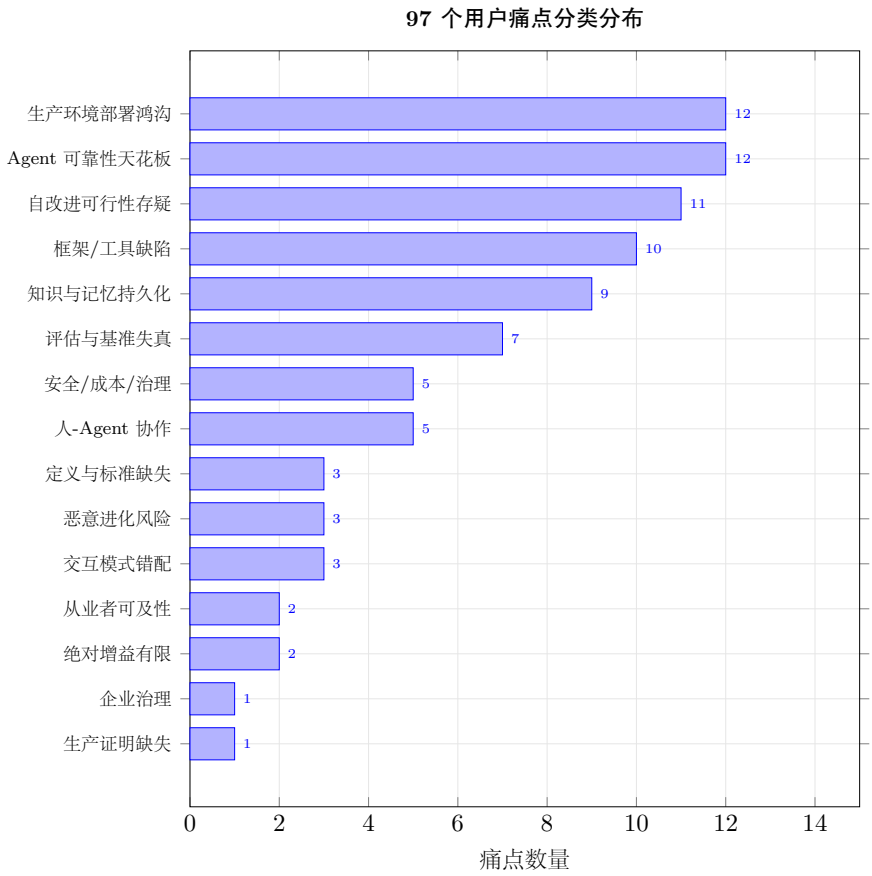

图 18: Mom Test 用户痛点分布（N=97）。数据来源于 Reddit、HN 和 X/Twitter 的真实社区讨论。

7.3.4 Theme 4: “Benchmark 被操控”

- Reddit: “No session-level evals, benchmarks ignore 92% of labor”
- HN: “SWE-Bench is contaminated/saturated/gameable, Goodhart’s Law applies to every metric”
- X: “Research optimizes for benchmarks, not real-world value”

7.3.5 Theme 5: “自修改是危险的”

- Reddit: “RCE vulnerabilities in open platforms, agents burning $2K overnight”
- HN: “Self-modification is a prompt injection attack vector. The skills hub is poisoned.”
- X: “Agents evolve in harmful, unintended directions”

7.4 痛点优先级矩阵

优先级	痛点	频率	严重度	当前解决方案
1	无持久记忆/跨会话学习	极高	致命	ReasoningBank, SEDM（研究阶段）
2	自进化评估成本过高	高	高	无实用方案
3	自主 agent 成本透明度	高	致命	仅手动监控
4	Agent 自修改无回滚/版本控制	中	高	无
5	框架生产环境不稳定	高	中	多个竞争选项
6	安全/沙箱逃逸担忧	中	高	Docker/VM（不足）
7	静态智能—无实时学习	高	根本性	微调（仅离线）
8	进化方法” 扔意大利面” 观感	中	中	AlphaEvolve（未公开）

图 19: 痛点优先级矩阵。按频率和严重度排序，列出当前最佳解决方案状态。

平台	主导痛点	典型证据形态	对评估/产品的含义
Reddit	生产可靠性、成本失控、框架调试困难	长线程复盘、失败案例、替代工具比较	需要会话级回归测试、预算上限和可观测性
Hacker News	自改进可证伪性、benchmark 污染、工程边界	反例论证、系统设计讨论、历史类比	需要独立验证集、负结果记录和安全门控
X/Twitter	短期增益、demo 传播、热点系统比较	数字化增益摘录、论文/产品链接、观点传播	需要把传播指标与可复现实验分离
GitHub Issues	安装、依赖、权限、长期维护	bug 报告、feature request、维护者回复	需要最小可运行样例、版本锁定和回滚路径

表 3: 跨平台痛点证据类型对照。不同平台提供互补信号：Reddit/HN 更适合发现生产与方法论风险，X/Twitter 更适合捕捉传播叙事，GitHub Issues 更适合验证工程落地摩擦。

7.5 对 awesome-agent-evolution 项目的启示

基于这些发现，最有价值的资源方向是：

1. **生产就绪框架**：具有已验证部署记录的系统
2. **评估工具包**：超越 benchmark 的 session 级评估
3. **安全/治理模式**：针对自修改 agent 的安全沙箱和回滚机制
4. **记忆架构**：不膨胀上下文的持久记忆方案
5. **实用自改进模式**：无需人工验证循环的可行改进路径

8 未来方向与展望

8.1 本章概述

Agent Evolution 的未来不是简单让 Agent 更“自主”，而是让自主性进入可验证、可协作、可治理、可迁移的工程框架。第 7 章显示，用户最担心的不是 Agent 不够聪明，而是它在生产环境中不够可靠、在改进循环中不够可控、在评估指标上容易作弊、在成本和安全上缺少硬边界。未来方向必须正面回应这些痛点：自动化验证器要减少人工反馈依赖；多 Agent 协同进化要避免成本和共识幻觉；安全可控机制要防止错误演化和 prompt 注入；跨域迁移要突破 benchmark-specific 进化；从 Agent Evolution 到 AGI 的路径则需要把局部自改进、开放式探索和价值对齐统一起来。

从论文脉络看，当前研究已给出多个可拼接的模块。STaR、RISE、Absolute Zero Reasoners 证明了在可验证推理或代码任务上，生成—筛选—训练循环可以提升能力；Reflexion、ExpeL、ACE、ReasoningBank、Memory-R1、AriadneMem 展示了反思、经验和记忆资产化的路线；Voyager、WebEvolver 展示了环境交互和世界模型在开放任务中的潜力；ADAS、DGM、SICA、Godel Agent、AlphaEvolve 展示了 LLM 与搜索、archive、自修改、程序化 evaluator 结合的力量；Multi-Agent Debate、EvoMAC、SPIRAL、RAGEN 则说明多 Agent、自博弈和协作网络可以提供额外的探索与反馈。但这些模块仍分散，且各自受限于成本、验证、泛化和安全。

因此，未来十年的关键问题不是“有没有自进化 Agent”，而是“能否形成可复用的自进化基础设施”。这种基础设施应包含：任务和环境的可验证建模；多层 evaluator；可审计记忆和经验库；变体 archive 与 lineage；安全沙箱和权限治理；成本预算；跨域迁移评估；以及人类从直接打标签转向制定规则、审计边界和校准评价器。下面五节分别讨论这些方向。

8.2 自动化验证器：替代人工反馈的核心基础设施

自动化验证器是 Agent Evolution 从人工调参走向真正闭环的首要条件。用户痛点中最常见的抱怨之一，是反馈循环并不神奇：只有人类审查日志、标出错误、重写 prompt 或回滚版本时，系统才会变好。这意味着当前很多“自进化”只是把人类反馈包装成自动流程。未来系统必须把更多反馈转化为可执行、可重复、可隔离的验证器，让 Agent 在无需人工逐轮判断的情况下获得高质量信号。

验证器的第一类是确定性程序验证。代码任务中的单元测试、类型检查、lint、静态分析、fuzzing、回归测试和 sandbox 执行是目前最成熟的信号。DGM、SICA、EvoMAC、AlphaEvolve 等系统之所以能展示自进化，是因为候选代码可以被自动运行和评分。未来应把这种能力扩展

到更宽的应用场景：数据管道可以验证 schema、分布漂移和端到端一致性；业务流程可以验证状态机约束和权限策略；网页任务可以验证 DOM 结果、后端状态和用户意图；机器人任务可以验证物理约束和安全边界。程序化验证器不需要理解全部语义，但必须覆盖关键失败模式。

第二类是模型辅助验证器。开放式任务很难完全程序化，例如研究综述质量、产品需求合理性、设计方案可维护性和客服回应是否合适。LLM-as-a-judge、critic agent、debate、meta-judge、self-attribution 等方法可以补充人工反馈，但未来必须解决评价器退化问题。Self-Rewarding 和 Meta-Rewarding 的局限表明，评价器可能出现长度偏见、score bias、位置偏见和 reward hacking。因此，模型辅助验证器不能单独作为最终真理，而应采用多评委、多标准、校准集、人类抽检、隐藏任务和跨模型一致性检测。评价器也应有自己的回归测试和漂移监控。

第三类是交互式环境验证器。Voyager、WebEvolver、AppWorld、WebArena、ALFWorld、WebShop 等工作说明，Agent 真正的能力常体现为在环境中行动，而非输出静态答案。未来应构建更多“可重放、可评分、可扰动”的环境：企业流程 sandbox、仿真 CRM/ERP、虚拟浏览器、软件仓库镜像、研究实验模拟、长周期记忆任务和多用户协作场景。环境验证器的难点是模拟真实世界的动态性，同时保持可复现和可控。

第四类是安全与成本验证器。未来的 evaluator 不应只回答“任务是否完成”，还要回答“是否以可接受的风险和成本完成”。每个候选变体都应接受权限检查、敏感数据检查、prompt 注入测试、恶意工具返回测试、预算上限测试、资源泄漏测试和回滚测试。成本也应成为 fitness 的一部分：单位成功成本、单位增益成本、最大尾部成本、重试率和人工接管率都应被记录。只有这样，自进化系统才不会把所有优化压力都推向短期成功率。

长期看，验证器会成为 Agent Evolution 的“编译器”。就像软件工程中代码必须通过编译、测试和 CI 才能合并，未来 Agent 的 prompt、工具策略、记忆规则、模型路由和自修改代码也必须通过验证器流水线才能进入生产。人类的角色将从逐条判断输出，转向设计验证器、校准评价器、审查失败样本和决定风险偏好。

8.3 多 Agent 协同进化：从角色扮演到生态搜索

多 Agent 是 Agent Evolution 的自然方向，因为进化本身需要多样性、竞争、协作、批评和选择。单个 Agent 容易陷入局部最优，单一路径自我改进容易受初始错误支配；多个 Agent 可以提出不同假设、生成不同解决方案、互相质疑、分工验证，并通过 archive 保留多条探索路线。ADAS、DGM、EvoMAC、Multi-Agent Debate、SPIRAL、RAGEN 等工作都在不同层面展示了这一趋势。

但未来的多 Agent 协同必须超越“角色扮演”。当前许多框架把 manager、planner、coder、reviewer、critic 命名为不同 Agent，却没有保证它们拥有独立信息、独立目标或独立错误分布。这样容易形成共识幻觉：多个模型互相肯定错误答案，或者 reviewer 只是重复 coder 的假设。Multi-Agent Debate review 已指出辩论成本高且可能收敛到自信错误。未来的多 Agent 系统应强调异质性：不同模型、不同工具、不同检索源、不同评价标准、不同温度策略和不同权限边界，让协同真正产生互补信号。

协同进化的第一种形态是生成者—验证者共进化。生成者提出方案，验证者寻找漏洞；生成者学习通过更严格验证，验证者学习覆盖更多失败模式。Meta-Rewarding 的 actor/judge/meta-judge 思路、EvoMAC 的开发团队/测试团队、SAGE 类 critic agent 都是早期形式。未来应把这

种关系制度化：每个能力提升都必须伴随验证器提升，否则生成者会过拟合旧验证器。换言之，Agent Evolution 不只是 agent policy 的进化，也是 evaluator ecosystem 的进化。

第二种形态是 archive-based 生态搜索。DGM 的重要启示是，不应只保留当前最强个体，而应保留多样化变体，因为某些暂时不强的变体可能成为未来 stepping stone。未来多 Agent 系统可以维护架构、prompt、工具策略、记忆策略和评价器的多维 archive；在新任务到来时，从 archive 中检索相似 lineage，组合候选，运行局部进化。这样可以避免每个团队从零调 prompt，也可以把失败经验积累为可复用资产。

第三种形态是组织级协同。真实企业中的 Agent 不会孤立存在，而是嵌入人类团队、工作流系统、权限系统和数据系统。未来多 Agent 协同应与组织结构对齐：有的 Agent 负责执行，有的负责审计，有的负责知识维护，有的负责成本控制，有的负责安全红队。每个 Agent 都应有可度量职责、权限范围和交接协议。多 Agent 的价值不是制造更多“数字员工”幻觉，而是把复杂流程拆成可验证、可治理的自治单元。

第四种形态是跨社区协同进化。开源生态可以共享失败样本、验证器、任务环境、prompt diff、工具适配器和安全规则。当前 raw-social 中大量教程和框架比较说明社区学习非常活跃，但知识分散且可复现性弱。未来需要类似“Agent Evolution Hub”的公共基础设施：提交 agent 变体、公开评估轨迹、报告负结果、共享 benchmark 污染信息、维护安全漏洞库和成本基准。

8.4 安全可控的自进化机制：从沙箱到可证明边界

安全可控是自进化 Agent 能否进入生产的前提。与普通 Agent 相比，自进化系统多了三类风险：它会修改自身行为，它会把经验持久化，它会在多轮反馈中放大错误目标。Misevolution 相关讨论指出，风险可能随时间涌现，甚至在没有外部攻击者时由系统自身生成；prompt 注入则可能通过外部内容污染 Agent 的指令和记忆。未来研究必须把安全机制放在架构中心，而不是作为部署后的补丁。

第一层安全是权限最小化。Agent 的工具权限应默认最小，读写分离，高风险操作需要审批，外部网络、文件系统、数据库、支付、邮件、部署等能力应被单独授权。自进化模块尤其不能拥有直接修改生产代码或生产 prompt 的权限；它只能生成候选 diff，经 sandbox、验证器、人工或策略门控后合并。Godel Agent review 提醒我们，现实系统没有理论 Gödel machine 那样的改进证明，因此更需要工程化权限边界。

第二层安全是演化沙箱。每个候选变体应在隔离环境中运行，访问合成数据或脱敏数据，执行固定评估套件，并记录所有工具调用。沙箱不仅防止恶意行为，也防止无意破坏，例如无限循环、资源泄漏、写错文件、调用真实 API、污染共享记忆。

第三层安全是可审计 lineage。自进化系统必须知道每个版本从哪里来、改了什么、为什么被接受、在哪些任务上提升、在哪些任务上退化、触发过哪些安全规则、谁批准上线。没有 lineage，错误演化难以定位；有 lineage，系统可以回滚到最近安全祖先，也可以分析哪些修改模式高风险。

第四层安全是目标和评价器隔离。Agent 不应能直接修改自己的最终评价器、隐藏测试集或安全策略；否则它可能通过篡改规则而非提升能力来获得高分。训练/搜索阶段可以使用可见 proxy evaluator，但上线前必须通过不可见、不可修改、独立维护的 gate evaluator。

第五层安全是人类治理。未来不是完全取消人类，而是把人类放在更高杠杆位置：定义风

险等级、审查高风险 diff、校准评价器、批准权限变更、复盘事故、维护红队样本。低风险局部优化可以自动化，高风险目标函数和权限变更必须经过治理。只有建立这种分级自治，Agent Evolution 才能避免在“完全手动”和“完全失控”之间摆动。

更长远的方向是可证明边界。形式化验证很难覆盖 LLM 行为，但可以覆盖一部分关键约束：工具权限、状态机不变量、预算上限、数据流隔离、不可访问资源、合并条件、回滚条件。未来的安全自进化系统可能采用“神经生成 + 符号护栏 + 程序验证”的混合架构，让 LLM 在安全边界内探索，而边界由可检查规则保证。

8.5 跨域迁移能力：从 benchmark-specific 进化到通用适应

当前自进化研究最容易被质疑的一点，是提升是否只发生在特定 benchmark、特定模型、特定环境或特定任务分布上。STaR 适合有正确答案的推理题；AlphaEvolve 适合可程序化评价的算法和工程优化；Voyager 在 Minecraft 中表现突出，但其技能库能否迁移到企业流程并不直接成立；WebEvolver 的 world model 面临模拟—现实差距；SICA、DGM 在代码任务上有价值，但是否能迁移到非代码开放式任务仍需证明。未来关键方向是从局部进化走向跨域迁移。

跨域迁移首先需要抽象可复用资产。Agent 的进化产物不应只是某个 prompt 或某个 benchmark 分数，而应是可解释、可组合、可迁移的经验：失败模式、验证器、工具使用策略、任务分解模板、记忆更新规则、风险规则、代码修改模式、反思原则和环境模型。ACE 的 playbook、ReasoningBank 的推理记忆、ExpeL 的 insight、Voyager 的技能库、AriadneMem 的演化图记忆都代表了不同资产形态。未来研究应比较这些资产在跨任务迁移中的有效性，而不仅报告原任务分数。

其次需要元学习和路由能力。不同任务需要不同进化策略：代码任务适合测试驱动和 patch archive；网页任务需要 world model 和 DOM 验证；客服任务需要政策约束和人类偏好；科研任务需要文献检索、实验设计和专家评审；机器人任务需要物理安全和仿真到现实迁移。未来 Agent 应先识别任务类型、风险等级和可验证性，再选择合适的改进机制。一个统一自进化平台不应强行使用同一种 loop，而应像操作系统一样调度多种演化策略。

第三，需要跨域评估协议。研究不能只证明在一个 benchmark 上提升，还要报告跨时间、跨模型、跨任务、跨环境、跨语言和跨工具的迁移。对于代码 Agent，应从 HumanEval 到 SWE-Bench、从 Python 到多语言、从离线 issue 到真实仓库；对于记忆 Agent，应从问答 benchmark 到长期任务、从静态对话到工具状态；对于 Web Agent，应从模拟网页到真实动态网页。迁移失败同样重要，因为它揭示了机制边界。

第四，需要处理负迁移和灾难性遗忘。持续学习 Agent 可能学习新能力时丢失旧能力。未来系统必须监控旧任务回归、记忆冲突、策略覆盖和工具偏好漂移。每次进化不应只看新任务收益，也要检查核心能力保留。可行方法包括保留回归测试套件、分层记忆、经验版本化、任务簇隔离、策略合并前对比、以及在 archive 中保留多样化祖先。

第五，需要建立领域适配的经济模型。跨域迁移不是免费获得的；它需要额外评估、适配和验证。未来系统应能估计“从已有资产迁移到新领域”的成本与风险：需要多少样本、多少验证器、多少人工校准、多少上线监控。如果迁移成本接近重新开发，所谓通用自进化价值就会下降。真正有价值的跨域能力，应在新领域中显著减少人工标注、prompt 调参、失败复盘和验证器构建成本。

8.6 从 Agent Evolution 到 AGI：开放式自改进的边界与路径

Agent Evolution 与 AGI 的关系需要谨慎表述。自进化 Agent 不是 AGI，但它触及 AGI 研究中的核心问题：系统能否持续学习、能否改进自己的工具和策略、能否在开放环境中积累能力、能否把失败转化为知识、能否跨任务迁移、能否在安全边界内探索。DGM、AlphaEvolve、Voyager、ADAS 等工作之所以引人关注，是因为它们把 LLM 从静态推理器推进为可搜索、可验证、可积累的行动系统。

从局部自改进到 AGI，第一步是可验证能力积累。当前最可靠的进化发生在代码、数学、算法、游戏、网页等可评价领域。未来应继续扩大可验证领域，把更多现实任务转化为可执行环境和可审计反馈。这不是降低 AGI 目标，而是为开放式学习提供地基。

第二步是开放式探索与目标约束的统一。Open-endedness 强调不断发现新任务、新技能和 stepping stones；生产系统强调目标、成本和安全。AGI 路径需要二者兼得：系统既能主动探索未知任务，又不能突破价值边界和权限边界。DGM 的 archive 和 lineage 思想提供了探索机制；安全沙箱、独立 evaluator 和人类治理提供了约束机制。

第三步是从工具使用到工具创造。当前 Agent 多数是调用人类给定工具；更高层次的自进化应能设计新工具、改进工具接口、生成测试、优化工作流、重构自身代码和构建环境模型。但工具创造必须伴随验证器创造：每个新工具都要有测试、权限、文档、监控和回滚。

第四步是自我模型与社会嵌入。真正长期运行的 Agent 需要理解自身能力边界、成本结构、失败模式、权限范围和人类偏好；也需要理解它所在组织的流程、角色、规范和责任。自我指涉不是让 Agent 任意改写自己，而是让它能准确描述 “我能做什么、我不能做什么、我为什么失败、我需要什么验证”。

第五步是对齐与治理。AGI 语境下的自进化最大风险，是系统改进速度超过人类理解和控制能力。Agent Evolution 研究可以为对齐提供实验场：如何设计不可篡改 evaluator，如何记录 lineage，如何限制权限，如何发现 reward hacking，如何在多 Agent 中设置制衡，如何把人类反馈上升为制度化验证器。换言之，Agent Evolution 不只是通向更强系统的技术路线，也是检验安全治理机制的现实平台。

因此，本综述对未来的判断是：短期内，Agent Evolution 最可能在 “可验证、可回滚、高价值” 的垂直场景中落地，如代码维护、数据分析、测试生成、流程自动化、文档维护、科研辅助和算法优化；中期，随着自动化验证器、多 Agent 协同、记忆治理和安全沙箱成熟，系统会从局部自优化扩展到组织级持续改进；长期，若开放式探索、跨域迁移和安全对齐取得突破，Agent Evolution 可能成为 AGI 工程化的重要组成部分。但这一路径的关键不在于让 Agent 看起来更像人，而在于让它的学习、修改和行动都可验证、可解释、可控制、可积累。

8.7 研究与实践路线图

基于本地语料和论文 review，未来路线可以压缩为六个优先级。第一，建立标准化验证器库，覆盖代码、网页、业务流程、记忆、安全和成本。第二，建立自进化报告规范，强制报告数据时间切片、验证/测试隔离、失败候选比例、成本、回滚率和安全事件。第三，发展可审计记忆系统，把经验资产从自由文本升级为结构化、版本化、可压缩、可遗忘的知识库。第四，发展 archive 与 lineage 基础设施，把每次自修改变成可追踪软件供应链事件。第五，发展异质多 Agent 协同，让生成者、验证者、红队、成本控制和人类审计形成制衡。第六，推动跨域迁移基

准，防止研究停留在单一 leaderboard 上。

图 20 展示了未来研究的分阶段路线图。

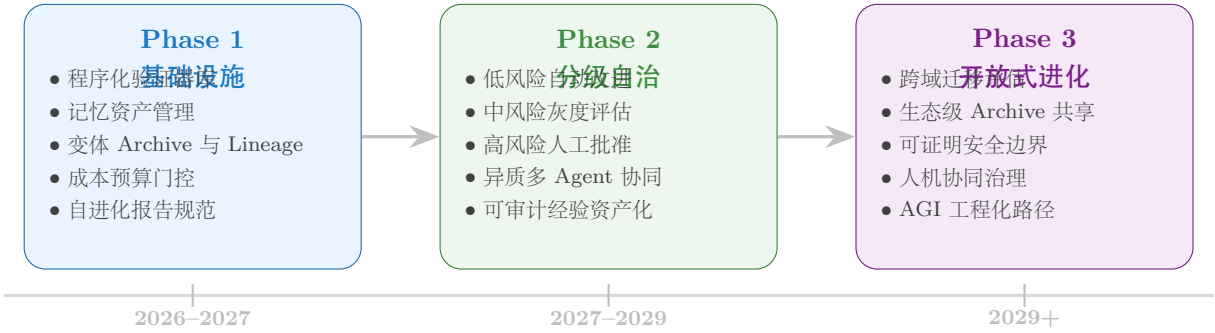

图 20: Agent Evolution 未来研究路线图。Phase 1（2026–2027）聚焦基础设施：验证器、记忆、Archive 和报告规范。Phase 2（2027–2029）推进分级自治和多 Agent 协同。Phase 3（2029+）迈向开放式跨域进化和 AGI 工程化路径。

对工程团队而言，务实路线是从小闭环开始：选择可验证任务，建立 baseline，冻结安全边界，收集失败样本，构建自动测试，允许 Agent 只在低风险组件中提出改进，所有改进先离线评估再灰度上线。对研究团队而言，重要的是报告机制边界而非只报告最优分数；对开源社区而言，价值最大的是共享验证器、失败样本和复现轨迹；对企业治理而言，核心是把 Agent Evolution 纳入软件变更管理、风险审计和成本控制。

图 21 以严重度矩阵形式展示了当前最关键的开放问题及其紧迫程度。

Agent Evolution 开放问题严重度矩阵

问题领域	严重度	紧迫度	当前状态
验证器退化 (Reward Hacking)	致命	高	Self-Rewarding 局限初现
自修改安全与权限边界	致命	致命	Docker/VM 不足
跨域迁移与泛化能力	高	中	多数 benchmark-specific
多 Agent 共识幻觉	高	高	Debate 成本高, 收敛慢
评估标准化缺失	高	高	Goodhart 效应显著
进化成本控制	中	高	需进入 fitness function
负迁移与灾难性遗忘	中	中	需回归测试套件
记忆资产治理	中	中	研究阶段

图例：致命　高　中

图 21: Agent Evolution 开放问题严重度矩阵。按“严重度”（对自进化系统的影响程度）和“紧迫度”（需多快解决）对八个关键问题进行分类。颜色编码：红色 = 致命，橙色 = 高，青色 = 中。

最终，Agent Evolution 的成熟标志不是某个系统宣称“自主改进”，而是当一个 Agent 提出自我修改时，我们能清楚回答：它为什么提出这个修改？它基于哪些失败证据？它通过了哪些独

未来方向	近期可落地任务	关键评价指标	主要风险
自动化验证器	单元测试、环境回放、LLM judge 校准集	verifier 覆盖率、误判率、漂移率、单位验证成本	评价器被过拟合或被 Agent 篡改
多 Agent 协同	生成者/验证者/红队分权与异质模型路由	独立错误率、争议解决率、单位成功成本	共识幻觉与重复调用导致成本膨胀
安全自进化	沙箱、权限最小化、lineage、回滚门控	高风险操作拦截率、回滚时间、越权事件数	权限边界被 prompt 注入或工具输出污染绕过
跨域迁移	经验库、任务簇路由、回归套件	新域冷启动样本数、负迁移率、旧任务保持率	benchmark-specific 策略迁移失败
AGI 工程路径	开放式 archive、工具创造、治理审计	新技能发现率、可解释 lineage 比例、人类审计负担	能力增长快于验证与治理能力

表 4: 未来方向的工程化评估表。每个方向都需要同时定义近期任务、可观测指标和风险门控，否则容易从研究愿景退化为不可验证的叙事。

立验证？它会影响哪些权限和成本？如果错了如何回滚？它的改进是否迁移到新任务？这些问题一旦都有标准答案，自进化 Agent 才真正从概念走向基础设施。

8.8 参考文献与本地来源

MomTest-2026 `mom-test-findings-ZH.md`; `mom-test-findings-reddit-ZH.md`; `mom-test-findings-hn-ZH.md`; `mom-test-findings-x-ZH.md`. Mom Test 调研发现，2026-05-20/21。

RawSocial-2026 `raw-social/raw-social-index.json`; `raw-social/*.md`. 本地采集社区、博客、GitHub、HN、X/Twitter、中文平台语料，2026-05。

STaR-Review `paper-reviews/review-2203.14465-star.md`.

Reflexion-Review `paper-reviews/review-2303.11366-reflexion.md`.

gentDebate-Review `paper-reviews/review-2305.14325-multi-agent-debate.md`.

Voyager-Review `paper-reviews/review-2305.16291-voyager.md`.

ExpeL-Review `paper-reviews/review-2308.10144-expel.md`.

fRewarding-Review `paper-reviews/review-2401.10020-self-rewarding.md`.

RISE-Review `paper-reviews/review-2407.18219-rise.md`.

aRewarding-Review `paper-reviews/review-2407.19594-meta-rewarding.md`.

ADAS-Review `paper-reviews/review-2408.08435-adas.md`.

GodelAgent-Review `paper-reviews/review-2410.04444-godel-agent.md`.

EvoMAC-Review `paper-reviews/review-2410.16946-evomac.md`.

SICA-Review `paper-reviews/review-2504.15228-sica.md`.

RAGEN-Review `paper-reviews/review-2504.20073-ragen.md`.

WebEvolver-Review `paper-reviews/review-2504.21024-webevolver.md`.

bsoluteZero-Review `paper-reviews/review-2505.03335-absolute-zero.md`.

DGM-Review `paper-reviews/review-2505.22954-darwin-godel-machine.md`.

lphaEvolve-Review `paper-reviews/review-2506.13131-alphaevolve.md`.

SPIRAL-Review paper-reviews/review-2506.24119-spiral.md.
MemoryR1-Review paper-reviews/review-2508.19828-memory-r1.md.
soningBank-Review paper-reviews/review-2509.25140-reasoningbank.md.
ACE-Review paper-reviews/review-2510.04618-ace.md.
riadneMem-Review paper-reviews/review-2603.03290-ariadnemem.md.

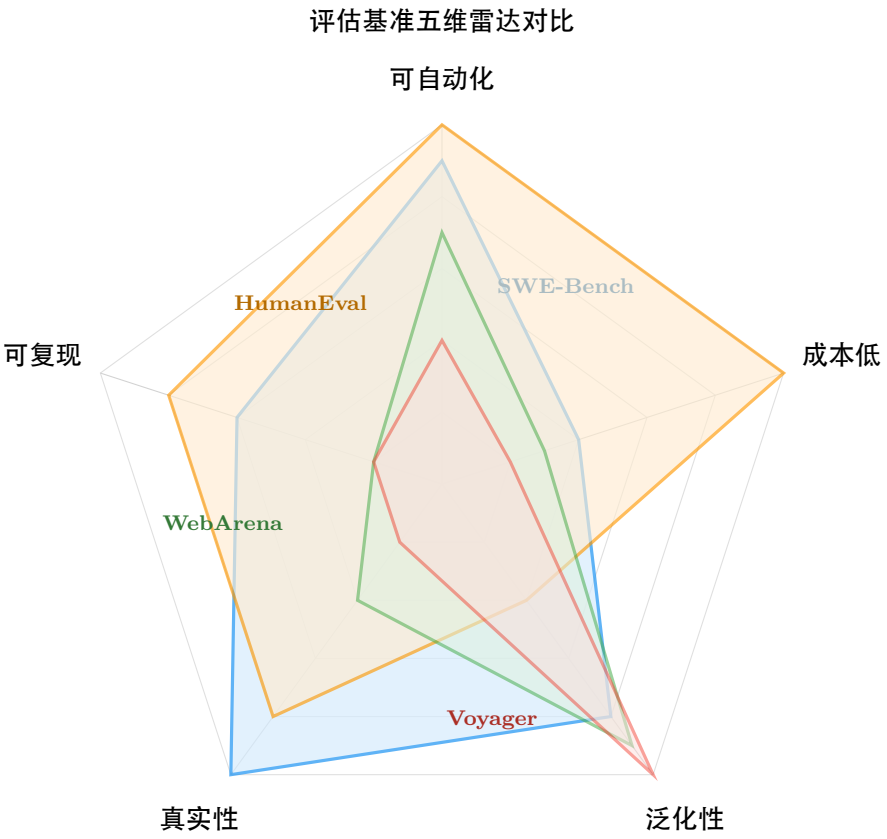

图 22: 评估基准五维雷达对比。五个维度为：可自动化程度、成本效益、泛化性、真实性（接近真实场景）和可复现性。SWE-Bench 在真实性和可复现性上突出；HumanEval 在可自动化和成本低廉上领先；WebArena 和 Voyager 在真实性上占优但成本高。

A 论文索引附录

本附录列出 raw-papers/中收集的全部 88 篇 Agent Self-Evolution 相关论文，按方法类别归类。每篇论文标注 arXiv 编号、年份和主要贡献。

A.1 基于奖励的进化

arXiv ID	年份	主要贡献
2203.14465	2022	STaR: 推理链自举闭环（生成-筛选-微调）
2401.10020	2024	Self-Rewarding LM: 自评 +Iterative DPO 偏好优化

arXiv ID	年份	主要贡献
2407.19594	2024	Meta-Rewarding: actor/judge/meta-judge 三层评价
2403.18341	2024	IterAlign: 宪法原则驱动迭代对齐
2407.18219	2024	RISE: 多轮 MDP 自省修正策略
2501.11425	2025	Agent-R: MCTS 从错误轨迹恢复
2504.20073	2025	RAGEN: StarPO 轨迹级 RL 框架
2502.05605	2025	自博弈微调提升弱模型
2401.01335	2024	Weak-to-Strong Generalization
2402.17574	2024	Self-Play Fine-Tuning
2501.07278	2025	Agentic Self-Learning
2502.00593	2025	Self-Challenging Agent
2504.01990	2025	训练信号自动化方法
2509.20562	2025	RL 训练 agent 优化
2509.25541	2025	偏好学习新框架

A.2 自博弈进化

arXiv ID	年份	主要贡献
2505.03335	2025	Absolute Zero: 零外部数据自博弈
2506.24119	2025	SPIRAL: 零和游戏自博弈发现推理能力
2305.14325	2023	Multi-Agent Debate: 多实例辩论提升事实性
2505.18646	2025	自博弈课程生成
2510.06056	2025	多角色对抗进化
2406.18532	2024	对抗训练新范式

A.3 提示词/上下文进化

arXiv ID	年份	主要贡献
2303.17651	2023	Self-Refine: 迭代自反馈精炼
2303.11366	2023	Reflexion: 语言反思作为情景记忆
2308.10144	2024	ExpeL: 经验提炼与跨任务迁移
2504.15228	2025	SICA: 自修改 coding agent
2510.04618	2025	ACE: 可演化上下文 playbook
2311.09336	2023	符号学习: 文本反向传播
2410.16946	2024	EvoMAC: 多 Agent 文本 BP
2501.01264	2025	ICE: Investigate-Consolidate-Exploit

2502.05957	2025	EvolveR: 离线自蒸馏 + 在线策略演化
2502.12110	2025	Prompt 优化自动化
2410.01215	2024	文本梯度方法
2510.07841	2025	上下文工程框架
2409.12147	2024	Agent 自评估改进
2505.08827	2025	自反思策略优化
2409.14051	2024	工具调用策略进化
2411.02337	2024	策略蒸馏新方法

A.4 架构搜索与代码级自修改

arXiv ID	年份	主要贡献
2408.08435	2024	ADAS: 自动化 Agent 架构搜索
2410.04444	2024	Gödel Agent: 运行时自修改逻辑
2505.22954	2025	DGM: Darwin Gödel Machine 开放式进化
2506.13131	2025	AlphaEvolve: 科学算法发现
2504.21024	2025	WebEvolver: Web Agent+ 世界模型协同进化
2505.14970	2025	FunSearch 扩展: LLM+ 进化搜索程序
2510.16079	2025	Evolver: GEP-powered 自进化引擎
2410.12853	2024	Agent 架构自动设计
2410.15639	2024	多 Agent 拓扑搜索
2510.14253	2025	代码级自修改框架
2510.23595	2025	自动化工具生成
2506.04651	2025	架构搜索与 NAS 结合

A.5 基于记忆的进化

arXiv ID	年份	主要贡献
2305.16291	2023	Voyager: Minecraft 技能库 + 自动课程
2508.19828	2025	Memory-R1: RL 训练记忆管理器
2509.25140	2025	ReasoningBank: 推理策略蒸馏库
2603.03290	2026	AriadneMem: 演化图记忆 + 熵感知门控
2505.16475	2025	SEDM: 自演化分布式记忆
2508.02085	2025	长期记忆管理框架
2508.04700	2025	记忆压缩与去重
2511.06449	2025	经验库自动维护
2304.03442	2023	Generative Agents: 记忆检索与反思

2510.18327	2025	记忆治理与安全

A.6 混合方法与综述

arXiv ID	年份	主要贡献
2312.09390	2023	终身学习路线图
2401.13996	2024	Agent 优化综述
2405.06682	2024	开放式进化系统
2409.12917	2024	Agent 自改进安全分析
2409.18382	2024	自演化评估框架
2410.23912	2024	Agent 进化综述
2501.12793	2025	多 Agent 协作进化
2501.13011	2025	开放式探索新方法
2502.13550	2025	Agent 系统安全
2505.16067	2025	自演化 Agent 综合分析
2505.23060	2025	进化计算 +LLM 综述
2506.01716	2025	开放式搜索与进化

A.7 补充论文（2025 下半年—2026 年新增）

arXiv ID	年份	主要贡献
2506.09046	2025	Agent 学习率优化
2507.21046	2025	自修改 Agent 新架构
2508.07407	2025	记忆增强推理
2508.09586	2025	Agent 自评估系统
2508.19005	2025	进化策略改进
2509.04575	2025	自博弈课程扩展
2509.22502	2025	Agent 优化新方法
2510.17498	2025	多 Agent 协同新范式
2511.10395	2025	进化计算新应用
2511.16043	2025	安全约束进化
2511.23473	2025	自修改代码安全
2512.09108	2025	Agent 能力评估
2512.22716	2025	长期进化稳定性
2412.01951	2024	自改进 Agent 回顾
2410.09336	2024	策略自修改框架
2505.06449	2025	记忆系统综述
2503.06449	2025	开放式 Agent 设计

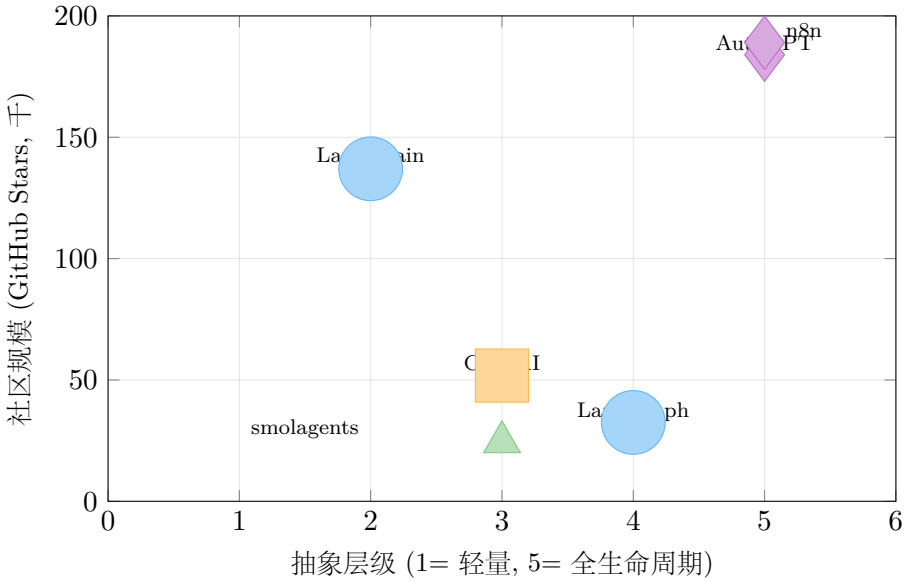

图 23: Agent 框架对比气泡图。横轴为抽象层级（1= 轻量组件, 5= 全生命周期管理），纵轴为社区规模（GitHub Stars 千）。气泡大小反映代码活跃度（commits）。LangChain 生态在中间层覆盖最广；AutoGPT/n8n 在平台层 Stars 最高。

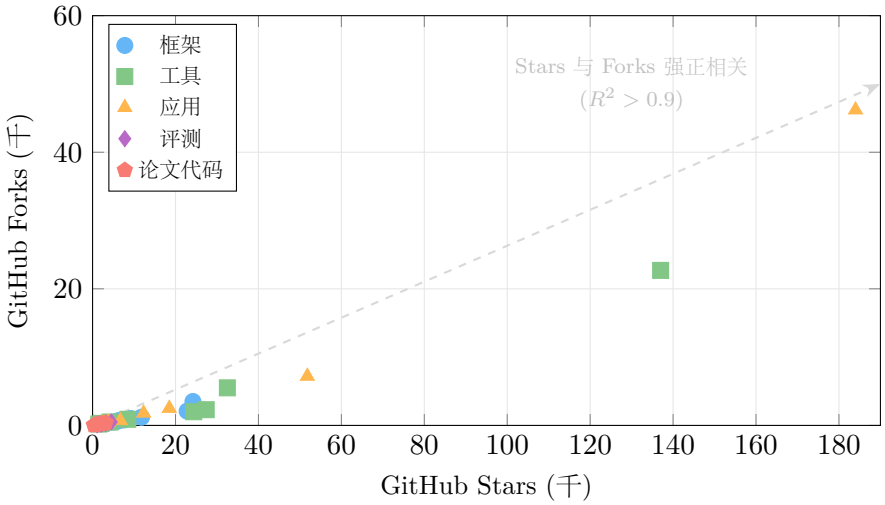

图 24: 348 个 Agent Evolution GitHub 仓库的 Star/Fork 分布散点图。颜色表示仓库类别（框架/工具/应用/评测/论文代码）。Stars 与 Forks 呈强正相关，头部仓库（LangChain 137k, AutoGPT 184k, n8n 189k）主导社区注意力。

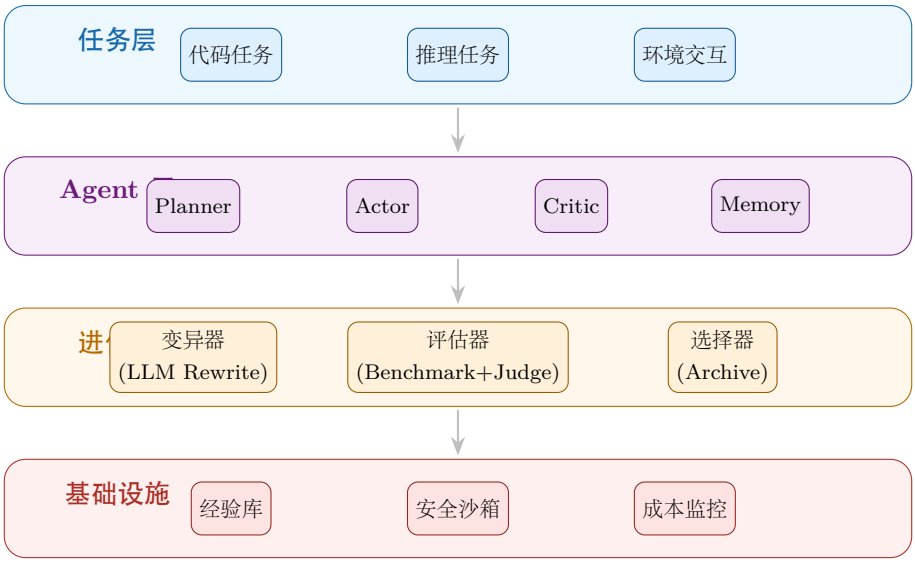

图 25: Agent 自进化系统四层参考架构。任务层定义目标和反馈信号；Agent 层执行规划-行动-批评-记忆循环；进化层实现生成-评估-选择闭环；基础设施层提供经验管理、安全保障和成本控制。

A.8 2026 年新增论文

arXiv ID	年份	主要贡献
2603.03290	2026	AriadneMem: 演化图记忆 + 熵感知门控
2603.07970	2026	Agent 架构新范式
2603.15255	2026	SAGE: 四 Agent 闭环自进化推理
2603.19461	2026	Hyperagents: 元认知自修改 Agent
2603.25928	2026	Agent 进化新方法
2603.28990	2026	自演化系统新架构
2604.01658	2026	Agent 系统改进
2604.02674	2026	记忆管理优化
2604.10923	2026	多 Agent 协作优化
2604.15034	2026	Autogenesis: 自进化 Agent 协议 (RSPL+SEPL)
2604.17091	2026	GenericAgent: Token 高效自进化 Agent
2604.18131	2026	奖励无关自进化探索
2605.04677	2026	进化策略新进展
2605.09315	2026	Capability Erosion: 自进化遗忘 +CPE 保留
2605.18930	2026	OEP: 自进化 Agent 经验投毒攻击
2605.19102	2026	安全治理新方法
2509.26354	2025/2026	Misevolution: 自进化 Agent 安全风险 (ICLR 2026)

2603.15255	2026	SAGE: 多 Agent 自进化推理框架
2603.19461	2026	Hyperagents: DGM 扩展，元认知自修改

总计: 107 篇论文（88+19 新增），覆盖 2022—2026 年 Agent Self-Evolution 领域主要工作。其中 2026 年论文 16 篇，安全/治理方向 3 篇（Misevolution, OEP, Capability Erosion），标志着该领域从性能优化向安全治理的重要转型。