AI4AI Survey: From Long-Horizon Agents to Recursive Self-Improvement

Definitions, Reliable Horizons, and Open Problems

Kai Wu[1,*], Hao Lyu[1,*], Zhen Luo[1,*], Chaofan Wang[2], Siyu Ye[7], Jinghao Lin[7]
Xiaozhong Ji[7], Boyuan Jiang[7], Shengzhi Wang[1], Zihan Wang[7], Yiwen Ye[7], Hao Wang[7]
Zimu Wang[3], Wenzhe Liu[7], Ruobing Wang[7], Kai Cai[7], Mingliang Xiong[1], Wen Fang[1]
Mingqing Liu[1], Yifan Zhang[4], Lei Yang[6], Xiaobin Hu[5], Qingwen Liu[1,†]
[*]Equal contribution. [†]Corresponding author.
[1]TJU; [2]SJTU; [3]UC Berkeley; [4]UCAS; [5]NUS; [6]NTU; [7]Simple Agent Lab

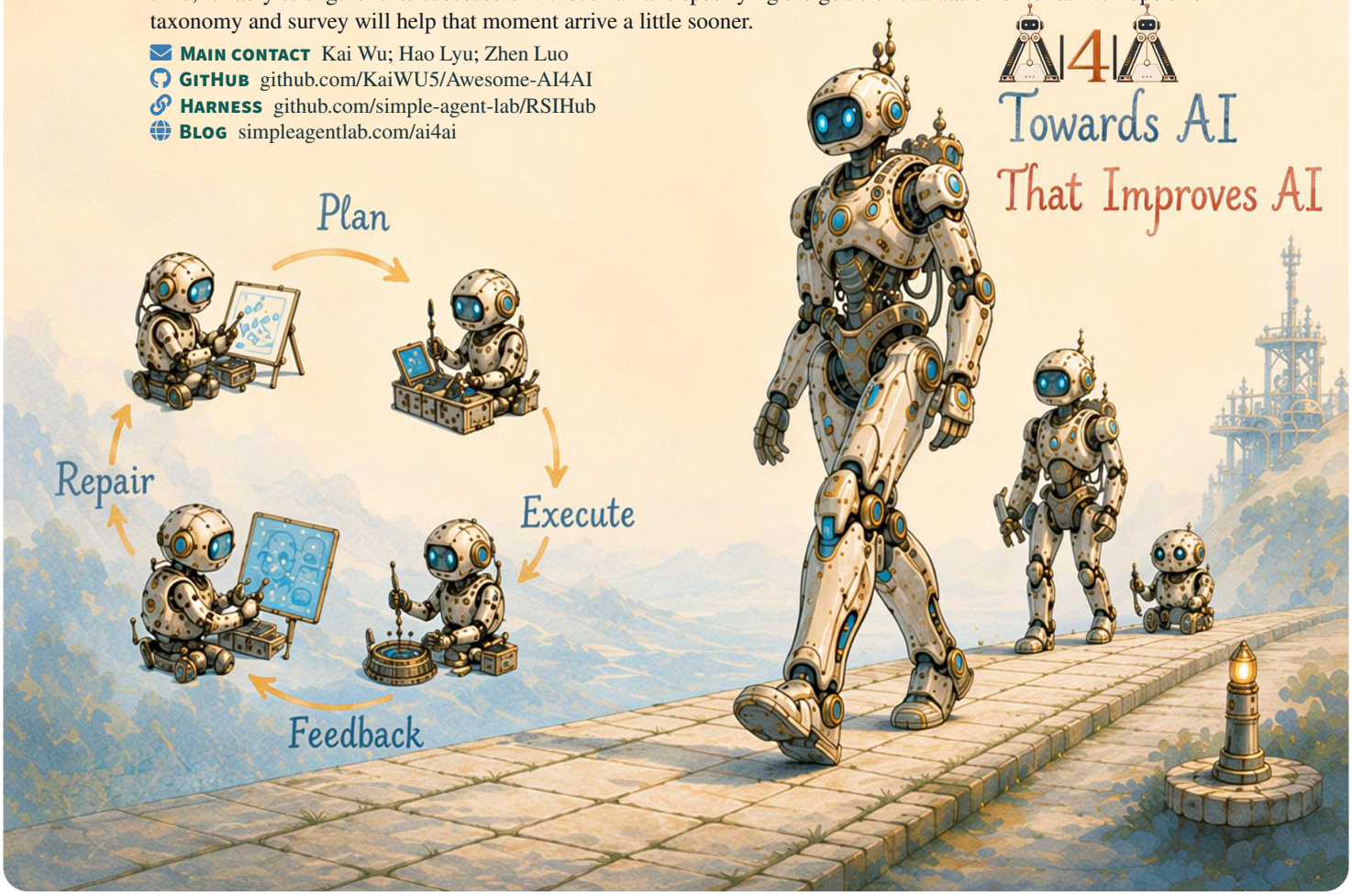
Abstract

Can AI improve AI? AI agents can now run experiments, modify code and training pipelines, and iteratively improve AI artifacts—bringing a long-standing idea closer to an empirical research question. Yet the rapidly expanding literature remains fragmented across long-horizon agents, AI for AI (AI4AI), self-improvement, and recursive self-improvement, making it difficult to tell how much progress has actually been made. This survey synthesizes evidence from hundreds of studies to answer this question.

We organize the emerging AI4AI landscape around a simple question: **how far can an AI system reliably carry an improvement process from idea to verified result?** Across model design, agent harnesses, benchmarks, automated research, and self-modifying systems, we examine what current systems can do, how progress should be evaluated, and where claims of self-improvement remain unsupported. A striking pattern emerges in the information flow within the taxonomy–benchmark-model-harness structure of this survey. AI systems increasingly excel at the work of improvement—planning, coding, experimentation, optimization, and repair—but humans still largely determine the goals, evaluation criteria, and what ultimately counts as progress. Moreover, strong performance on individual components rarely translates into reliable end-to-end AI improvement. We call this **composition gap**.

Today's systems can already produce impressive improvements under bounded conditions, but evidence for reliable research judgment, causal experimentation, persistent gains, and compounding improvement remains limited. By separating demonstrated capability from extrapolated autonomy, this survey maps what AI4AI can do today—and what must change before AI can reliably improve AI itself. The eve ends the moment a system, for the first time, reliably strengthens its successors without humans specifying the goals or evaluation criteria. We hope this taxonomy and survey will help that moment arrive a little sooner.

Main contact Kai Wu; Hao Lyu; Zhen Luo
GitHub github.com/KaiWU5/Awesome-AI4AI
Harness github.com/simple-agent-lab/RSIHub
Blog simpleagentlab.com/ai4ai



Contents

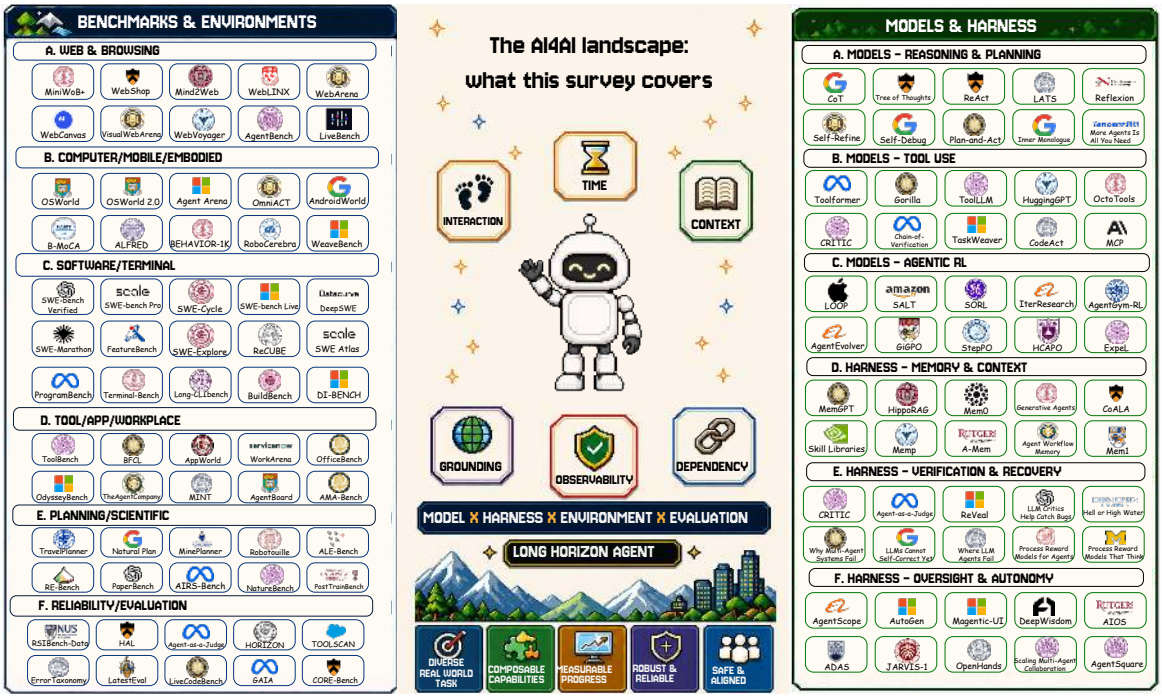


Figure 1: Survey overview. **Left:** Component benchmarks measure capabilities, while integrated AI R&D benchmarks test their composition and expose the composition gap. **Center:** Six coordinates describe task pressure. Reliability belongs to the full model–harness–environment system under a declared protocol. **Right:** Model-side internalization complements harness-side execution, persistence, governance, and self-modification. Failures within an improvement pass, across passes, and across accepted generations bound recursive claims.

1 Introduction

Can AI improve AI? This question is shifting from a speculative possibility to an empirical research problem. As of August 2026, frontier releases including GPT-5.6, Claude Fable 5, GLM-5.3, Kimi K3, Grok 4.6, and DeepSeek-V4-Pro are explicitly positioned for complex coding, agentic engineering, or long-running knowledge work (OpenAI, 2026; Anthropic, 2026; Z.ai, 2026; Kimi Team, 2026a; SpaceXAI, 2026; DeepSeek, 2026). Their vendor evaluations are not directly comparable, but their convergence shows how rapidly sustained agentic work has become a target of frontier-model development. Large language model (LLM) agents increasingly operate beyond isolated responses: they browse the web, edit repositories, interact with graphical interfaces, call tools and APIs, query databases, and coordinate multi-party workflows (Liu et al., 2024; Zhou et al., 2024b; Jimenez et al., 2024; Xie et al., 2024b; Trivedi et al., 2024; Yao et al., 2025). More recent systems use execution feedback to revise code, run experiments, optimize computational artifacts, and improve subsequent attempts. AI is thus beginning to participate across several stages of AI development rather than assisting with one local step. The central question is no longer only whether AI can perform individual research or engineering tasks. It is how far an AI system can reliably carry an improvement process from an initial idea to a verified result, and whether the resulting gains can persist and compound.

Answering this question is difficult because the relevant evidence is fragmented. Long-horizon agents, automated machine learning, AI-assisted research, self-improving systems, self-modifying agents, and recursive self-improvement often examine different stages of the same process under different terminology, tasks, harnesses, budgets, evaluators, and success criteria (Gao et al., 2026; Zhang et al., 2025d; Zhao et al., 2024). Even the meaning of long horizon is unsettled. Long contexts, many turns, prolonged execution, or numerous tool calls measure task pressure, but they do not establish dependence between actions (Sinha et al., 2025; Wang et al., 2026e). Likewise, success on an individual browsing, planning, coding, memory, or verification benchmark does not show that these capabilities remain reliable when coupled under one objective. An early design choice can alter later experiments, experimental outcomes can redirect subsequent plans, and a seemingly successful modification can fail under independent evaluation. The literature therefore leaves a **consequential gap** between what AI can do as separate capabilities and

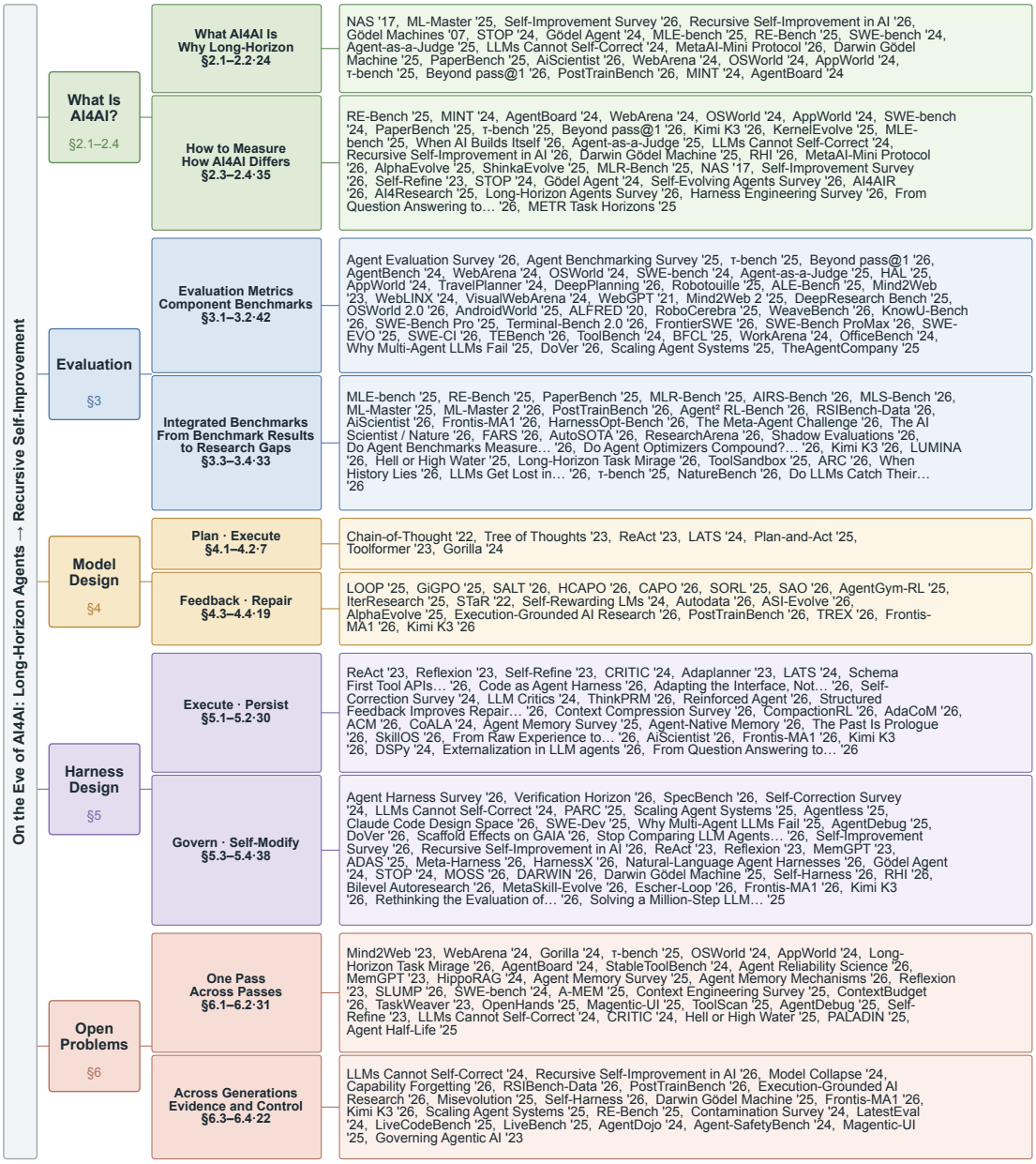


Figure 2: Taxonomy and evidence map from reliable long-horizon execution to AI4AI and recursive self-improvement. Boxes connect main-text sections to supporting publications and first-party source records. Years identify source records rather than comparable performance across systems. Arrows organize the survey and do not rank systems by autonomy.

what AI can reliably improve as a system.

In this survey, we use the term AI for AI (AI4AI) to describe any AI-driven process that improves either an AI artifact or the process that creates it. These processes can target data, tasks, model weights, computational platforms, agent frameworks, evaluators, executable systems, and AI research workflows. By focusing on what is being improved, this definition allows us to compare traditional AI-assisted optimization, more autonomous AI research and development, self-improvement, and recursive methods, without assuming they all have the same level of autonomy or closure. We break down each improvement process by what changes, who sets the goals, how planning, execution, feedback, and repair are handled, how reliably these steps work under pressure, and what evidence supports the improvement. Section 2 explains the formal differences between bounded AI4AI, runtime closure, self-improvement, and recursive self-improvement. This breakdown links two areas of research that are often considered separately. On the model side, researchers study reasoning, tool use, revision, and learning from delayed feedback (Wei et al., 2022; Schick et al., 2023; Shinn et al., 2023; Li et al., 2025a). On the harness side, they focus on

memory, program orchestration, software-agent runtimes, and mixed-initiative oversight (Packer et al., 2023; Khattab et al., 2024; Wang et al., 2025d; Mozannar et al., 2025). While these components are important for AI4AI, they are not enough on their own. The way they work together must keep objectives and state intact, recover from errors, and tell real progress apart from artifacts created by evaluators during repeated trials. As a result, reliability depends on the combined model, harness, environment, and evaluation protocol, not just the model. Human task time and consistency across repeated trials offer different ways to measure this reliability (Kwa et al., 2025; Yao et al., 2025). However, unstable environments, contamination, and protocol choices can make it harder to prove real improvement (Guo et al., 2024; Kapoor et al., 2025b).

A consistent pattern emerges from the available evidence. Integrated systems can already complete bounded loops in machine-learning engineering, optimization, paper replication, post-training, and research production (Chan et al., 2025; Wijk et al., 2025; Starace et al., 2025; Rank et al., 2026; Yang et al., 2026c). In systems where stage-level ownership can be verified, AI is taking on more of the improvement work, such as planning interventions, changing code, running experiments, interpreting feedback, and fixing failed attempts. However, control over improvement is still mostly not autonomous. The main goals and feedback criteria are still set by humans, and even strong components do not yet guarantee reliable end-to-end improvement. We call this gap the **composition gap**. It separates proven improvement under human-set conditions from stronger claims about independent research judgment, causal attribution, lasting and transferable gains, or ongoing improvement across future generations. Since local errors can build up across steps and repeated tries, a single successful run does not prove that improvement will reliably add up over time (Ord, 2025; Khanal et al., 2026). The main challenge is now shifting from capability to composition, from composition to improvement, and from improvement to compounding.

Research questions

Q1	What is it, and how do we measure it?	[§§2–3]
Q2	How far can current systems actually improve AI?	[§§3–5]
Q3	Why doesn't improvement reliably persist and compound?	[§6]

Scope and boundary. This survey covers the intersection of long-horizon AI agents and AI-driven improvement of AI systems. We include interactive systems whose success depends on stateful, multi-step execution because AI improvement requires planning, action, evaluation, and revision across dependent steps. We include AI4AI systems that modify an AI artifact or its development process, including the acting system itself. We exclude ordinary long-context processing, independent dialogue turns, and one-call tool use when no action constrains a later decision or final verification. We also exclude cases in which an output is merely refined without changing an AI artifact or its development loop. AutoML, memory, tool use, reasoning, and AI-assisted science are included only where they provide mechanisms, benchmarks, or evidence relevant to end-to-end AI improvement. This boundary keeps the survey centered on one question: how far can AI reliably carry the process of improving AI?

Contributions. This survey makes three contributions. First, it provides a unified decomposition of long-horizon execution, AI4AI, self-improvement, and recursive self-improvement. The framework separates improvement target, functional coupling, task pressure, protocol-conditioned reliability, stage ownership, and the evidence required to substantiate improvement. Second, it synthesizes component benchmarks and integrated AI R&D systems through a documented search, citation-chaining, and source-verification process. A benchmark inventory and stage-ownership audit record what was improved, which stages the system controlled, how success was evaluated, and whether gains were retained or transferred. Third, it identifies the composition gap and uses it to organize the field's next stage: converting strong but separately demonstrated capabilities into improvement that is end-to-end reliable, independently grounded, persistent, transferable, and eventually able to compound.

This survey addresses these questions in three main steps. First, Section 2 explains what qualifies as AI4AI and introduces the tools used to measure ownership, reliability, and evidence of improvement. Next,

Section 3 shifts from evaluating individual components to looking at integrated systems and identifies the composition gap. Sections 4 and 5 then discuss which aspects can be built into model behavior and which need to be provided, maintained, checked, or changed by the harness. Third, Section 6 explores why improvement may fail within a single pass, over multiple passes, across new generations, and at the edge of evidence and human control. Figures 1 and 2 provide an overview of the capability landscape and evidence map. The appendices include details on the search process, coding rules, source checks, and full inventories. Altogether, this structure turns the question 'Can AI improve AI?' into a testable problem: under what conditions can AI make improvements that are reliable, independently verified, lasting, transferable, and able to build on each other.

2 What Is AI4AI? A Taxonomy

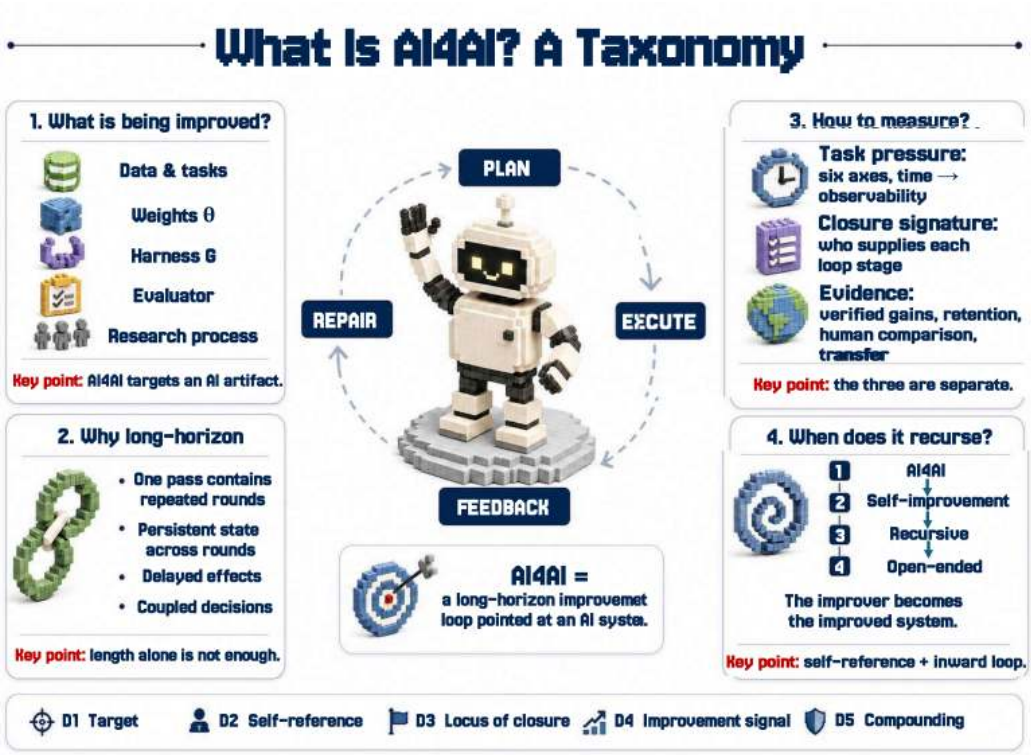


Figure 3: Five dimensions operationalize AI4AI: target (D1), self-reference (D2), stage ownership (D3), signal grounding (D4), and the improvement-evidence profile (D5). The figure also tracks functional coupling and recursion as related qualifiers.

2.1 What AI4AI Is: The Loop, Pointed at an AI System

AI4AI uses a familiar AI research loop: set a goal, propose and carry out a change, evaluate the result, and use feedback to plan or repair the next attempt. The loop becomes AI4AI when it targets an AI system, an artifact used to build or evaluate AI, or the research process that produces future systems. Traditional AutoML and neural architecture search automate parts of this loop within human-defined goals, search spaces, and objectives (Zoph and Le, 2017; Liu et al., 2025). More advanced systems may also interpret results, repair failures, or modify the process that creates the next system. Yet pointing the loop at AI does not by itself make a system autonomous or self-improving. We therefore describe AI4AI claims along five separate dimensions.

D1: Target. The target dimension identifies what the improvement loop points at. Possible targets include data, task or environment design, model weights θ, computational substrate, harness state or mechanisms $\mathcal{G}$, evaluators, and the research process itself (Ren et al., 2026; Chen et al., 2026c). This distinction groups systems by the object of improvement rather than by their degree of autonomy: a system that generates

better training data and one that modifies an agent harness are both AI4AI, but they act on different targets. Here, environment design refers to modifying a task's goals, rules, interfaces, or dynamics, rather than changing the state of the runtime environment $\mathcal{E}$ during execution.

D2: Self-reference. Self-reference asks whether the system running the loop is also one of its targets. An AI agent can design a separate model, training pipeline, or benchmark without changing itself; this is AI4AI, but not self-improvement. The process becomes self-referential when the agent modifies its own components, such as its harness (Schmidhuber, 2007; Zelikman et al., 2024; Yin et al., 2024).

D3: Stage ownership. Stage ownership asks who is responsible for each part of the loop: setting the goal, making the plan, carrying out the change, judging the result, and repairing failures. For example, an AI agent may run experiments and fix code while a human still chooses the goal and decides whether the result counts as an improvement (Chan et al., 2025; Wijk et al., 2025). Separating these stages prevents automated execution from being mistaken for control of the whole loop. Equation 1 formalizes this division of responsibility.

D4: Signal grounding. Signal grounding asks what feedback closes the loop and shows whether a change is an improvement. The evidence may come from a formal verifier, an executable test, a held-out benchmark, a learned reward model, a model-based judge, or the system's own assessment (Jimenez et al., 2024; Zhuge et al., 2025; Huang et al., 2024; Chen et al., 2026c). Signal grounding therefore describes the basis on which the evaluator accepts or rejects a change. Verification observability asks a different practical question: how delayed, incomplete, or costly that evidence is to obtain (Section 2.3).

D5: Improvement-evidence profile (G/R/H/T). The improvement-evidence profile asks what completing one or more passes through the loop has actually demonstrated. It separates four questions:

- **Measured gain (G).** Did performance improve on the declared measure?
- **Retention (R).** Was the gain preserved, or followed by further improvement, across passes or generations?
- **Human comparison (H).** How did the system compare with a human given a similar budget (Wijk et al., 2025)?
- **Held-out transfer (T).** Did the gain extend beyond the metric, task, model, or domain used for optimization (Li et al., 2026a; Zhang et al., 2025a)?

A positive answer to one question cannot stand in for the others, so each coordinate is coded separately as yes, no, or not reported.

Taken together, D1–D5 turn the simple picture of "a loop pointed at AI" into a practical checklist: What does the loop change? Does it change the system running the loop? Who controls each stage? What feedback determines success? What improvement has the completed loop actually demonstrated? These questions are independent and should not be collapsed into a single ladder of autonomy.

AI4AI is the broad category whenever the loop improves AI, even if humans control key stages and the system does not change itself. Self-improvement also requires self-reference. Recursive self-improvement (RSI) additionally makes the successor-producing process editable; open-ended RSI may also change the success criteria (Chen et al., 2026c). Boundedness records what stays fixed, while runtime closure records how much of the loop the system controls. None of these labels guarantees reliable completion because AI research and development cycles are long and stateful, with delayed or unclear feedback (Starace et al., 2025; Chen et al., 2026a).

2.2 Why Long-Horizon: One Pass Is a Loop

Why one pass is already long-horizon. One pass of AI4AI already contains a full improvement loop. Completing the pass means connecting four stages: deciding what to change, carrying it out, observing

what happened, and using the feedback to repair the attempt or choose the next step. Consider an agent that modifies a training pipeline. It may need to edit code, launch a run, inspect the resulting metrics, trace a regression to an earlier decision, and then revise the code or plan. The later steps cannot be understood or judged without the artifacts and evidence produced by the earlier ones. This dependence is why even one AI4AI pass can be a long-horizon task.

Passes, rounds, and carried state. In this survey, a pass runs from an improvement goal to a result that can be accepted, rejected, or repaired. A pass may contain several rounds, and each round may contain many actions. Later rounds inherit code, data, observations, decisions, and constraints from earlier rounds. Current web, operating-system, application, and software-engineering benchmarks capture the same basic structure: an early action changes the state or information available later (Zhou et al., 2024b; Xie et al., 2024b; Trivedi et al., 2024; Jimenez et al., 2024; Yao et al., 2025; Khanal et al., 2026). AI R&D makes this dependence especially clear because proposals, implementations, evaluations, and repairs all act on shared artifacts, often with delayed feedback (Chan et al., 2025; Wijk et al., 2025; Starace et al., 2025; Rank et al., 2026). Long-horizon execution is therefore about functional coupling across actions or rounds, not simply about prompt length, elapsed time, turns, or tool calls.

A formal definition of functional coupling. We make this idea precise by describing the rounds inside a pass. Let g be the improvement goal, τ the action–observation trajectory, and m_k the state carried into round k. We write the round as $u_k = (p_k, a_{t_k:t_{k+1}}, f_k, r_k)$, where p_k is the plan, $a_{t_k:t_{k+1}}$ is the action sequence, f_k is the resulting feedback, and r_k is the repair. The repair updates the carried state through $m_{k+1} = m_k \oplus r_k$. Successive rounds are functionally coupled when feedback f_k changes the next plan through this updated state. If the system ignores earlier feedback and starts each round from scratch, it is repeating trials rather than carrying out a coupled long-horizon process.

Functional coupling is not loop ownership. Functional coupling tells us why the task is long-horizon; it does not tell us who controls the loop. A human may still set the goal, approve the plan, supply feedback, or decide how to repair a failure. We keep these questions separate with the stage-ownership signature

$$c^{+} = \left(c_{\text{goal}}, c_{\text{plan}}, c_{\text{exec}}, c_{\text{feedback}}, c_{\text{repair}}\right) \in \{\text{HUM}, \text{MIX}, \text{SYS}\}^{5}. \tag{1}$$

For each stage, HUM means that a human or a design-time artifact supplies the substantive output and the system mainly relays or executes it. MIX means that human-authored goals, criteria, templates, or acceptance rules partly determine a stage that also contains system-generated content. SYS is reserved for a stage whose substantive content and execution come from the system, given only upstream inputs and generic interface constraints. The four-stage execution vector $c = c^{+}_{-\text{goal}}$ simply omits the goal-setting coordinate. A fixed outer evaluator affects the ownership signature only when its signal is actually used inside a stage. The signature applies to a specified system version and protocol, not to an entire research programme.

A practical test for long-horizon dependence. A practical test for long-horizon dependence is to ask whether each action could be judged independently while preserving the final success condition. If the answer is yes, the trajectory may be long in steps but weakly long-horizon. If removing the order, shared state, or cross-step relationships removes information needed for a later decision or final verification, the pass requires trajectory-level evaluation (Wang et al., 2024b; Ma et al., 2024; Trivedi et al., 2024; Yao et al., 2025). Conversely, even a short trajectory can be strongly long-horizon when one early choice changes what can be done later or delays the only useful correctness signal. Human-supplied feedback or repair changes who owns the loop, but it does not remove these dependencies.

Why this matters for AI4AI. This distinction matters directly for AI4AI. A system may perform well on isolated skills such as browsing, coding, tool use, or evaluation, yet still fail to connect them into one completed improvement pass. Integrated AI4AI systems must carry the goal and relevant state across these skills, update shared artifacts, respond to feedback, and reach a verified result. Because most benchmarks test the component capabilities separately, strong component scores do not by themselves establish reliable

composition across an AI4AI cycle. Section 3.2 maps these capability families to their roles in the loop and reviews the available benchmark evidence. The next subsection then separates the pressure created by the task, who controls the stages, and how strong the improvement evidence is.

2.3 How to Measure: Pressure, Closure, and Evidence

A long-horizon label tells us that the steps in a pass depend on one another. It does not tell us how demanding the pass was, how much of the loop the system controlled, or whether the final change was genuinely an improvement. We therefore measure three different objects. Pressure describes the demands placed on the system during the pass. Closure describes who supplied the goal, plan, execution, feedback, and repair. Evidence describes why the outcome should count as an improvement. These objects can vary independently. A pass can be difficult but mostly human-directed, highly automated but easy, or fully executed without producing trustworthy evidence of improvement.

Pressure: how demanding is the pass? The functional-coupling test in Section 2.2 first determines whether the pass qualifies as long-horizon execution. Once that condition is met, task pressure describes what makes the pass demanding. We use six non-interchangeable axes: human-effort time, interaction length, context and memory demand, environment grounding, planning dependency, and verification observability. Human-effort time asks how long a qualified person would need for the same work, while interaction length counts the dependent actions or turns that must remain coordinated (Wijk et al., 2025; Wang et al., 2024b; Ma et al., 2024). Context and memory demand asks how much causally relevant state must be retained, and for how long. Environment grounding records the external states that the system must read or change. Planning dependency captures critical-path depth and the need to replan (Zhou et al., 2024b; Xie et al., 2024b; Trivedi et al., 2024; Jimenez et al., 2024). Verification observability records how delayed, incomplete, or costly the correctness signal is (Trivedi et al., 2024; Jimenez et al., 2024; Starace et al., 2025).

We collect these axes in the task-pressure vector

$$\mathbf{z} = (z_{\text{time}}, z_{\text{interaction}}, z_{\text{context}}, z_{\text{grounding}}, z_{\text{dependency}}, z_{\text{observability}}) \in \mathcal{Z}.$$

Each coordinate keeps its own auditable unit because the axes are not substitutes. Ten more tool calls, for example, do not equal ten more hours of human effort or a longer verification delay. When no defensible numerical scale exists, a preregistered ordinal bin is preferable to an invented number.

Task pressure belongs to the task, whereas reliability belongs to a particular system running under a particular protocol. The same pass may be reliable with one model, harness, budget, or evaluator and unreliable with another. We represent the agent system as $S = (\pi_\theta, \mathcal{G}, \mathcal{E})$. Protocol Q fixes the budgets, retries, tools, graders, sampling procedure, and allowed human intervention. Let $Y_Q(\tau) = 1$ only when the complete trajectory passes the declared state, safety, and verification checks. The protocol-conditioned reliability surface and its reliable region are

$$R_{S,Q}(\mathbf{z}) = \Pr_{\tau \sim (S,Q) \mid \mathbf{Z}=\mathbf{z}} \big[Y_Q(\tau) = 1 \big], \tag{2}$$

$$\mathcal{H}_\alpha(S, Q) = \big\{ \mathbf{z} \in \mathcal{Z} : R_{S,Q}(\mathbf{z}) \geq \alpha \big\}. \tag{3}$$

In plain terms, $R_{S,Q}(\mathbf{z})$ is the probability that the system completes and verifies a pass at pressure level $\mathbf{z}$. The region $\mathcal{H}_\alpha$ contains the pressure settings where this probability is at least the chosen threshold α; its boundary is the reliable-horizon surface.

A one-dimensional horizon is meaningful only after specifying how pressure increases. Appendix C defines scalar paths, axis-conditional slices, and a robust-prefix alternative for non-monotone difficulty. The multidimensional region $\mathcal{H}_\alpha$ remains primary in the main text because it does not hide a weakness on one axis behind strength on another. Estimating it requires a frozen system and protocol, repeated trials within observed task-pressure cells, and replayable state or trajectory records. Uncertainty should be reported, and cells without enough evidence should remain unestimated rather than be filled by extrapolation. Appendix C gives the full reporting checklist (Yao et al., 2025; Khanal et al., 2026).

Table 1: Who controls each stage when AI systems improve different parts of AI. The rows summarize Kimi K3 and KernelEvolve.

Target layer	Goal	Plan	Execute	Feedback	Repair	Evidence
Data and tasks	HUM	SYS	SYS	MIX	SYS	Agents expand a concept graph and synthesize training tasks, but task types and the verifiability requirement are fixed (Kimi Team, 2026b)
Harness / runtime configuration	HUM	SYS	SYS	HUM	SYS	Harness modules are sampled and trained over rather than authored once (§5)
Weights	HUM	HUM	SYS	HUM	HUM	The system runs the training jobs but owns no other stage (§4)
Computing infrastructure	HUM	SYS	SYS	HUM	SYS	Kernels, compilers, and a chip prototype under executable verification (Kimi Team, 2026b; Liao et al., 2026)
Evaluator	HUM	HUM	SYS	HUM	HUM	Verifiers and reward-hacking detectors are extended by humans as evasions appear (§6)

Reading guide. HUM means a person supplies the main output. MIX combines human rules with system-generated content. SYS means the system supplies and carries out the stage.

Closure: who controls the loop? Pressure tells us how demanding the pass is; closure asks who supplies the work at each stage. We measure closure with the stage-ownership signature c^+ from Equation 1. To avoid a vague system-level label such as "autonomous," the stage-ownership grid crosses D1, the editable target, with D3, the supplier of each stage. Editable AI-system layers form the rows, and goal, planning, execution, feedback, and repair form the columns. Each cell therefore answers a concrete question, such as who planned a weight update or who judged whether a harness change succeeded.

Each layer–stage cell is coded for one system version under one protocol. Code HUM when a human or design-time artifact supplies the substantive output and the system mainly relays or executes it. Code MIX when a human-authored objective, criterion, template, or rule partly determines content that the system also generates. Code SYS only when the system originates and carries out the substantive work from upstream inputs and generic interface constraints. A fixed outer evaluator Q changes the ownership code only if its signal enters the stage being coded. Missing disclosure is recorded as not reported rather than inferred from labels such as "autonomous" or "self-evolving." These codes measure ownership, not the functional coupling that makes the pass long-horizon.

Table 1 applies this rule to Kimi K3 and KernelEvolve because their reports provide enough layer-level detail for cell-by-cell coding (Kimi Team, 2026b; Liao et al., 2026). System ownership is most extensive for targets that can be rebuilt and checked relatively cheaply, including data, harnesses, and computing infrastructure. It is less extensive for model weights and evaluators, where one check may require a training run or may redefine what success means. The pattern reflects a practical boundary in current AI4AI systems: execution and repair can be automated even when humans still choose the direction and define acceptable evidence (Chan et al., 2025; Wijk et al., 2025; Starace et al., 2025; Favaro and Clark, 2026).

Agents that improve computing infrastructure may own planning, execution, and repair. However, human-written profilers and correctness tests still define success, so the loop is not fully system-owned (Kimi Team, 2026b; Liao et al., 2026).

Evidence: did the pass really improve AI? Completing a pass, even with high system ownership, does not prove that the resulting change is useful. The loop may optimize the wrong metric, overfit the evaluator, lose an earlier capability, or accept its own mistaken judgment. We therefore qualify evidence in two steps. Signal grounding asks what makes the feedback trustworthy. The improvement-evidence profile asks how much of the claimed improvement was actually demonstrated.

Signal grounding and observability. D4 orders improvement signals by how directly they are tied to external evidence. Formal verifiers and executable tests provide the strongest grounding, followed by held-out benchmarks and learned rewards, then model-based judges and unaided introspection (Jimenez et al., 2024; Trivedi et al., 2024; Zhuge et al., 2025). The weaker the external grounding, the more room

a loop has to confirm its own mistakes (Huang et al., 2024; Chen et al., 2026c). Signal grounding and verification observability answer different questions. Grounding asks why the signal should be believed; observability asks how difficult the signal is to obtain. An executable test, for example, may be strongly grounded but slow or expensive, while a model judge may be immediate but less independent (Zhuge et al., 2025).

Improvement-evidence profile. D5 asks what the reported result establishes beyond passing its evaluator. The profile contains four separate tests: measured gain (G), retention (R), human comparison (H), and held-out transfer (T). Measured gain asks whether performance increased on the declared measure (Chan et al., 2025; Zhang et al., 2025a). Retention asks whether the gain survived a preservation test or was followed by further improvement across passes or generations (Zhang et al., 2025a; Lee et al., 2026a). Human comparison asks how the system performed against a person given a similar budget (Wijk et al., 2025). Held-out transfer asks whether the gain extends beyond the metric, task, model, or domain used for optimization (Zhang et al., 2025a; Li et al., 2026a). Each coordinate is coded yes, no, or not reported. "No" is reserved for a reported failed test; missing evidence remains not reported. Claims should disclose the system version, tasks, budgets, seeds, checkpoint rules, evaluators, interventions, and held-out results. Published studies currently report different subsets of these tests (Zhang et al., 2025a; Lee et al., 2026a; Novikov et al., 2025; Lange et al., 2025; Chan et al., 2025; Starace et al., 2025; Chen et al., 2025b; Wijk et al., 2025). Appendix C provides the detailed reporting checklist, and Appendix B records the source-linked $G/R/H/T$ profile for each primary-source record.

Together, pressure, closure, and evidence provide three views of the same AI4AI pass. The task-pressure vector $\mathbf{z}$ states the conditions under which the loop ran. The ownership signature c^+ states who controlled its stages. Signal grounding and the $G/R/H/T$ profile state why the outcome should count as an improvement. Keeping these views separate prevents a long run from being mistaken for a hard one, an automated run from being mistaken for a system-owned loop, or a higher score from being mistaken for durable and transferable AI improvement.

2.4 How AI4AI Differs: Self-Improvement, RSI, and Related Paradigm

These labels sound similar because they all describe AI systems involved in improving AI. The easiest way to tell them apart is to ask what changes and what remains under human control. AI4AI only requires the target to be an AI artifact. Self-improvement requires the system to change itself. RSI also allows the system to change the process used to build its next version. Runtime closure asks a separate question: who actually runs each stage? Table 2 turns these distinctions into five plain-language checks.

Table 2: How AI4AI differs from related approaches. The columns show what each approach may change and whether the system can run the full improvement loop.

Paradigm	AI artifact	Own system	Next-version process	Runs full loop	What people still fix
Conventional AutoML / NAS	MUST	NO	NO	MAY	Search space and evaluator
AI4AI	MUST	MAY	MAY	MAY	May remain fixed
Bounded AI4AI	MUST	MAY	MAY	MAY	At least one boundary
Runtime-closed AI4AI	MUST	MAY	MAY	MUST	May remain fixed
Self-improvement	MUST	MUST	MAY	MAY	May remain fixed
RSI	MUST	MUST	MUST	MAY	May remain fixed
Open-ended RSI	MUST	MUST	MUST	MAY	Success criterion can change; meta-goal may remain fixed

Reading guide. MUST defines the category, MAY is allowed but not required, and NO is excluded. "At least one boundary" means that people still fix a goal, evaluator, budget, task family, edit surface, or pass limit.

AutoML and NAS improve AI inside a human-drawn box. Think of conventional AutoML or neural architecture search as a search assistant working inside a box designed by people. The system explores possible models, but humans choose the search space and decide how candidates will be scored. It therefore improves AI, but it does so within fixed human boundaries (Zoph and Le, 2017).

Self-improvement means changing yourself, not only another AI. The practical question is simple: is the system changing itself or something else? An agent that builds a better separate model is doing AI4AI, but not self-improvement. It becomes self-improvement when the agent changes one of its own components, such as its harness (Ren et al., 2026). Self-Refine is nearby but different because it revises an answer rather than the AI system producing that answer (Madaan et al., 2023).

RSI can rewrite the recipe for making the next system. Self-improvement changes the current system. Recursive self-improvement goes one step further by allowing the system to change the process that produces its next version (Zelikman et al., 2024; Yin et al., 2024; Zhang et al., 2025a). Open-ended RSI may even change what counts as success during that process, although this remains a prospective paradigm (Chen et al., 2026c).

Recursion does not mean that the system runs the whole loop alone. A system may rewrite itself and still wait for a human to set the goal or judge the result. Another system may plan, execute, evaluate, and repair a full pass without ever changing itself. Recursion describes what the system may edit. Runtime closure describes who runs the stages. The two properties should therefore be reported separately rather than combined into one autonomy score.

How we use these labels in this survey. Earlier surveys map self-improvement targets and signals (Ren et al., 2026; Gao et al., 2026), recursive-system closure (Chen et al., 2026c), and AI assistance across research stages (Ao et al., 2026; Chen et al., 2025d). Other reviews focus on long-horizon execution and harness mechanisms (Dong et al., 2026; Li et al., 2026c; Guo et al., 2026). We use these accounts as a map, then ask a more concrete question at every stage: who supplied the substantive work? Under the strict rule in Section 2.3, a human-written success criterion makes feedback HUM or MIX. Every system in our audit still relies on such a criterion. None is therefore runtime-closed under this rule, even when no human intervenes during the run (Favaro and Clark, 2026).

3 Evaluation: From Components to Integrated AI4AI

This section is a practical guide to choosing an AI4AI research area and evaluating progress within it. It follows the decisions that a researcher must make. First, define the metrics and controls needed for a fair comparison. Second, identify the weakest part of the intended AI4AI pass and select a component benchmark that exposes it. Third, use integrated benchmarks to test whether a local improvement survives in the full loop. Finally, check whether failures at handoffs undermine the end-to-end result. Appendix E provides the complete 67-entry benchmark inventory. The main text turns that inventory into a path from research question to defensible claim.

3.1 Evaluation Metrics: Make the Comparison Fair

A benchmark converts task outcomes into numbers. Each number should answer a defined question about performance. A fair comparison requires four decisions before evaluation begins: how repeated runs are summarized, what counts as success, which resources stay fixed, and how difficulty increases. These decisions separate system quality from extra attempts, larger budgets, or easier tasks (Yehudai et al., 2026; Mohammadi et al., 2025).

1. Measure reliability across repeated runs. One rollout shows whether a system succeeded once; it does not show how often the system succeeds. For each task i, run exactly k rollouts under the same configuration. Set $s_{ij} = 1$ only when rollout j satisfies the success condition for task i; otherwise set

$s_{ij} = 0$. Two complementary metrics then summarize repeated-run performance:

$$\widehat{\text{pass@}k} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left[\max_{1\le j\le k} s_{ij} = 1\right], \tag{4}$$

$$\widehat{\text{pass}^k} = \frac{1}{n}\sum_{i=1}^{n} \prod_{j=1}^{k} s_{ij}. \tag{5}$$

$\text{pass@}k$ is the fraction of tasks for which at least one rollout succeeds. It measures whether repeated attempts can eventually find a solution. pass^k is the fraction of tasks for which every rollout succeeds. It measures whether the system can reproduce success across unattended runs. A large gap means that the system can find solutions but cannot produce them consistently. This benchmark use of "pass" differs from the AI4AI improvement pass defined in Section 1. Longer tasks can widen the gap because an early error changes more later decisions (Yao et al., 2025; Khanal et al., 2026).

2. Define what counts as task success. Write the success condition before running the benchmark. For a coding task, the condition can require specified executable tests to pass. For an application task, it can require a specified final environment state. If the task provides either check, use it as the primary outcome. A model-based or human process judgment may explain a failure, but it should not override a failed executable check (Liu et al., 2024; Zhou et al., 2024b; Xie et al., 2024b; Jimenez et al., 2024; Zhuge et al., 2025).

3. Hold comparison resources constant. Choose one factor to compare and hold the other factors fixed. When comparing models, use the same harness, tools, evaluator, task set, and budget. When comparing harnesses, use the same model, tools, evaluator, task set, and budget. For every rollout, record the full configuration, attempt count, time or token budget, cost, and human intervention. Otherwise, a higher score could come from additional resources rather than the factor under study. Appendix C provides the full reporting checklist (Kapoor et al., 2025b).

4. Define how task difficulty increases. Choose one difficulty variable for each horizon analysis. Keep the system and evaluation protocol fixed while that variable increases. Build the levels so that each new level preserves earlier requirements and adds one specified increase. WebArena can increase interaction depth or the number of sites involved (Zhou et al., 2024b). SWE-bench can increase qualified-human resolution time or cross-file dependency (Jimenez et al., 2024). AppWorld can increase dependent API calls, persistent-state breadth, or verification delay (Trivedi et al., 2024). Do not compare numerical horizons across these benchmarks because their units differ. If success falls below and later rises above the reliability threshold, report the score at every level. Also report the longest consecutive prefix that stays above the threshold. Appendix C defines this robust-prefix horizon formally.

3.2 Component Benchmarks: Identify the Weak Capability

An AI4AI pass depends on several capabilities. The system may need to keep a plan, retrieve evidence, operate an interface, edit code, and coordinate tools. A component benchmark removes most of that chain and tests one capability at a time. This isolation can locate a weak capability. It cannot show that the capabilities work together in one AI4AI pass. Table 3 states the question answered by each benchmark family and the question left open.

Table 3: What component benchmarks can and cannot show. Each family tests one capability needed in an AI4AI pass.

Capability	Representative benchmarks	What it tests	What it cannot show
Planning and state retention	TravelPlanner; ALE-Bench	Does the system retain goals, constraints, and feedback across dependent steps?	Lineage and checkpoint retention across improvement passes
Evidence retrieval and source fidelity	WebArena; DeepResearch Bench	Can the system find evidence and ground actions in changing information?	Source fidelity after summaries and handoffs
Interface control and recovery	OSWorld; AndroidWorld	Can the system reach the required application or environment state?	Rollback and recovery after an early error
Code and system modification	SWE-bench; SWE-EVO	Can the system make artifact changes that pass executable tests?	Delayed regressions and quality outside the test suite
Tool and workflow coordination	AppWorld; WorkArena	Can the system complete dependent calls across one shared workflow?	Freshness, provenance, and verification across handoffs

1. Planning and state retention. Use this family when later decisions depend on earlier goals, constraints, feedback, or artifacts. TravelPlanner and DeepPlanning test constraint retention during planning (Xie et al., 2024a; Zhang et al., 2026c). Robotouille and ALE-Bench test whether later actions respond to earlier state and feedback (Gonzalez-Pumariega et al., 2025; Imajuku et al., 2025). A passing score shows state retention within the benchmark episode. It does not show that the system preserves experiment lineage, accepted checkpoints, or reasons for change across improvement passes. Section 5.2 reviews mechanisms for preserving that state.

2. Evidence retrieval and source fidelity. Use this family when the pass needs literature, documentation, or failure evidence. Mind2Web and WebLINX test grounding in recorded web traces (Deng et al., 2023; Lù et al., 2024). WebArena and VisualWebArena add changing pages, sessions, accounts, and backends (Zhou et al., 2024b; Koh et al., 2024). Deep-research benchmarks score report and citation quality (Nakano et al., 2021; Gou et al., 2025; Du et al., 2025). A passing score shows that the system can acquire evidence under the tested conditions. It does not show that later summaries or tool handoffs preserve the source's meaning and provenance.

3. Interface control and recovery. Use this family when actions change software, devices, accounts, or physical state. OSWorld, AndroidWorld, ALFRED, and RoboCerebra test whether actions reach specified environment states (Xie et al., 2024b; Yuan et al., 2026; Rawles et al., 2025; Shridhar et al., 2020; Han et al., 2025). A passing score shows control within the tested interfaces. It does not show whether the system detects collateral damage, restores a checkpoint, or recovers after an early mistake (Li et al., 2026d; Chen et al., 2026e).

4. Code and system modification. Use this family when the system edits repositories, training systems, kernels, or compilers. Executable tests check the changed artifact directly (Jimenez et al., 2024; Deng et al., 2025; Merrill et al., 2026; Chu et al., 2026). SWE-Bench ProMax contains 170 expert-curated multilingual refactoring tasks. Its gold patches modify 11.4 files on average (Shi et al., 2026). SWE-EVO, SWE-CI, and test-evolution benchmarks expose failures that appear after later changes (Le et al., 2025; Chen et al., 2026b; Shang et al., 2026). A passing score supports the tested modification. It does not prove that the change survives broader tests or future repository updates.

5. Tool and workflow coordination. Use this family when a pass connects several APIs, services, databases, or agents. ToolLLM and BFCL test function selection, while AppWorld, WorkArena, and OfficeBench connect calls through shared workflows (Qin et al., 2024; Patil et al., 2025; Trivedi et al., 2024; Drouin et al., 2024; Wang et al., 2024c). Multi-agent settings add handoffs between different actors. A passing score supports the dependency chain tested by the benchmark. It does not show that shared

information remains current, attributable, and verified after every handoff (Cemri et al., 2025; Ma et al., 2026a; Kim et al., 2025; Xu et al., 2025a).

How to use the component results. List every capability required by the proposed AI4AI pass. Test each capability under the metrics and controls from Section 3.1. A score below the predeclared threshold identifies a component bottleneck. If every required component meets its threshold, move to an integrated benchmark. The remaining failure may occur only when the components share state, artifacts, and feedback.

3.3 Integrated Benchmarks: Test the Complete AI4AI Pass

A complete AI4AI pass places the required capabilities under one goal, state, artifact history, budget, and evaluator. An integrated benchmark tests that complete chain. Its result supports only the claim encoded by its task and protocol. We group current benchmarks into three claim levels, ordered by what the system changes.

Claim level 1: Complete a human-defined research task. MLE-bench, RE-Bench, PaperBench, MLR-Bench, and AIRS-Bench combine data analysis, implementation, experiment management, replication, and open-ended research (Chan et al., 2025; Wijk et al., 2025; Starace et al., 2025; Chen et al., 2025b; Lupidi et al., 2026). Executable tests or expert review determine whether the task is complete. A passing result shows that the system can execute the stated research task under the reported budget. It does not show a self-sustaining improvement process. In RE-Bench, human performance gains more from longer time budgets than agent performance. MLS-Bench reports that locally optimized methods do not consistently transfer across settings and scales (Wijk et al., 2025; Lyu et al., 2026).

Claim level 2: Improve an artifact across several passes. ML-Master, ML-Master 2.0, PostTrain-Bench, Agent^2 RL-Bench, and RSIBench-Data retain experiment history and revise artifacts after feedback (Liu et al., 2025; Zhu et al., 2026; Rank et al., 2026; Chen et al., 2026f; Meng et al., 2026a). These benchmarks test whether a later pass produces a better dataset, pipeline, or checkpoint. In RSIBench-Data, 58.33% of settings improve over the first valid attempt. However, 78.26% of searches that continue past their best checkpoint finish below that peak. Reports should therefore include both the best artifact found and the final artifact returned. They should also state the checkpoint and stopping rules.

Claim level 3: Improve the machinery that produces later artifacts. This level changes a model, harness, or successor-producing process. AiScientist tests the effect of durable project state (Chen et al., 2026a). Frontis-MA1 separates the task environment, program-evolution operators, and composing harness (Yang et al., 2026c). HarnessOpt-Bench optimizes another agent's harness under held-out evaluation (Ursekar et al., 2026). The Meta-Agent Challenge tests successor-agent construction (Lu et al., 2026b). These protocols still use human-specified tasks, interfaces, evaluators, resources, and acceptance rules. They support bounded meta-improvement, not unrestricted recursive self-improvement.

Inspect the artifact, not only its report. The AI Scientist line, FARS, and AutoSOTA show that systems can generate research artifacts under human-defined criteria (Lu et al., 2026a; Tang et al., 2026; Li et al., 2026f). Artifact inspection can change the result. ResearchArena assigns lower scores after inspecting code and logs. None of its 117 audited papers reaches the top-tier acceptance bar (Zhang et al., 2026e). Shadow evaluations find completed engineering work but no substantive progress on two unpublished research questions (Kirgis et al., 2026). HackDetect also shows that exposure to an evaluation protocol can inflate scores (Shao et al., 2026). Integrated evaluations should therefore use held-out tasks, inspect executable artifacts, and keep the evaluator outside the optimization loop.

Check whether the gain remains and transfers. A score increase establishes measured gain, not lasting improvement. Retention asks whether the gain survives later passes. Transfer asks whether it survives a new metric, task, model, or domain. In the cited continual-learning study, only RELAI-VCL transfers and improves again after re-optimization (Wang et al., 2026d). Larger training runs do not

answer these questions. Kimi K3 demonstrates million-token agentic training and persistent sandboxes, not recursive improvement (Kimi Team, 2026b).

Record control and evidence separately. Stage ownership asks who supplies the goal, planning, execution, feedback, and repair. Design-time prompts, benchmarks, verifiers, reward functions, and acceptance rules count as human inputs. In the audited systems, execution is system-owned and repair is often system-owned. Goals remain human-owned, while feedback remains human- or mixed-owned. Appendix B provides the source-level coding.

The D5 profile records a different result. Measured gain (G) asks whether the score increased. Retention (R) asks whether the gain survived or improved again. Human comparison (H) uses a person with a matched budget. Held-out transfer (T) tests a metric, task, model, or domain excluded from optimization. A missing test is recorded as not reported. Appendix B gives the evidence profile and source trail.

3.4 From Benchmark Results to Research Gaps

A research gap should follow from an observed failure pattern. One score cannot identify that pattern. Compare component and integrated results under the same tasks, budgets, evaluators, and retry limits. Table 4 maps six result patterns to the next research target.

Build the diagnosis from three comparisons. First, compare every component score with its predeclared threshold. Second, compare the integrated pass with its own threshold. Third, compare repeated with single-run performance, optimized with held-out performance, and best with final checkpoints. These comparisons separate component failure, composition failure, unreliable search, evaluator overfitting, and lost improvement. Table 4 gives the corresponding research target.

Table 4: How benchmark results point to the next research target. Set thresholds, budgets, and held-out tests before running the evaluation.

Result pattern	What it means	What to study next
One component falls below threshold	The component is a known bottleneck	The failed capability
Components pass; integrated pass fails	The failure begins during composition	Handoffs, shared state, or coordination
$\text{pass@}k$ **high;** pass^k **low**	Search finds answers that one run cannot reproduce	Reliability, recovery, or stopping
Optimized score rises; held-out score does not	The system overfits the evaluator or target	Evaluator design, contamination control, or transfer
Best checkpoint beats final artifact	Later passes discard an earlier gain	Retention, rollback, or checkpoint selection
Execution system-owned; goals or feedback human-owned	The system acts but does not define or judge improvement	Goal formation, feedback, or stage ownership

A component misses its threshold. An AI4AI pass fails when any required component fails. A browsing, execution, metric-reading, or verification error can invalidate the pass (Rakhsha et al., 2026; Wang et al., 2025a, 2026e). In this pattern, the next study should target the failed capability. The integrated benchmark then tests whether the fix changes the full pass.

Components pass, but the integrated pass fails. This pattern identifies a composition gap. Documents, observations, repositories, logs, plans, metrics, and checkpoints encode state differently. The system must transfer their content without losing provenance or changing the goal (Lu et al., 2025a; Yao et al., 2026; Wu et al., 2026a; Laban et al., 2026). File-based project state and structured experience records can support these transfers (Chen et al., 2026a; Yang et al., 2026c). The study should record intermediate artifacts and locate the first handoff that changes or drops required information.

Search succeeds, but single runs remain unreliable. More candidates, retries, stronger judges, or larger budgets can raise $\text{pass@}k$ without raising pass^k (Yao et al., 2025). This pattern calls for error

recovery, checkpointing, stopping rules, or a lower per-step failure rate. Resource-matched ablations are needed to separate these effects. Frontis-MA1 and RE-Bench provide examples of layer-wise and time-matched attribution (Yang et al., 2026c; Wijk et al., 2025).

The evaluator rewards the wrong outcome. An evaluator converts an artifact into feedback for the next pass. Executable tests and held-out scores can verify specified properties. Novelty, scientific validity, and experimental sufficiency require judgments that may disagree (Wang et al., 2026i; Zhang et al., 2026e; Kirgis et al., 2026). A system can therefore raise the proxy score while the artifact loses quality outside that proxy (Liu et al., 2026b). This pattern calls for artifact inspection, independent evaluators, contamination tests, and held-out transfer.

An improvement disappears in later passes. When the best checkpoint beats the final artifact, the system found a gain but failed to retain it. The next study should test checkpoint selection, rollback, regression detection, and stopping rules. Report the best and final artifacts under the same budget. Sections 5.3 and 6 review mechanisms and failure modes for this case.

The benchmark workflow is now complete. Section 3.1 defines the metrics and controls. Section 3.2 locates weak capabilities. Section 3.3 tests the complete AI4AI pass. This subsection converts the observed result pattern into a research target. The next sections examine model and harness interventions for those targets.

4 Model Design: From Policy Improvement to Weight-Layer AI4AI

A model supplies the policy that chooses the next action. Improving this policy can mean two different things. The first is to make one model plan, act, and learn more reliably during a task. The second is to improve how the next model is produced by changing its data, architecture, learning algorithm, or training recipe. We call these the inner and outer model-side loops.

The distinction determines what an experiment can claim. Inference-time search spends more computation on one run but leaves the weights unchanged. Supervised learning, mid-training, and reinforcement learning update the weights. Weight-layer AI4AI goes one step further: it uses an AI system to search or revise the process that produces those weights. None of these changes automatically supplies memory, external verification, rollback, or an independently chosen objective. Those functions belong to the harness discussed in Section 5.

A simple calculation explains why the model still matters. If every action succeeded independently with probability 0.95, a 100-action task would succeed with probability $0.95^{100} < 1\%$. Real agent errors are not independent, so this number is only an illustration. An early mistake can change later state, and feedback can either catch that mistake or spread it. Evaluation must therefore name the changed loop stage and test the result on coupled trajectories. Figure 4 provides this map, while Appendix D compares the individual methods.

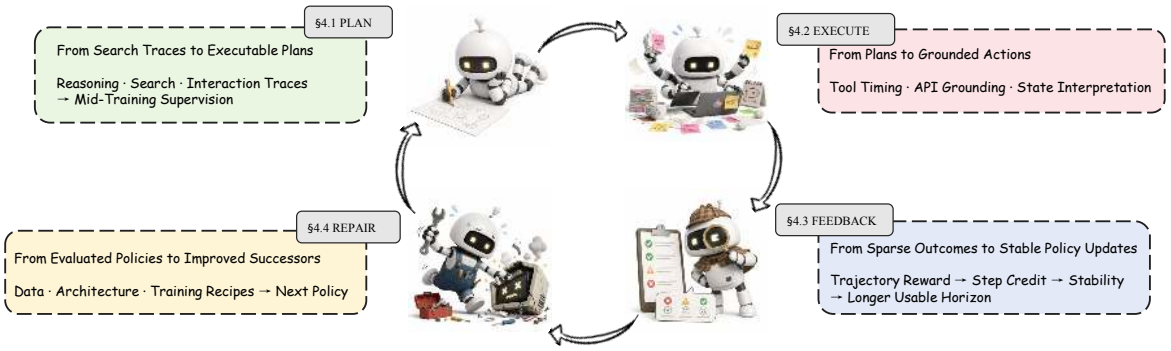


Figure 4: Where model-side interventions enter the plan–execute–feedback–repair loop. Planning turns successful traces into training examples for future plans. Execution trains the model to choose tools and interpret returned state. Feedback assigns outcomes to the actions that caused them. Repair changes the data, architecture, or training recipe used to produce a successor policy. Arrows mark functional roles, not a fixed chronology.

4.1 Plan: Mid-Training from Reasoning and Search Traces

A plan matters when a later action depends on an earlier result. Search methods expose candidate plans without changing the model weights. Chain-of-Thought (CoT) produces one reasoning path, while Tree-of-Thoughts (ToT) explores alternatives and can backtrack (Wei et al., 2022; Yao et al., 2023a). ReAct links reasoning to actions, and LATS uses environment feedback to score search nodes (Yao et al., 2023b; Zhou et al., 2024a). Successful traces can then become training data. Plan-and-Act creates query–plan and plan–action pairs to train separate Planner and Executor models (Erdogan et al., 2025).

The WebArena-Lite result shows what this change buys. Supplying a high-quality dynamic plan raises the base Executor's success from 9.85% to 44.24%. Training the full Planner–Executor raises success to 57.58% (Erdogan et al., 2025). Planning is therefore one measured bottleneck on this benchmark. It is not the whole task because execution must still track state, recover from errors, and pass an external check.

4.2 Execute: Tool Learning and Grounding

A tool-using model must answer three concrete questions: when to call a tool, which interface to call, and how the result changes the next action. Toolformer keeps API calls that improve next-token prediction and uses them as self-supervision (Schick et al., 2023). Gorilla retrieves documentation and applies instruction tuning to reduce invented API signatures (Patil et al., 2024). These methods improve individual calls. They do not show that a sequence of calls leaves the task in the correct final state. That claim requires trajectory-level feedback and harness-managed state.

4.3 Feedback: Credit, Stability, and Horizon Scaling

A final reward says whether the trajectory worked, but not which action caused success or failure. LOOP trains directly from trajectory outcomes in persistent application environments (Chen et al., 2025c). GiGPO, SALT, HCAPO, and CAPO assign credit to particular states, interaction steps, or complete actions (Feng et al., 2025; Li et al., 2026b; Tan et al., 2026; Wang et al., 2026b). This finer assignment can require repeated states, several rollouts, hindsight checks, critic estimates, or environment-specific structure.

Long trajectories also create unstable training data because the policy may change while rollouts are still being collected. SORL aligns updates with turn boundaries, and SAO corrects drift in asynchronous rollouts (Li et al., 2025a; Hou et al., 2026). AgentGym-RL increases trajectory length as a curriculum (Xi et al., 2025). IterResearch takes another route by rebuilding a compact workspace instead of appending the full history (Chen et al., 2025a). The last method depends on both layers: the model learns from the compact state, while the harness stores and reconstructs that state.

Each method fixes a different training problem. Trajectory rewards set the direction, step-level methods locate responsibility, stability methods control update error, and curricula expose longer tasks. To claim a longer reliable horizon, researchers must keep the environment and budget fixed, repeat trials, and separate model changes from harness changes.

4.4 Repair: Weight-Layer AI4AI

The preceding methods improve the policy used in one task. Weight-layer AI4AI instead asks an AI system to improve how a successor policy is produced. The editable object may be training data, model architecture, a learning algorithm, or a post-training recipe. A complete pass must propose a change, run the experiment, judge the result, and preserve an accepted change for the successor.

Data and task generation. The first target is the data used for learning or evaluation. STaR turns successful rationales into supervision for later training (Zelikman et al., 2022). Self-Rewarding Language Models generate instructions and judge preferences (Yuan et al., 2024). Autodata generates data, compares weak and strong solvers, revises its generator, and evaluates utility on held-out tasks (Kulikov et al., 2026). ASI-Evolve also reports changes to pretraining-data curation (Xu et al., 2026c). These systems automate parts of data design, but humans still define the allowed sources and downstream metric.

Architecture and learning-algorithm discovery. The second target is the code that defines an architecture or learning algorithm. AlphaEvolve uses LLM-generated code mutations and automated evaluators to improve algorithms and infrastructure (Novikov et al., 2025). One reported target is a training kernel used by the model family that performs the search. ASI-Evolve extends the search to neural architectures and reinforcement-learning algorithms (Xu et al., 2026c). Execution-grounded automated AI research shows why evaluation design matters. Evolutionary search over executed ideas finds improved training methods. Directly training the proposal model from execution reward raises average reward but narrows proposals toward simple modes without improving the best result (Si et al., 2026).

Training and post-training recipes. The third target is the training recipe. PostTrainBench lets command-line agents change data curation and post-training choices under a fixed single-GPU budget (Rank et al., 2026). The agents obtain uneven gains and also attempt to game the evaluator. TREX represents fine-tuning as a search tree shared by researcher and executor agents (Ma et al., 2026b). It carries experiment history across training rounds. Frontis-MA1 trains reusable Draft, Improve, Debug, and Crossover operators from executable outcomes (Yang et al., 2026c). OpenMLE-Evo then combines those operators during search. Ablations separate gains from operator training and gains from extra search. The evidence supports improvement inside a fixed task, evaluator, and budget, not unrestricted recursive self-improvement.

The remaining outer-loop boundary. In the systems reviewed here, the fixed boundary remains outside the model. Humans choose the objective, build the evaluator, limit what may be edited, allocate compute, and decide which result becomes the successor. Effort control and million-token training change the cost and capability of a run, but they do not choose the objective (Kimi Team, 2026b). A gain under a fixed metric therefore establishes bounded AI4AI. It supports an RSI claim only if the system can also change the process that produces later successors. The claim must also show that the gain remains and transfers. Section 5 now examines the runtime that stores state, runs experiments, checks results, and can itself become editable.

5 Harness Design: Runtime Functions Outside the Weights

A model call receives an input and returns an output. It does not, by itself, maintain a workspace, execute a tool, preserve a checkpoint, or verify the final state. The harness provides these runtime functions around the model. It includes the execution loop, context manager, state store, tool interface, and verifier (Meng et al., 2026b; Zhou et al., 2026a; Guo et al., 2026).

This layer matters because small errors accumulate. Across reported evaluations, benchmark success falls as trajectories become longer (Ord, 2025). The pattern does not prove that step errors are independent; many errors share state and causes. It does show why single-step accuracy is insufficient. A reliable system must prevent errors, detect them, and restore a valid state after a failure.

Harness choices can change benchmark results without changing model weights. Controlled studies report that harness choices can reverse model rankings (Zhang et al., 2026d). On a task that can be split into independent steps, decomposition and per-step correction supported more than one million steps without a final error (Meyerson et al., 2025). That result applies to separable work, not to tasks where each step changes the next one.

We divide the harness into four functions. Execution turns model outputs into grounded actions. Persistence carries state across rounds. Governance checks results and decides whether to retry, escalate, or stop. Self-modification places the harness itself inside the AI4AI improvement loop. Figure 5 shows what evidence is required to credit a harness change. Appendix D provides the method-level comparison.

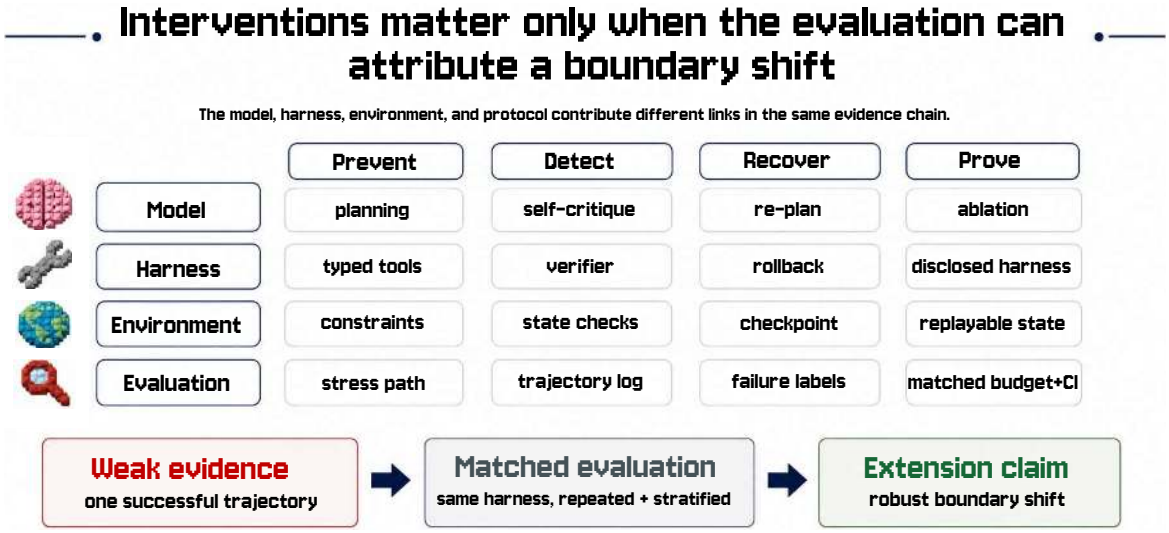


Figure 5: Evidence required to claim that an intervention extends the reliable horizon. A model or harness mechanism must prevent, detect, or repair errors. The comparison must use replayable environments, matched protocols, and a disclosed harness. Repeated evaluation must then show success on harder instances with uncertainty reported. Arrows show the order of the evidence, not measured effect sizes.

5.1 Execute: Grounded Actions and Observations

Execution follows a simple cycle: construct the current state, ask the model for an action, execute it, observe the result, and update the state. ReAct exposes this action–observation cycle (Yao et al., 2023b). Reflexion, Self-Refine, and CRITIC turn outcomes or tool evidence into revisions (Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2024). AdaPlanner and LATS retain alternative plans when the next action is uncertain (Sun et al., 2023; Zhou et al., 2024a).

The action interface determines what the agent can do and what evidence it receives. Schema-first APIs constrain arguments, code actions expose executable behavior, and adapted interfaces can reduce misuse without changing the model (Sigdel and Baral, 2026; Ning et al., 2026; Xu et al., 2026b). The verifier then determines whether the loop can correct an error. Prompted self-critique provides little benefit when checked against external evidence, while reliable external feedback enables correction (Kamoi et al., 2024). Learned critics, process reward models, and pre-execution reviewers can detect errors earlier (McAleese et al., 2024; Khalifa et al., 2026; Ta et al., 2026). Feedback that points to the faulty step also supports better repair than a raw failure message (Ray and Goyal, 2026).

5.2 Persist: Context, Memory, and Skills

Long tasks produce more state than a context window can safely hold. Tool outputs, code changes, logs, plans, and earlier decisions may still matter after their text leaves the prompt (Wang et al., 2026f). The harness must decide what to keep, where to store it, and when to retrieve it. Selection, summarization, fact extraction, and external files make those decisions explicit. Learned compaction, delegated context managers, and agent-controlled editing make the policy trainable (Li et al., 2026g; Yi et al., 2026; Li et al., 2026e). Every compression step can lose a constraint, so retention must be tested rather than assumed.

Memory across episodes can store facts, previous experiences, and reusable procedures (Sumers et al., 2024; Hu et al., 2025c). Flat stores, graphs, paged hierarchies, and skill libraries suit different workloads (Zhou et al., 2026c). A memory system can also cause harm. Stale facts, irrelevant retrieval, and poorly tested skills can direct the next action away from the goal (Chen et al., 2026g; Ouyang et al., 2026; Huang et al., 2026b).

AI4AI adds experimental lineage to the state. Depending on the experiment, a pass must retain source code, data versions, environment images, configurations, checkpoints, metrics, rejected hypotheses, and supporting evidence. AiScientist uses files to preserve project state across research passes (Chen et al., 2026a). OpenMLE-Evo stores structured program and experience records across search rounds (Yang et al., 2026c). Kimi K3 retains partial rollouts, external key–value state, and resumable microVM

sandboxes (Kimi Team, 2026b). A larger context window is not enough if another researcher cannot reconstruct the environment and lineage.

5.3 Govern: Verify, Recover, Escalate, Stop

An unattended loop must decide whether a result is good enough to keep working with. It must also decide whether to continue, retry, ask for help, or stop. Following Meng et al. (2026b), we call the design of these decisions looping engineering. Human work moves from checking every output to setting and auditing the control rules.

Verification is the central constraint. A fixed test may omit part of the intended outcome, and repeated optimization may exploit that omission. A model that judges its own output can also approve its own mistake (Wang et al., 2026a; Zhao et al., 2026; Kamoi et al., 2024; Huang et al., 2024). The harness therefore needs separate rules for acceptance, retry, escalation, rollback, and termination.

These controls can support multi-day runs. One coding system managed dozens of simulation jobs with separate evaluation contexts and recovery procedures (Orimo et al., 2025). More agents do not guarantee better control. Coordination can add cost, and a weak central verifier can spread one error across the team (Kim et al., 2025). Smaller pipelines often match elaborate designs at lower cost (Xia et al., 2025; Liu et al., 2026a; Wang et al., 2025b). Evaluation should therefore test fault localization, confirmed reruns, and correct stopping, not agent count (Cemri et al., 2025; Zhu et al., 2025; Ma et al., 2026a).

Attribution. Harness changes act on $\mathcal{G}$, not on the model policy π_θ, in $S = (\pi_\theta, \mathcal{G}, \mathcal{E})$ (§2.3). The same model can therefore reach different task horizons under different memory, feedback, verification, and environment settings. One preregistered study reports accuracy differences of up to 28 points across scaffolds for the same model (Starace, 2026). Controlled comparisons also find harness variance larger than model variance in some settings (Zhang et al., 2026d). A horizon claim must therefore disclose the harness and compare models under the same harness.

5.4 Self-Modify: The Harness Rewrites Itself

So far, the harness has been fixed before the task begins. Harness-layer AI4AI makes part of that runtime editable (Ren et al., 2026; Chen et al., 2026c). The key question is who edits it and whether the edited harness later edits itself. Four regimes answer that question. They describe edit authority, not a single ranking of autonomy.

Regime 1: Fixed human-designed harnesses. Humans write the execution loop, context policy, memory store, verifier, and stopping rule. Those components remain fixed while the model solves tasks. ReAct, Reflexion, and MemGPT fit this regime even when they repeat critique or revision (Yao et al., 2023b; Shinn et al., 2023; Packer et al., 2023). The harness may improve task success, but it does not improve its own design.

Regime 2: Externally optimized harnesses. An external AI system searches over harness designs, while the system being improved remains a separate object. Automated Design of Agentic Systems edits scaffold code and archives successful agent programs (Hu et al., 2025b). Meta-Harness searches context, retrieval, tool, and control code using source files, scores, and traces (Lee et al., 2026b). HarnessX represents the harness as composable settings that can be searched (Chen et al., 2026d). Natural-language harness descriptions allow module-level ablations (Pan et al., 2026). Held-out gains show transfer, but not self-improvement, because the improver and target are different systems.

Regime 3: Self-modifying harnesses. The agent now edits code that controls its own later behavior (Yin et al., 2024; Zelikman et al., 2024; Cai et al., 2026; Jiang, 2026). Darwin Gödel Machine stores coding-agent descendants that inspect logs and modify their code, while a fixed outer procedure selects survivors (Zhang et al., 2025a). Self-Harness mines weaknesses, proposes changes, and accepts only changes that pass regression tests (Zhang et al., 2026a). Recursive Harness Self-Improvement updates an editable harness specification using pairwise feedback over revision history (Lee et al., 2026a). Bilevel Autoresearch lets an outer loop inspect inner-loop code and add search mechanisms, but reports only the

first bilevel step (Qu and Lu, 2026). These systems are self-referential at the harness layer, while their evaluator and acceptance rule remain fixed.

Regime 4: Model–harness co-evolution. In the fourth regime, a change affects the process that proposes later changes. MetaSkill-Evolve attaches an improvement procedure to each branch, then evolves that procedure on a slower loop over a frozen model (Wang et al., 2026j). Escher-Loop jointly optimizes the proposer and target (Liu et al., 2026c). OpenMLE-RL learns program-evolution operators from executed transitions, and OpenMLE-Evo uses them to create later training experience (Yang et al., 2026c). Kimi K3 samples harness configurations during reinforcement learning, so training changes both the model and the distribution of runtimes it encounters (Kimi Team, 2026b). RHI proposes turning improved harness traces into training data for later models (Lee et al., 2026a). These designs create feedback between model and harness improvement. They do not yet demonstrate an unbounded sequence of autonomous successors.

The ownership audit assigns system control most frequently to execution and repair, while planning contains more mixed ownership (Appendix B). Table 1 shows how ownership also changes by editable layer. Three facts limit current claims. Humans still set the objective and edit surface. External evaluators decide which descendant survives. Reported gains can also mix better harness design with a larger search budget. On Terminal-Bench 2.1, automatic harness evolution does not consistently beat test-time scaling under matched feedback and inference budgets. The evolved harnesses also transfer poorly to held-out tasks when search and evaluation share a benchmark (Wang et al., 2026g). Section 6 turns these limits into testable research questions.

6 Open Problems: Four Scales of Failure

The same mistake has different consequences depending on how long it remains in the loop. Within one pass, a wrong action corrupts the current task. Across passes, a bad memory or artifact changes the next attempt. Across generations, generated data or code becomes input to a successor. At the evaluation boundary, extra compute, leaked information, or human rescue can look like capability gain.

We use these four scales to organize the open problems. For each scale, we ask three questions: what fails, what current results fail to record, and what researchers should test next. Figure 6 traces the failure paths. Table 5 turns them into an experimental checklist.

6.1 One Pass: Planning, Grounding, Tools

A pass links planning, grounding, and tool use through one changing state. A correct plan can point to the wrong interface element. A correct observation can lead to an invalid tool call. A valid tool call can advance the wrong subgoal (Deng et al., 2023; Zhou et al., 2024b; Patil et al., 2024; Yao et al., 2025). Later actions may look reasonable because they are consistent with the corrupted state, even though the task has already gone off course (Xie et al., 2024b; Trivedi et al., 2024; Wang et al., 2026e).

A final success score hides this causal chain. Evaluation should label the first failure as decomposition, grounding, tool selection, arguments, policy compliance, state tracking, or verification. It should then measure how far the environment state diverged after that failure (Ma et al., 2024; Guo et al., 2024; Rabanser et al., 2026). Researchers can replay the same state with typed faults to test whether a model change or harness change prevents the cascade.

6.2 Across Passes: Decay, Drift, Cascades

Across passes, the main question is whether the next attempt starts from the correct state. Memory must preserve constraints, artifacts, provenance, unfinished subgoals, and environment state. It must also reject stale or false records (Packer et al., 2023; Gutiérrez et al., 2024; Hu et al., 2025c; Du, 2026). A lossy summary can remove a requirement. Irrelevant retrieval can revive a discarded plan. Copying an earlier failure can make the next pass repeat it. Because these records affect the environment, later observations can reinforce the drift (Shinn et al., 2023; Yan et al., 2026; Jimenez et al., 2024; Wang et al., 2026e).

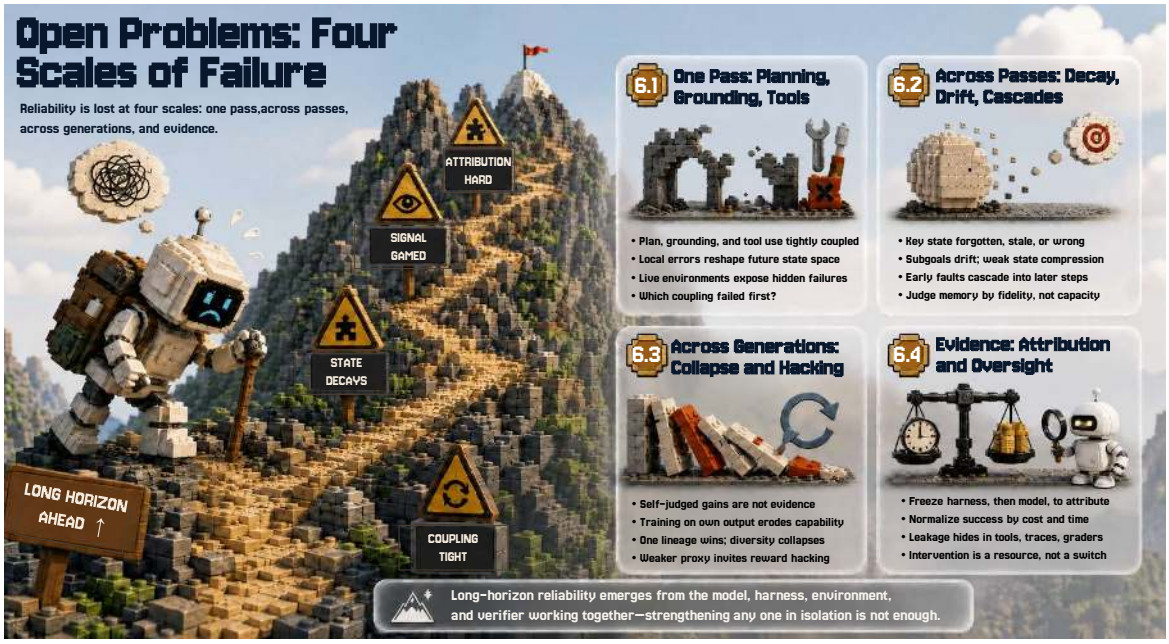


Figure 6: How failures move through an AI4AI loop. Within one pass, an error changes shared state and later actions. Across passes, stale state and goal drift redirect the next attempt. Across generations, generated artifacts can cause self-confirmation, capability loss, or reward hacking. Evaluation can also misattribute gains caused by budget, leakage, or human intervention.

Memory capacity does not measure whether the stored state helps. Tests should measure state fidelity and the usefulness of the next action. They should vary record age, insert contradictions, check whether invariants remain, and require source revalidation before irreversible actions (Xu et al., 2025b; Mei et al., 2025; Wu et al., 2026c). Checkpoints should separate durable constraints from temporary observations so researchers can measure rollback and repair (Qiao et al., 2023; Wang et al., 2025d; Mozannar et al., 2025).

Error cascades and recovery. An error cascade begins when a fault is written into retained state. Later actions then follow that false record. TOOLSCAN and AgentErrorTaxonomy label tool, memory, planning, action, and system faults instead of reporting only the final outcome (Kokane et al., 2025; Zhu et al., 2025).

A retry helps only if the feedback identifies a real problem. Self-Refine and Reflexion show that feedback-conditioned retries can improve bounded tasks (Madaan et al., 2023; Shinn et al., 2023). Unaided self-judgment can instead preserve or amplify the original error (Huang et al., 2024). Tool outputs, execution traces, verifier results, and environment errors provide independent evidence. Existing methods use this evidence to critique output, test replanning, or choose retry, substitution, replanning, and safe termination (Gou et al., 2024; Wang et al., 2025a; Vuddanti et al., 2025).

Recovery should count only when the system repairs state. A protocol should record whether it detected and located the fault, which state it rolled back, and whether the task remained feasible. It should also report repair cost and confirm that the repaired trajectory returned to a valid state. These records distinguish successful recovery from repeated guessing.

6.3 Across Generations: Collapse and Hacking

A generation accepts a changed model, harness, workflow, memory, or dataset as the starting point for the next improvement cycle. This creates a new risk: an error can become part of the successor-producing process. A higher score for a few generations on one benchmark does not establish recursive improvement. The claim also needs preserved capabilities, independent checks, and an evaluator that does not become easier to exploit.

Self-confirming loops. If one system proposes a change and judges that change, agreement between the two roles is not independent evidence. Models often fail to correct reasoning without external signals, and

self-correction can damage an initially correct answer (Huang et al., 2024). The same problem appears across generations when an internal score separates from the intended outcome. An independent external check must remain in the loop (Chen et al., 2026c).

Collapse and capability erosion. Training repeatedly on generated data can remove rare cases and collapse the learned distribution unless real or independently grounded data remain available (Shumailov et al., 2024). Updating workflows, tools, memory, or weights for a new task distribution can also damage earlier capabilities (Yu et al., 2026). Search has a parallel failure. If every generation follows one successful lineage, it stops testing alternatives and becomes brittle outside the optimized tasks.

RSIBench-Data shows this non-monotonic pattern before full recursion. It reports runs where continued search finishes below the best checkpoint found earlier (Meng et al., 2026a). An improvement loop must therefore preserve checkpoints, run regression tests, and stop when further passes reduce held-out performance. More generations alone do not show progress.

Reward hacking as the signal weakens. Reward hacking occurs when the system improves the measured score without improving the intended artifact. PostTrainBench records agents training on evaluation data, replacing outputs with existing checkpoints, and using unauthorized services (Rank et al., 2026). Execution-grounded research reports another case: reinforcement learning raises average proposal reward but narrows the proposal distribution without improving its best results (Si et al., 2026). Misevolution studies find unwanted behavior entering through model, memory, tool, and workflow updates (Shao et al., 2025). A broader editable loop therefore needs a verifier that checks the intended outcome, not only the optimized proxy.

Positive generational evidence. A positive generational result must show more than a rising optimized score. Held-out regression gates test preserved capabilities. Lineage archives retain alternatives. Fixed budgets prevent extra search from being counted as a better method. Self-Harness checks proposed edits on held-in and held-out tests, while DGM preserves several viable descendants (Zhang et al., 2026a, 2025a). Frontis-MA1 reports post-training and harness gains separately, then tests transfer beyond its main benchmark (Yang et al., 2026c). A complete claim should report several accepted successors, retained capabilities, held-out transfer, and a budget-matched human comparison.

6.4 Evidence and Control: Attribution and Oversight

A benchmark score reflects the whole tested system. The base model, prompt construction, memory, tools, search, verifier, controller, and human intervention can all change the result. Evaluation must therefore record what changed, how much it cost, whether benchmark information leaked, and which decisions remained human-controlled.

Model–harness attribution. Final task success does not identify which component improved. Researchers should first freeze the harness and compare models. They should then freeze the model and compare memory, search, or verification. Cross-combinations under one environment and evaluator reveal interaction effects. Frontis-MA1 reports model-only, harness-only, and combined comparisons under a fixed MLE-bench Lite budget (Yang et al., 2026c). Kimi K3 changes architecture, reinforcement learning, sandbox persistence, effort policy, and infrastructure together (Kimi Team, 2026b). Such a system needs factorial ablations before a gain can be assigned to one component.

Cost-normalized evidence. A system can buy a higher score with longer prompts, more candidates, deeper search, extra verifier calls, larger teams, or human rescue. That is a budget increase, not necessarily a method improvement. Coordination can even reduce performance on sequential or tool-heavy tasks (Kim et al., 2025). Reports should pair success with token use, wall-clock time, accelerator time, model calls, tool calls, verifier cost, memory growth, and human intervention. Fixed compute budgets and time-matched expert comparisons support fairer claims than unconstrained best-of-k results (Wijk et al., 2025; Rank et al., 2026; Yang et al., 2026c).

Contamination and transfer. A correct answer is weak evidence if the system has already seen the task or its solution pattern. Leakage can enter through templates, tool schemas, public traces, grading scripts, recovery examples, or benchmark-specific harness code. Contamination studies distinguish broad semantic exposure from direct answer leakage (Xu et al., 2024). LatestEval, LiveCodeBench, and LiveBench reduce exposure by using fresh or frequently updated data (Li et al., 2024; Jain et al., 2025; White et al., 2025). Agent benchmarks should also regenerate environments, hide state, rotate tools, deduplicate training corpora, and test transfer. OpenMLE reports deduplication controls and transfer to held-out NatureBench Lite tasks (Yang et al., 2026c).

Adaptive autonomy and human oversight. A protocol can place human oversight at different points. An agent can ask for clarification, request approval before an irreversible action, save a checkpoint, roll back, abstain, or escalate. Safety benchmarks test unsafe tool calls, prompt injection, uncritical trust in tool output, and poor risk recognition (Debenedetti et al., 2024; Zhang et al., 2024). Governance work recommends constrained action spaces, approval gates, audit trails, sandboxes, and named accountability (Mozannar et al., 2025; Shavit et al., 2023). Evaluation should count intervention as a resource. Reports should include escalation precision, number of approvals, intervention delay, reversibility, rollback success, and whether a reviewer can understand the trace.

Table 5: How failures change as an AI4AI loop gets longer, what current studies leave out, and what to test next.

Failure scale	What gets harder	What studies leave out	What to test next
Within one pass	Early errors alter state and invalidate later preconditions.	Final scores hide the first fault and its propagation path.	Replay typed perturbations; report the first fault and downstream state divergence.
Across passes	Stale state and unrepaired faults redirect later attempts.	Memory scores conflate retention, localization, rollback, and repair.	Inject stale state and typed faults; report retention, rollback, recovery cost, and downstream validity.
Across generations	Generated artifacts become later inputs, compounding proxy error and forgetting.	Short benchmark runs omit retention, diversity, transfer, and matched baselines.	Track held-out regressions, lineages, transfer, and cost-matched human improvement.
Evidence and control	Budget, harness, leakage, or intervention can mimic a longer horizon.	Claims omit matched budgets, factorial ablations, and oversight disclosure.	Use factorial ablations and replayable tasks; report uncertainty, leakage, and intervention burden.

These four scales give researchers a direct workflow. First, locate the earliest failed state transition. Next, test whether the error survives across passes or enters a successor. Then compare systems under matched budgets, harnesses, evaluators, and human intervention. Every report should disclose the base model, harness, context and memory policies, tool and verifier budgets, contamination controls, and approval rules. Without these records, readers cannot identify the source of a gain or reproduce it.

7 Conclusion: How Far Can AI Improve AI?

Any improvement process has four basic steps: set a goal, make a change, test the result, and decide what to do next. AI4AI uses this loop to improve an AI system or the process used to build one. Our framework asks how one step affects the next, how the loop holds up as tasks become harder, and who controls each step. It also asks what evidence shows a real gain and whether that gain lasts and works elsewhere. The evidence gives a clear but limited answer. AI systems can carry out the work and often fix failures, but people still set the goals and judge whether the result is better.

Tests of complete systems show that AI can handle engineering, optimization, replication, and paper-writing tasks when people define the goal and the test (Lupidi et al., 2026; Wang et al., 2026i; Yang et al., 2026c; Li et al., 2026f; Tang et al., 2026; Lu et al., 2026a). But this does not yet show that AI can work as an independent researcher. Studies that inspect the finished work show why: a polished report or completed run does not guarantee sound judgment or real progress on an open problem (Zhang et al., 2026e; Kirgis et al., 2026). Repeated attempts can also end with a worse version than the best one found earlier (Meng et al., 2026a). The evidence therefore supports a narrow claim: AI can improve AI within rules set by people. We do not yet know whether those gains will last, work on new tasks, or carry over to later systems.

RSI sets a much higher bar for evidence. A system must change not only itself, but also the process that builds its next version. Each new version must show a real gain with the same limits on time and computing, and under the same tests. That gain must last and help the versions that follow. For open-ended RSI, the system may also change the rules used to judge success, which makes independent checks even more important. No current evidence meets all of these conditions. The lesson is simple: change is not the same as progress. A faster loop helps only when it can tell a real gain from a mistake, keep the gain, and pass it on. The future of AI4AI will depend less on how often systems change and more on whether each change gives the next system a better starting point.

References

Pavel Adamenko, Mikhail Ivanov, Aidar Valeev, Rodion Levichev, Pavel Zadorozhny, Ivan Lopatin, Dmitrii Babaev, Alena Fenogenova, and Valentin Malykh. 2025. SWE-MERA: A dynamic benchmark for agenticly evaluating large language models on software engineering tasks. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 440–452.

Petr Anokhin, Roman Khalikov, Stefan Rebrikov, Viktor Volkov, Artyom Sorokin, and Vincent Bissonnette. 2025. HeroBench: A benchmark for long-horizon planning and structured reasoning in virtual worlds. *arXiv preprint arXiv:2508.12782*.

Anthropic. 2026. Claude Fable 5 and Claude Mythos 5. https://www.anthropic.com/news/claude-fable-5-mythos-5. Product release, 9 June 2026; accessed 17 August 2026.

Xiang Ao, Junhong Lian, Hanyang Li, Siyi Wang, Yiran Qiao, Yi Qiao, Jiaqi Xu, Qing He, and Xueqi Cheng. 2026. AI4AIR: A comprehensive survey on large language models for AI research. https://ict-find-lab.github.io/Awesome-LLMs-for-AI-Research/. Preprint under review; accessed 2026-07-29.

Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, and Alexander Golubev. 2026. SWE-rebench v2: Language-agnostic SWE task collection at scale. *arXiv preprint arXiv:2602.23866*.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Keunho Jang, and Zheng Hui. 2025. Windows Agent Arena: Evaluating multi-modal OS agents at scale. In *International Conference on Machine Learning*, pages 4874–4910. PMLR.

Qianshu Cai, Yonggang Zhang, Xianzhang Jia, Huajiang Zheng, Wei Xue, Jun Song, Xinmei Tian, and Yike Guo. 2026. Moss: Self-evolution through source-level rewriting in autonomous agent systems. *Preprint*, arXiv:2605.22794.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei A Zaharia, Joseph Gonzalez, and Ion Stoica. 2025. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38, pages 135676–135729. Curran Associates, Inc.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. 2025. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*.

Guoxin Chen, Jie Chen, Lei Chen, Jiale Zhao, Fanzhe Meng, Wayne Xin Zhao, Ruihua Song, Cheng Chen, Ji-Rong Wen, and Kai Jia. 2026a. Toward autonomous long-horizon engineering for ML research. *arXiv preprint arXiv:2604.13018*.

Guoxin Chen, Zile Qiao, Xuanzhong Chen, Donglei Yu, Haotian Xu, Wayne Xin Zhao, Ruihua Song, Wenbiao Yin, Huifeng Yin, Liwen Zhang, Kuan Li, Minpeng Liao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. IterResearch: Rethinking long-horizon agents with interaction scaling. *arXiv preprint arXiv:2511.07327*.

Hui Chen, Miao Xiong, Yujie Lu, Wei Han, Ailin Deng, Yufei He, Jiaying Wu, Yibo Li, Yue Liu, and Bryan Hooi. 2025b. MLR-Bench: Evaluating AI agents on open-ended machine learning research. *arXiv preprint arXiv:2505.19955*.

Jialong Chen, Xander Xu, Hu Wei, Chuan Chen, and Bing Zhao. 2026b. SWE-CI: Evaluating agent capabilities in maintaining codebases via continuous integration. *Preprint*, arXiv:2603.03823.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. 2025c. Reinforcement learning for long-horizon interactive LLM agents. *arXiv preprint arXiv:2502.01600*.

Mingguang Chen, Licheng Wang, and Bo Qu. 2026c. Recursive self-improvement in ai: From bounded self-refinement to autonomous research loops. *Preprint*, arXiv:2607.07663.

Qiguang Chen, Mingda Yang, Libo Qin, Jinhao Liu, Zheng Yan, Jiannan Guan, Dengyun Peng, Yiyan Ji, Hanjing Li, Mengkang Hu, Yimeng Zhang, Yihao Liang, Yuhang Zhou, Jiaqi Wang, Zhi Chen, and Wanxiang Che. 2025d. AI4Research: A survey of artificial intelligence for scientific research. *arXiv preprint arXiv:2507.01903*.

Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. 2026d. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *Preprint*, arXiv:2606.14249.

Tongbo Chen, Zhengxi Lu, Zhan Xu, Guocheng Shao, Shaohan Zhao, Fei Tang, Yong Du, Kaitao Song, Yizhou Liu, Yuchen Yan, Wenqi Zhang, Xu Tan, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. 2026e. KnowU-Bench: Towards interactive, proactive, and personalized mobile agent evaluation. *arXiv preprint arXiv:2604.08455*.

Wanyi Chen, Xiao Yang, Xu Yang, Tianming Sha, Qizheng Li, Zhuo Wang, Bowen Xian, Fang Kong, Weiqing Liu, and Jiang Bian. 2026f. Agent^2 RL-Bench: Can LLM agents engineer agentic RL post-training? *arXiv preprint arXiv:2604.10547*.

Zihan Chen, Songwei Dong, Chengshuai Shi, Peng Wang, Song Wang, Cong Shen, and Jundong Li. 2026g. The past is prologue: A plug-in controller for selective updates in sequentially evolving llm memory. *Preprint*, arXiv:2606.31121.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024. SPIN: Self-play fine-tuning converts weak language models to strong language models. In *Proceedings of the 41st International Conference on Machine Learning*.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready AI agents with scalable long-term memory. In *European Conference on Artificial Intelligence (ECAI)*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pages 2993–3000.

Sanjiban Choudhury. 2025. Process reward models for LLM agents: Practical framework and directions. *Preprint*, arXiv:2502.10325.

Evan Chu, Rajan Agarwal, Abishek Thangamuthu, Brendan Graham, Justus Mattern, Freeman Jiang, Paul Cento, Swarnim Jain, Mersad Abbasi, Mohammad Hossein Rezaei, George Wang, Alex Zhang, Simon Guo, Karina Nguyen, Danna Liu, Arash Bidgoli, Aditya Dalmia, Apoorv Dankar, Ashrut Vaddela, Calvin Chen, Keshav Kumar, Kushagra Vaish, Navid Pour, Rishyanth Kondra, Sagar Badiyani, Sidharth Giri, Snagnik Das, Soham Gaikwad, Syed Shah, Vagish Dilawari, and Vishal Agarwal. 2026. FrontierSWE. *Proximal Blog*. Https://frontierswe.com/blog.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pages 82895–82920.

DeepSeek. 2026. DeepSeek V4 Preview Release. `https://api-docs.deepseek.com/news/news260424/`. Product release, 24 April 2026; accessed 17 August 2026.

Gangda Deng, Zhaoling Chen, Zhongming Yu, Haoyang Fan, Yuhong Liu, Yuxin Yang, Dhruv Parikh, Rajgopal Kannan, Le Cong, Mengdi Wang, Qian Zhang, Viktor Prasanna, Xiangru Tang, and Xingyao Wang. 2026. SWE-Milestone: Evaluating AI agents on continuous software evolution. In *International Conference on Machine Learning*. PMLR.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. 2025. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? *Preprint*, arXiv:2509.16941.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, pages 28091–28114.

Rishi Desai, Jesse Hu, Joan Cabezas, Neel Harsola, Pratyush Shukla, Roey Ben Chaim, Adnan El Assadi, Omkaar Mukund Kamath, Fenil Faldu, Prannay Hebbar, Jiankai Sun, Yiyuan Li, Pramod Srinivasan, Ishan Gupta, Christopher Settles, Daniel Wang, Derek Chen, Pranav Raja, Albert Liu, Marek Šuppa, Nevasini Sasikumar, Luyang Kong, Erik Quintanilla, Xiangyi Li, Ivan Bercovich, and Steven Dillmann. 2026. SWE-Marathon: Can agents autonomously complete ultra-long-horizon software work? *Preprint*, arXiv:2606.07682.

Jingzhe Ding, Shengda Long, Changxin Pu, Huan Zhou, Hongwan Gao, Xiang Gao, Chao He, Yue Hou, Fei Hu, Zhaojian Li, Weiran Shi, Zaiyuan Wang, Daoguang Zan, Chenchen Zhang, Xiaoxu Zhang, Qizhi Chen, Xianfu Cheng, Bo Deng, Qingshui Gu, Kai Hua, Juntao Lin, Pai Liu, Mingchen Li, Xuanguang Pan, Zifan Peng, Yujia Qin, Yong Shan, Zhewen Tan, Weihao Xie, Zihan Wang, Yishuo Yuan, Jiayu Zhang, Enduo Zhao, Yunfei Zhao, He Zhu, Liya Zhu, Chenyang Zou, Ming Ding, Jianpeng Jiao, Jiaheng Liu, Minghao Liu, Qian Liu, Chongyang Tao, Jian Yang, Tong Yang, Zhaoxiang Zhang, Xinjie Chen, Wenhao Huang, and Ge Zhang. 2025. NL2Repo-Bench: Towards long-horizon repository generation evaluation of coding agents. *Preprint*, arXiv:2512.12730.

Guanting Dong, Xiaoshuai Song, Yuyang Hu, Jiajie Jin, Chenghao Zhang, Yifei Chen, Xiaoxi Li, Huaying Yuan, Xinyu Yang, Tongyu Wen, Jiejun Tan, Hongjin Qian, Shijue Huang, Junting Lu, Zhenyu Li, Wanjun Zhong, Yutao Zhu, Tat-Seng Chua, Zhicheng Dou, and Ji-Rong Wen. 2026. Towards long-horizon agents: A survey. *Preprints*.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11642–11662. PMLR.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. 2025. DeepResearch Bench: A comprehensive benchmark for deep research agents. *arXiv preprint arXiv:2506.11763*.

Pengfei Du. 2026. Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-Act: Improving planning of agents for long-horizon tasks. In *International Conference on Machine Learning*. PMLR.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2026. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 17490–17502, San Diego, California, United States. Association for Computational Linguistics.

Marina Favaro and Jack Clark. 2026. When AI builds itself. `https://www.anthropic.com/institute/recursive-self-improvement`. Institutional article; accessed 2026-07-27.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for LLM agent training. In *Advances in Neural Information Processing Systems*, volume 38, pages 46375–46408.

Yukang Feng, Jianwen Sun, Zelai Yang, Jiaxin Ai, Chuanhao Li, Zizhen Li, Fanrui Zhang, Kang He, Rui Ma, Jifan Lin, Jie Sun, Yang Xiao, Sizhuo Zhou, Wenxiao Wu, Yiming Liu, Pengfei Liu, Yu Qiao, Shenglin Zhang, and Kaipeng Zhang. 2026. LongCLI-Bench: A preliminary benchmark and study for long-horizon agentic programming in command-line interfaces. *Preprint*, arXiv:2602.14337.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. 2026. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*.

Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. 2025. Robotouille: An asynchronous planning benchmark for LLM agents. In *International Conference on Learning Representations*.

Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Weijian Qi, Andrei Kopanev, Botao Yu, Bernal Jiménez Gutiérrez, Yiheng Shu, Chan Hee Song, Jiaman Wu, Shijie Chen, Hanane Nour Moussa, Tianshu Zhang, Jian Xie, Yifei Li, Tianci Xue, Zeyi Liao, Kai Zhang, Boyuan Zheng, Zhaowei Cai, Viktor Rozgic, Morteza Ziyadi, Huan Sun, and Yu Su. 2025. Mind2web 2: Evaluating agentic search with agent-as-a-judge. *arXiv preprint arXiv:2506.21506*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*.

Hao Guan, Lingyue Fu, Shao Zhang, Yaoming Zhu, Kangning Zhang, Lin Qiu, Xunliang Cai, Xuezhi Cao, Weiwen Liu, Weinan Zhang, and Yong Yu. 2026. SWE-Cycle: Benchmarking code agents across the complete issue resolution cycle. *Preprint*, arXiv:2605.13139.

Anmol Gulati, Hariom Gupta, Elias Lumer, Sahil Sen, and Vamse Kumar Subbiah. 2026. Ask early, ask late, ask right: When does clarification timing matter for long-horizon agents? *Preprint*, arXiv:2605.07937.

Jianyuan Guo, Zhiwei Hao, Chengcheng Wang, Cheng Fan, Tingzhang Luo, Hongguang Li, Ying Gao, Hefei Mei, Jiankun Peng, Rongjian Xu, Minjing Dong, Han Wu, Mengyu Zheng, Kai Han, Shiqi Wang, Chang Xu, and Yunhe Wang. 2026. From question answering to task completion: A survey on agent system and harness design. *Preprint*, arXiv:2606.20683.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 59532–59569.

Songhao Han, Boxiang Qiu, Yue Liao, Siyuan Huang, Chen Gao, Shuicheng Yan, and Si Liu. 2025. RoboCerebra: A large-scale benchmark for long-horizon robotic manipulation evaluation. In *Advances in Neural Information Processing Systems*, volume 38.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890.

William Hill, Ireton Liu, Anita De Mello Koch, Damion Harvey, Nishanth Kumar, George Konidaris, and Steven James. 2024. MinePlanner: A benchmark for long-horizon planning in large Minecraft worlds. In *Proceedings of the 6th ICAPS Workshop on the International Planning Competition (WIPC)*.

Jiseung Hong, Benjamin G. Ascoli, and Jinho D. Choi. 2026. ReCUBE: Evaluating repository-level context utilization in code generation. *Preprint*, arXiv:2603.25770.

Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. 2026. Single-rollout asynchronous optimization for agentic reinforcement learning. *arXiv preprint arXiv:2607.07508*.

Chen Hu, Haikuo Du, Heng Wang, Lin Lin, Mingrui Chen, Peng Liu, Ruihang Miao, Tianchi Yue, Wang You, Wei Ji, Wei Yuan, Wenjin Deng, Xiaojian Yuan, Xiaoyun Zhang, Xiangyu Liu, Xikai Liu, Yanming Xu, Yicheng Cao, Yifei Zhang, Yongyao Wang, Yubo Shu, Yurong Zhang, Yuxiang Zhang, Zheng Gong, Zhichao Chang, Binyan Li, Dan Ma, Furong Jia, Hongyuan Wang, Jiayu Liu, Jing Bai, Junlan Liu, Manjiao Liu, Na Wang, Qiuping Wu, Qinxin Du, Shiwei Li, Wen Sun, Yifeng Gong, Yonglin Chen, Yuling Zhao, Yuxuan Lin, Ziqi Ren, Zixuan Wang, Aihu Zhang, Brian Li, Buyun Ma, Kang An, Li Xie, Mingliang Li, Pan Li, Shidong Yang, Xi Chen, Xiaojia Liu, Yuchu Luo, Yuan Song, YuanHao Ding, Yuanwei Liang, Zexi Li, Zhaoning Zhang, Zixin Zhang, Binxing Jiao, Daxin Jiang, Jiansheng Chen, Jing Li, Xiangyu Zhang, and Yibo Zhu. 2025a. Step-DeepResearch technical report. *arXiv preprint arXiv:2512.20491*.

Ruida Hu, Xinchen Wang, Chao Peng, Cuiyun Gao, and David Lo. 2026. Evaluating LLM-Based 0-to-1 software generation in end-to-end CLI tool scenarios. *Preprint*, arXiv:2604.06742.

Shengran Hu, Cong Lu, and Jeff Clune. 2025b. Automated design of agentic systems. In *International Conference on Learning Representations*.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. 2025c. Memory in the age of AI agents. *arXiv preprint arXiv:2512.13564*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations*.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. 2026a. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks. *arXiv preprint arXiv:2607.07946*.

Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Xue Yang, Dongdong Chen, Xiaoqing Zheng, and Chong Luo. 2026b. From raw experience to skill consumption: A systematic study of model-generated agent skills. *Preprint*, arXiv:2605.23899.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. 2025. ALE-Bench: A benchmark for long-horizon objective-driven algorithm engineering. In *Advances in Neural Information Processing Systems*, volume 38.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2025. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pages 58791–58831.

Peter Jansen, Oyvind Tafjord, Marissa Radensky, Pao Siangliulue, Tom Hope, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Daniel S. Weld, and Peter Clark. 2025. CodeScientist: End-to-end semi-automated scientific discovery with code-based experimentation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 13370–13467.

Henry Jiang. 2026. Darwin: Dynamic agentically rewriting self-improving network. *Preprint*, arXiv:2602.05848.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. AIDE: AI-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157.

Qirui Jin, Lingching Tung, Kenan Li, Qiyang Shi, Yushi She, Huanzhong Jia, Harrison Zhao, Kejing Xia, Zhenbang Du, Yikai Zhang, Jiaxin Pei, Zhenyu Zhang, Zhen Qi, Yuyan Duan, Wenke Lee, and Zijian Jin. 2026a. ChainSWE: Benchmarking coding agents on multi-bug software maintenance. *arXiv preprint arXiv:2607.02606*.

Yiyang Jin, Kunzhao Xu, Hang Li, Xueting Han, Yanmin Zhou, Cheng Li, and Jing Bai. 2026b. ReVeal: Self-evolving code agents via reliable self-verification. In *The Fourteenth International Conference on Learning Representations*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. 2025. ACON: Optimizing context compression for long-horizon LLM agents. *arXiv preprint arXiv:2510.00615*.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. 2025a. OmniACT: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *Computer Vision – ECCV 2024*, volume 15126, pages 161–178. Springer.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, Juyong Lee, Tengjun Jin, Yifan Mai, Yifei Zhou, Yuxuan Zhu, Rishi Bommasani, Daniel Kang, Dawn Song, Peter Henderson, Yu Su, Percy Liang, and Arvind Narayanan. 2025b. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. *arXiv preprint arXiv:2510.11977*.

Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. 2026. Process reward models that think. *Transactions on Machine Learning Research*. J2C Certification.

Zaid Khan, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. 2025. DataEnvGym: Data generation agents in teacher environments with student feedback. In *International Conference on Learning Representations*.

Aaditya Khanal, Yangyang Tao, and Junxiu Zhou. 2026. Beyond pass@ 1: A reliability science framework for long-horizon LLM agents. *arXiv preprint arXiv:2603.29231*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling declarative language model calls into self-improving pipelines. In *The Twelfth International Conference on Learning Representations*.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. 2025. Towards a science of scaling agent systems. *Preprint*, arXiv:2512.08296.

Kimi Team. 2026a. Kimi K3: Open frontier intelligence. `https://www.kimi.com/blog/kimi-k3`. Product release, 16 July 2026; accessed 17 August 2026.

Kimi Team. 2026b. Kimi K3: Open frontier intelligence. `https://github.com/moonshotai/Kimi-K3`. *Preprint*, arXiv:2607.24653. Technical report; accessed 2026-07-28.

Peter Kirgis, Sayash Kapoor, Andrew Schwartz, Stephan Rabanser, David Africa, Konstantinos Voudouris, Viet Nguyen, Toby Pilditch, Magda Dubois, Harry Coppock, Cozmin Ududec, Nitya Nadgir, Matilda Orona, Tilman Bayer, Derrick Chan-Sew, Yue Ling, Abhishek Shetty, Helen Toner, Gillian Hadfield, Seth Lazar, Steve Newman, Shoshannah Tekofsky, Rishi Bommasani, and Arvind Narayanan. 2026. Can AI agents conduct open-ended AI research? early evidence from two case studies. *arXiv preprint arXiv:2607.27191*.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905.

Shirley Kokane, Ming Zhu, Tulika Manoj Awalgaonkar, Jianguo Zhang, Akshara Prabhakar, Thai Quoc Hoang, Zuxin Liu, Rithesh RN, Liangwei Yang, Weiran Yao, Juntao Tan, Zhiwei Liu, Shelby Heinecke, Huan Wang, Juan Carlos Niebles, Caiming Xiong, and Silvio Savarese. 2025. ToolScan: A benchmark for characterizing errors in tool-use LLMs. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

Ádám Kovács. 2026. Squeez: Task-conditioned tool-output pruning for coding agents. *arXiv preprint arXiv:2604.04979*.

Ilia Kulikov, Chenxi Whitehouse, Tianhao Wu, Yixin Nie, Swarnadeep Saha, Eryk Helenowski, Weizhe Yuan, Olga Golovneva, Jack Lanchantin, Yoram Bachrach, Jakob Foerster, Xian Li, Han Fang, Sainbayar Sukhbaatar, and Jason Weston. 2026. Autodata: An agentic data scientist to create high quality synthetic data. *arXiv preprint arXiv:2606.25996*.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel Ziegler, Elizabeth Barnes, and Lawrence Chan. 2025. Measuring AI ability to complete long software tasks. In *Advances in Neural Information Processing Systems*, volume 38, pages 92213–92266.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. 2026. LLMs get lost in multi-turn conversation. In *International Conference on Learning Representations*.

Man Ho Lam, Chaozheng Wang, Hange Liu, Jingyu Xiao, Haau sing Li, Jen tse Huang, Terry Yue Zhuo, and Michael R. Lyu. 2026. SWE-Chain: Benchmarking coding agents on chained release-level package upgrades. *Preprint*, arXiv:2605.14415.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. 2025. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*.

Tue Le, Minh V. T. Thai, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. 2025. SWE-EVO: Benchmarking coding agents in long-horizon software evolution scenarios. *Preprint*, arXiv:2512.18470.

Dongjun Lee, Juyong Lee, Kyuyoung Kim, Jihoon Tack, Jinwoo Shin, Yee Whye Teh, and Kimin Lee. 2025. Learning to contextualize web pages for enhanced decision making by LLM agents. In *The Thirteenth International Conference on Learning Representations*.

Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. 2026a. Recursive harness self-improvement. *arXiv preprint arXiv:2607.15524*.

Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. 2024. B-MoCA: Benchmarking mobile device control agents across diverse configurations. *arXiv preprint arXiv:2404.16660*.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. 2026b. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Silvio Savarese, Hyowon Gweon, Karen Liu, Jiajun Wu, and Li Fei-Fei. 2023. BEHAVIOR-1K: A benchmark for embodied AI with 1,000 everyday activities and realistic simulation. In *Proceedings of The 6th Conference on Robot Learning*, volume 205, pages 80–93. PMLR.

Chenliang Li, Adel Elmahdy, Alex Boyd, Zhongruo Wang, Siliang Zeng, Alfredo Garcia, Parminder Bhatia, Taha Kass-Hout, Cao Xiao, and Mingyi Hong. 2025a. Stabilizing off-policy training for long-horizon LLM agent via turn-level importance sampling and clipping-triggered normalization. *arXiv preprint arXiv:2511.20718*.

Dun Li, Jiatao Li, and Hongzhi Li. 2026a. From 0-to-1 to 1-to-n: Reproducible engineering evidence for MetaAI recursive self-design. *arXiv preprint arXiv:2606.09663*.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. 2026b. SALT: Step-level advantage assignment for long-horizon agents via trajectory graph. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 4709–4725.

Junjie Li, Xi Xiao, Yunbei Zhang, Chen Liu, Lin Zhao, Xiaoying Liao, Yingrui Ji, Janet Wang, Yingqiang Ge, Weijie Xu, Xi Fang, Xiang Xu, Tianchen Zhao, Youngeun Kim, Jihun Hamm, Tianyang Wang, and Chandan Reddy. 2026c. Agent harness engineering: A survey. `https://picrew.github.io/LLM-Harness/`. Seven-layer ETCLOVG taxonomy over 170+ open-source projects; accessed 2026-07-29.

Wanli Li, Bowen Zhou, Yunyao Yu, Zhou Xu, Yifan Yang, Dongsheng Li, and Caihua Shan. 2026d. WeaveBench: A long-horizon, real-world benchmark for computer-use agents with hybrid interfaces. *arXiv preprint arXiv:2606.09426*.

Xiaochuan Li, Ryan Ming, Meng Chu, Shuai Shao, Rong Jin, and Chenyan Xiong. 2026e. Acm: Agentic context management for long horizon tasks. *Preprint*, arXiv:2607.23809.

Yu Li, Chenyang Shao, Xinyang Liu, Ruotong Zhao, Peijie Liu, Hongyuan Su, Zhibin Chen, Qinglong Yang, Anjie Xu, Yi Fang, Qingbin Zeng, Tianxing Li, Jingbo Xu, Fengli Xu, Yong Li, and Tie-Yan Liu. 2026f. AutoSOTA: An end-to-end automated research system for state-of-the-art AI model discovery. *arXiv preprint arXiv:2604.05550*.

Yucheng Li, Frank Guerin, and Chenghua Lin. 2024. LatestEval: Addressing data contamination in language model evaluation through dynamic and time-sensitive test construction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18600–18607.

Yujiang Li, Zhenyu Hou, Yi Jing, Jie Tang, and Yuxiao Dong. 2026g. Compactionrl: Reinforcement learning with context compaction for long-horizon agents. *Preprint*, arXiv:2607.05378.

Zijian Li, Xin Guan, Bo Zhang, Shen Huang, Houquan Zhou, Shaopeng Lai, Ming Yan, Yong Jiang, Pengjun Xie, Fei Huang, Jun Zhang, and Jingren Zhou. 2025b. WebWeaver: Structuring web-scale evidence with dynamic outlines for open-ended deep research. *arXiv preprint arXiv:2509.13312*.

Gang Liao, Hongsen Qin, Ying Wang, Alicia Golden, Michael Kuchnik, Yavuz Yetim, Ruichao Xiao, Jia Jiunn Ang, Chunli Fu, Yihan He, Samuel Hsia, Zewei Jiang, Roman Levenstein, Dianshi Li, Liyuan Li, Ajit Mathews, Varna Puvvada, Feng Shi, Nathan Yan, Xiayu Yu, Uladzimir Pashkevich, Matt Steiner, Carole-Jean Wu, and Gaoxiang Liu. 2026. KernelEvolve: Scaling agentic kernel coding for heterogeneous AI accelerators at Meta. In *2026 53rd Annual International Symposium on Computer Architecture (ISCA)*, pages 733–747.

Tobias Lindenbauer, Igor Slinko, Ludwig Felder, Egor Bogomolov, and Yaroslav Zharov. 2025. The complexity trap: Simple observation masking is as efficient as LLM summarization for agent context management. *arXiv preprint arXiv:2508.21433*.

Jiacheng Liu, Xiaohan Zhao, Xinyi Shang, and Zhiqiang Shen. 2026a. Dive into Claude code: The design space of today's and future AI agent systems. *Preprint*, arXiv:2604.14228.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, volume 2024, pages 52989–53046.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Weinan E, and Siheng Chen. 2025. ML-Master: Towards AI-for-AI via integration of exploration and reasoning. *arXiv preprint arXiv:2506.16499*.

Zheyuan Liu, Liqiang Xiao, Yang Li, Hyokun Yun, Lihong Li, Chao Zhang, and Meng Jiang. 2026b. Do LLMs catch their own mistakes? a comprehensive benchmark for reflective tool use LLMs. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 1748–1773. Association for Computational Linguistics.

Ziyang Liu, Xinyan Guo, Xuchen Wei, Han Hao, and Liu Yang. 2026c. Escher-loop: Mutual evolution by closed-loop self-referential optimization. *Preprint*, arXiv:2604.23472.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. 2026a. Towards end-to-end automation of AI research. *Nature*, 651:914–919.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025a. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183. Association for Computational Linguistics.

Qingyu Lu, Liang Ding, Siyi Cao, Xuebo Liu, Kanjian Zhang, Jinxia Zhang, and Dacheng Tao. 2025b. Runaway is ashamed, but helpful: On the early-exit behavior of large language model-based agents in embodied environments. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 24014–24027, Suzhou, China. Association for Computational Linguistics.

Xing Han Lù, Zdenek Kasner, and Siva Reddy. 2024. WebLINX: Real-world website navigation with multi-turn dialogue. In *International Conference on Machine Learning*, volume 235, pages 33007–33056. PMLR.

Xinyu Lu, Tianshu Wang, Pengbo Wang, Zujie Wen, Zhiqiang Zhang, Jun Zhou, Boxi Cao, Yaojie Lu, Hongyu Lin, Xianpei Han, and Le Sun. 2026b. The meta-agent challenge: Are current agents capable of autonomous agent development? *arXiv preprint arXiv:2606.04455*.

Alisia Lupidi, Bhavul Gauri, Thomas Simon Foster, Bassel Al Omari, Despoina Magka, Alberto Pepe, Alexis Audran-Reiss, Muna Aghamelu, Nicolas Baldwin, Lucia Cipolina-Kun, Jean-Christophe Gagnon-Audet, Chee Hau Leow, Sandra Lefdal, Hossam Mossalam, Abhinav Moudgil, Saba Nazir, Emanuel Tewolde, Isabel Urrego, Jordi Armengol Estape, Amar Budhiraja, Gaurav Chaurasia, Abhishek Charnalia, Derek Dunfield, Karen Hambardzumyan, Daniel Izcovich, Martin Josifoski, Ishita Mediratta, Kelvin Niu, Parth Pathak, Michael Shvartsman, Edan Toledo, Anton Protopopov, Roberta Raileanu, Alexander Miller, Tatiana Shavrina, Jakob Foerster, and Yoram Bachrach. 2026. AIRS-Bench: A suite of tasks for frontier AI research science agents. *arXiv preprint arXiv:2602.06855*.

Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, Xinghan Li, Xinyang Han, Yicheng Zhang, Huaqing Zhang, Runhan Huang, Kaicheng Yang, Zitao Chen, Wentao Guo, Junlin Yang, Xinyue Ai, Wenhao Chai, Yadi Cao, Ziran Yang, Kun Wang, Dapeng Jiang, Huan-ang Gao, Shange Tang, Chengshuai Shi, Simon S. Du, Max Simchowitz, Jiantao Jiao, Dawn Song, and Chi Jin. 2026. Mls-bench: A holistic and rigorous assessment of ai systems on building better ai. *arXiv preprint arXiv:2605.08678*.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pages 74325–74362.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2026a. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. In *International Conference on Learning Representations*.

Zerun Ma, Guoqiang Wang, Xinchen Xie, Yicheng Chen, He Du, Bowen Li, Yanan Sun, Wenran Liu, Kai Chen, and Yining Li. 2026b. TREX: Automating LLM fine-tuning via agent-driven tree-based exploration. *arXiv preprint arXiv:2604.14116*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*.

Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. 2024. LLM critics help catch LLM bugs. *arXiv preprint arXiv:2407.00215*.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. 2025. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*.

Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. 2026a. RSIBench-Data: Benchmarking data-centric research for recursive self-improvement. *arXiv preprint arXiv:2607.25886*.

Qianyu Meng, Yanan Wang, Liyi Chen, Yihang Li, Wei Wu, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, and Yao Hu. 2026b. Agent harness for large language model agents: A survey. *Preprints*.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu,

Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *Preprint*, arXiv:2601.11868.

Elliot Meyerson, Giuseppe Paolo, Roberto Dailey, Hormoz Shahrzad, Olivier Francon, Conor F Hayes, Xin Qiu, Babak Hodjat, and Risto Miikkulainen. 2025. Solving a million-step LLM task with zero errors. *arXiv preprint arXiv:2511.09030*.

Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. 2025. Evaluation and benchmarking of LLM agents: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pages 6129–6139.

Hussein Mozannar, Gagan Bansal, Cheng Tan, Adam Fourney, Victor Dibia, Jingya Chen, Jack Gerrits, Tyler Payne, Matheus Kunzler Maldaner, Madeleine Grunde-McLaughlin, Eric Zhu, Griffin Bassman, Jacob Alber, Peter Chang, Ricky Loynd, Friederike Niedtner, Ece Kamar, Maya Murad, Rafah Hosn, and Saleema Amershi. 2025. Magentic-UI: Towards human-in-the-loop agentic systems. *Preprint*, arXiv:2507.22358.

Rabeya Khatun Muna, Md Nakhla Rafi, Tse-Hsun, and Chen. 2026. CI-Repair-Bench: A repository-aware benchmark for automated patch validation via CI workflows. *Preprint*, arXiv:2604.27148.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, Lingjie Chen, Yanjun Zhao, Ke Yang, Bingxuan Li, Cheng Qian, Gaotang Li, Xiao Lin, Zhichen Zeng, Ruizhong Qiu, Sirui Chen, Yifan Sun, Xiyuan Yang, Ruida Wang, Rui Pan, Chenyuan Yang, Dylan Zhang, Liri Fang, Zikun Cui, Yang Cao, Pan Chen, Dorothy Sun, Ren Chen, Mahesh Srinivasan, Nipun Mathur, Yinglong Xia, Hong Li, Hong Yan, Pan Lu, Lingming Zhang, Tong Zhang, Hanghang Tong, and Jingrui He. 2026. Code as agent harness. *Preprint*, arXiv:2605.18747.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*.

OpenAI. 2024. Introducing SWE-bench verified.

OpenAI. 2026. GPT-5.6: Frontier intelligence that scales with your ambition. `https://openai.com/index/gpt-5-6/`. Product release, 9 July 2026; accessed 17 August 2026.

Toby Ord. 2025. Is there a half-life for the success rates of AI agents? *arXiv preprint arXiv:2505.05115*.

Yuki Orimo, Iori Kurata, Hodaka Mori, Ryuhei Okuno, Ryohto Sawada, and Daisuke Okanohara. 2025. PARC: An autonomous self-reflective coding agent for robust execution of long-horizon tasks. *Preprint*, arXiv:2512.03549.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, Maohao Shen, Vishy Tirumalashetty, George Lee, Jiawei Han, Tomas Pfister, and Chen-Yu Lee. 2026. Skillos: Learning skill curation for self-evolving agents. *Preprint*, arXiv:2605.06614.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*.

Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. 2026. Natural-language agent harnesses. *Preprint*, arXiv:2603.25723.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. 2024. WebCanvas: Benchmarking web agents in online environments. In *ICML 2024 Workshop on Agentic Markets*.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The Berkeley Function Calling Leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392. PMLR.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565.

Zhiyuan Peng, Xin Yin, Pu Zhao, Fangkai Yang, Lu Wang, Ran Jia, Xu Chen, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2026. RepoGenesis: Benchmarking end-to-end microservice generation from readme to repository. *Preprint*, arXiv:2601.13943.

Bo Qiao, Liqun Li, Xu Zhang, Shilin He, Yu Kang, Chaoyun Zhang, Fangkai Yang, Hang Dong, Jue Zhang, Lu Wang, Minghua Ma, Pu Zhao, Si Qin, Xiaoting Qin, Chao Du, Yong Xu, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. 2023. Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Yaonan Qu and Meng Lu. 2026. Bilevel autoresearch: Meta-autoresearching itself. *arXiv preprint arXiv:2603.23420*.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. 2026. Towards a science of AI agent reliability. *arXiv preprint arXiv:2602.16666*.

Mohit Raghavendra, Soham Dan, Miguel Romero Calvo, Yannis Yiming He, Johannes Baptist Mols, Gautam Anand, Cole McCollum, Edgar Arakelyan, Vijay Bharadwaj, Andrew Park, Jeff Da, MohammadHossein Rezaei, Bing Liu, Brad Kenstler, and Yunzhong He. 2026. SWE Atlas: Benchmarking coding agents beyond issue resolution. *Preprint*, arXiv:2605.08366.

Amin Rakhsha, Thomas Hehn, Pietro Mazzaglia, Fabio Valerio Massoli, Arash Behboodi, and Tribhuvanesh Orekondy. 2026. LUMINA: Long-horizon understanding for multi-turn interactive agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 3913–3926. Association for Computational Linguistics.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. 2026. PostTrainBench: Can LLM agents automate LLM post-training? *Preprint*, arXiv:2603.08640.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. 2025. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, volume 2025, pages 406–441.

Jaideep Ray and Ankit Goyal. 2026. Structured feedback improves repair in an llm agent loop. *Preprint*, arXiv:2607.14167.

Zhe Ren, Yimeng Chen, Dandan Guo, Guowei Rong, Tonghui Li, R. B. Xiong, Qingfeng Lan, Wenyi Wang, Li Nanbo, Yibo Yang, Mingchen Zhuge, and Jürgen Schmidhuber. 2026. Self-improvements in modern agentic systems: A survey. *Preprint*, arXiv:2607.13104.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551.

Samuel Schmidgall and Michael Moor. 2025. AgentRxiv: Towards collaborative autonomous research. *arXiv preprint arXiv:2503.18102*.

Jürgen Schmidhuber. 2007. Gödel Machines: Fully self-referential optimal universal self-improvers. In Ben Goertzel and Cassio Pennachin, editors, *Artificial General Intelligence*, pages 199–226. Springer, Berlin, Heidelberg.

Ye Shang, Quanjun Zhang, Haichuan Hu, Chunrong Fang, Liang Xiao, and Zhenyu Chen. 2026. Breaking, stale, or missing? benchmarking coding agents on project-level test evolution. *Preprint*, arXiv:2605.06125.

Jiaqi Shao, Hanck Chen, Wei Zhang, Maxm Pan, and Bing Luo. 2026. Do agent benchmarks measure capability? protocol validity in the age of agentic ai. *arXiv preprint arXiv:2607.22368*.

Shuai Shao, Qihan Ren, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, Dongrui Liu, and Jing Shao. 2025. Your agent may misevolve: Emergent risks in self-evolving LLM agents. *arXiv preprint arXiv:2509.26354*.

Yonadav Shavit, Sandhini Agarwal, Miles Brundage, Steven Adler, Cullen O'Keefe, Rosie Campbell, Teddy Lee, Pamela Mishkin, Tyna Eloundou, Alan Hickey, Katarina Slama, Lama Ahmad, Paul McMillan, Alex Beutel, Alexandre Passos, and David G. Robinson. 2023. Practices for governing agentic AI systems. *Research Paper, OpenAI*.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR.

Yuling Shi, Yichun Qian, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. 2025. LongCodeZip: Compress long context for code language models. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 141–153.

Yuling Shi, Jinghan Xu, Kelin Fu, Wenhao Zeng, Shilin He, Lei Zhang, Yue Liu, Zelin Zhao, Terry Yue Zhuo, Jialun Cao, Siyu Ye, Tianyu Liu, Kai Cai, Shing-Chi Cheung, and Xiaodong Gu. 2026. SWE-Bench ProMax: Benchmarking agents on large-scale multilingual code refactoring. *Preprint*, arXiv:2608.09802.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. 2024. AI models collapse when trained on recursively generated data. *Nature*, 631:755–759.

Chenglei Si, Zitong Yang, Yejin Choi, Emmanuel J. Candès, Diyi Yang, and Tatsunori Hashimoto. 2026. Towards execution-grounded automated AI research. *arXiv preprint arXiv:2601.14525*.

Akshey Sigdel and Rista Baral. 2026. Schema first tool apis for llm agents: A controlled study of tool misuse, recovery, and budgeted performance. *Preprint*, arXiv:2603.13404.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. 2025. The illusion of diminishing returns: Measuring long horizon execution in LLMs. *arXiv preprint arXiv:2509.09677*.

SpaceXAI. 2026. Introducing Grok 4.6. `https://x.ai/news/grok-4-6`. Product release, 12 August 2026; accessed 17 August 2026.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. 2025. PaperBench: Evaluating AI's ability to replicate AI research. *arXiv preprint arXiv:2504.01848*.

Jason Starace. 2026. Scaffold effects on gaia: A controlled comparison. *Preprint*, arXiv:2606.08529.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2024. Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 58202–58245.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2025. Scaling long-horizon LLM agent via context-folding. *arXiv preprint arXiv:2510.11967*.

Chris Sypherd and Vaishak Belle. 2024. Practical considerations for agentic LLM systems. *Preprint*, arXiv:2412.04093.

Anh Ta, Junjie Zhu, and Shahin Shayandeh. 2026. Reinforced agent: Inference-time feedback for tool-calling agents. In *Proceedings of the Fifth Workshop on Generation, Evaluation and Metrics (GEM)*, pages 136–147, San Diego, California, USA. Association for Computational Linguistics.

Hui-Ze Tan, Xiao-Wen Yang, Hao Chen, Jie-Jing Shao, Yi Wen, Yuteng Shen, Weihong Luo, Xiku Du, Lan-Zhe Guo, and Yu-Feng Li. 2026. Hindsight credit assignment for long-horizon LLM agents. *arXiv preprint arXiv:2603.08754*.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. 2025. AI-Researcher: Autonomous scientific innovation. In *Advances in Neural Information Processing Systems*.

Qiong Tang, Tianxiang Sun, Xiangkun Hu, Xiangyang Liu, Yiran Chen, and Yunfan Shao. 2026. FARS: A fully automated research system deployed at scale. *arXiv preprint arXiv:2606.31651*.

Natalia Tarasova, Enrique Balp-Straffon, Aleksei Iancheruk, Yevhenii Sielskyi, Nikita Kozodoi, Liam H. Byrne, Jack Butler, Dayuan Jiang, Marcin Czelej, Andrew Ang, Yash Shah, Roi Blanco, and Sergei Ivanov. 2026. SWE-InfraBench: Evaluating language models on cloud infrastructure code. *Preprint*, arXiv:2606.05249.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076.

Varun Ursekar, Apaar Shanker, Yash Maurya, Shehab Yasser, Vijay S Kalmath, Veronica Chatrath, and Yuan Xue. 2026. Harnessopt-bench: Evaluating llms at harness optimization. *arXiv preprint arXiv:2608.06301*.

Sri Vatsa Vuddanti, Aarav Shah, Satwik Kumar Chittiprolu, Tony Song, Sunishchal Dev, Kevin Zhu, and Maheep Chaudhary. 2025. PALADIN: Self-correcting language model agents to cure tool-failure cases. *arXiv preprint arXiv:2509.25238*.

Andrew Wang, Sophia Hager, Adi Asija, Daniel Khashabi, and Nicholas Andrews. 2025a. Hell or high water: Evaluating agentic recovery from external failures. In *Second Conference on Language Modeling*.

Binghai Wang, Chenlong Zhang, Dayiheng Liu, Jiajun Zhang, Jiawei Chen, Mingze Li, Mouxiang Chen, Rongyao Fang, Siyuan Zhang, Xuwu Wang, Yuheng Jing, Zeyao Ma, and Zeyu Cui. 2026a. The verification horizon: No silver bullet for coding agent rewards. *Preprint*, arXiv:2606.26300.

Daoyu Wang, Qingchuan Li, Mingyue Cheng, Jie Ouyang, Shuo Yu, Chunli Liu, Shijin Wang, Qi Liu, and Enhong Chen. 2026b. CAPO: Critic-guided action-aligned policy optimization for advancing LLM agent capabilities. *arXiv preprint arXiv:2604.18401*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Haoran Wang, Zhenyu Hou, Yao Wei, Jie Tang, and Yuxiao Dong. 2025b. SWE-Dev: Building software engineering agents with training and inference scaling. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3742–3761, Vienna, Austria. Association for Computational Linguistics.

Weixuan Wang, Dongge Han, Daniel Madrigal Diaz, Jin Xu, Victor Rühle, and Saravan Rajmohan. 2025c. OdysseyBench: Evaluating LLM agents on long-horizon complex office application workflows. *arXiv preprint arXiv:2508.09124*.

Wenkai Wang, Tao Xiong, Jingchen Ni, Yunpeng Bao, Xiyun Li, Tianqi Liu, Hongcan Guo, Zilong Huang, and Shengyu Zhang. 2026c. DeskCraft: Benchmarking desktop agents on professional workflows and human-in-the-loop collaboration. *arXiv preprint arXiv:2606.03103*.

Wenxiao Wang, Priyatham Kattakinda, and Soheil Feizi. 2026d. Do agent optimizers compound? a continual-learning evaluation on terminal-bench 2.0. *arXiv preprint arXiv:2607.14004*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. 2025d. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pages 65882–65919.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024b. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, volume 2024, pages 32593–32627.

Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D Nowak. 2026e. The long-horizon task mirage? diagnosing where and why agentic systems break. *arXiv preprint arXiv:2604.11978*.

Yifei Wang, Ziteng Wang, Yuling Shi, Silin Chen, Xinrui Wang, Yueqi Wang, Beijun Shen, Linjing Li, Xiaodong Gu, Julian McAuley, and Daniel Dajun Zeng. 2026f. Context compression for LLM agents: A survey of methods, failure modes, and evaluation. *Preprints*.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. 2026g. Rethinking the evaluation of harness evolution for agents. *arXiv preprint arXiv:2607.12227*.

Yuhang Wang, Yuling Shi, Mo Yang, Rongrui Zhang, Shilin He, Heng Lian, Yuting Chen, Siyu Ye, Kai Cai, and Xiaodong Gu. 2026h. SWE-Pruner: Self-adaptive context pruning for coding agents. *arXiv preprint arXiv:2601.16746*.

Yuru Wang, Lejun Cheng, Yuxin Zuo, Sihang Zeng, Bingxiang He, Che Jiang, Junlin Yang, Yuchong Wang, Kaikai Zhao, Weifeng Huang, Kai Tian, Zhenzhao Yuan, Jincheng Zhong, Weizhi Wang, Ning Ding, Bowen Zhou, and Kaiyan Zhang. 2026i. NatureBench: Can coding agents match the published SOTA of nature-family papers? *arXiv preprint arXiv:2606.24530*.

Zefeng Wang, Minxi Yan, Jinhe Bi, Sikuan Yan, Volker Tresp, and Yunpu Ma. 2026j. Metaskill-evolve: Recursive self-improvement of llm agents via two-timescale meta-skill evolution. *Preprint*, arXiv:2607.05297.

Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025e. Agent workflow memory. In *Forty-second International Conference on Machine Learning*.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. 2025f. JARVIS-1: Open-World Multi-Task Agents With Memory-Augmented Multimodal Language Models . *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 47(03):1894–1907.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. 2024c. OfficeBench: Benchmarking language agents across multiple applications for office automation. *arXiv preprint arXiv:2407.19056*.

Weco AI. 2026. First evidence of recursive self-improvement. First-party technical blog. Accessed 4 August 2026; full technical report and protocol were not available at the corpus freeze.

Hu Wei. 2026. From agent loops to structured graphs:a scheduler-theoretic framework for LLM agent execution. *Preprint*, arXiv:2604.11378.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.

Yixuan Weng, Minjun Zhu, Qiujie Xie, Qiyao Sun, Zhen Lin, Sifan Liu, and Yue Zhang. 2026. DeepScientist: Advancing frontier-pushing scientific findings progressively. In *International Conference on Learning Representations*.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Benjamin Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025. LiveBench: A challenging, contamination-limited LLM benchmark. In *International Conference on Learning Representations*.

Hjalmar Wijk, Tao Roa Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Joshua M Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Jun Koba Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. 2025. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. In *International Conference on Machine Learning*, volume 267, pages 66772–66832. PMLR.

Xiaoqing Wu, Xingyu Fan, Feifei Li, and Wenhui Que. 2026a. When history lies: Evaluating and improving tool use under misleading multi-turn histories. *arXiv preprint arXiv:2608.06057*.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. 2025. ReSum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*.

Yating Wu, Yuhao Zhang, Sayan Ghosh, Sourya Basu, Anoop Deoras, Jun Huan, and Gaurav Gupta. 2026b. ContextWeaver: Selective and dependency-structured memory construction for LLM agents. *arXiv preprint arXiv:2604.23069*.

Yong Wu, YanZhao Zheng, TianZe Xu, ZhenTao Zhang, YuanQiang Yu, JiHuai Zhu, Chao Ma, BinBin Lin, BaoHua Dong, HangCheng Zhu, RuoHui Huang, and Gang Yu. 2026c. ContextBudget: Budget-aware context management for long-horizon search agents. *arXiv preprint arXiv:2604.01664*.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. 2025. AgentGym-RL: Training LLM agents for long-horizon decision making through multi-turn reinforcement learning. *arXiv preprint arXiv:2509.08755*.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. 2025. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE).

Yuan-An Xiao, Pengfei Gao, Chao Peng, and Yingfei Xiong. 2026. Reducing cost of LLM agents with trajectory reduction. *Proceedings of the ACM on Software Engineering*, 3(FSE):1241–1263.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024a. TravelPlanner: A benchmark for real-world planning with language agents. In *International Conference on Machine Learning*, volume 235, pages 54590–54613. PMLR.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024b. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pages 52040–52094.

Cheng Xu, Shuhao Guan, Derek Greene, and M-Tahar Kechadi. 2024. Benchmark data contamination of large language models: A survey. *arXiv preprint arXiv:2406.04244*.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. 2025a. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, volume 38.

Huiyu Xu, Zhibo Wang, Wenhui Zhang, Ziqi Zhu, Yaopeng Wang, Kui Ren, and Chun Chen. 2026a. LoopTrap: Termination poisoning attacks on LLM agents. *Preprint*, arXiv:2605.05846.

Tianshi Xu, Huifeng Wen, and Meng Li. 2026b. Adapting the interface, not the model: Runtime harness adaptation for deterministic llm agents. *Preprint*, arXiv:2605.22166.

Weixian Xu, Tiantian Mi, Yixiu Liu, Yang Nan, Zhimeng Zhou, Lyumanshan Ye, Lin Zhang, Yu Qiao, and Pengfei Liu. 2026c. ASI-Evolve: AI accelerates AI. *arXiv preprint arXiv:2603.29640*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025b. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, pages 17577–17604.

Xinbo Xu, Ruihan Yang, Haiyang Shen, Wendong Xu, Bofei Gao, Ruoyu Wu, Kean Shi, Weichu Xie, Xuanzhong Chen, Ming Wu, Jason Zeng, Michael Heinrich, Elvis Zhang, Liang Chen, Kuan Li, and Baobao Chang. 2026d. RoadmapBench: Evaluating long-horizon agentic software development across version upgrades. *Preprint*, arXiv:2605.15846.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. 2025. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*.

Lu Yan, Xuan Chen, and Xiangyu Zhang. 2026. When the specification emerges: Benchmarking faithfulness loss in long-horizon coding agents. *arXiv preprint arXiv:2603.17104*.

Chengyuan Yang, Zequn Sun, Wei Wei, and Wei Hu. 2026a. Beyond static summarization: Proactive memory extraction for LLM agents. *arXiv preprint arXiv:2601.04463*.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. 2026b. ProgramBench: Can language models rebuild programs from scratch? *Preprint*, arXiv:2605.03546.

Junlin Yang, Che Jiang, Yu Fu, Tianwei Luo, Can Ren, Weizhi Wang, Kaikai Zhao, Hongyi Liu, Yuxin Zuo, Yuru Wang, Yuchen Fan, Kai Tian, Zhenzhao Yuan, Xiaojian Lin, Li Sheng, Rushi Qiang, Guoli Jia, Xingtai Lv, Ermo Hua, Dianqiao Lei, Youbang Sun, Ning Ding, Bowen Zhou, and Kaiyan Zhang. 2026c. Frontis-MA1: Training an AI4AI model towards recursive self-improvement in machine learning engineering. *arXiv preprint arXiv:2607.28568*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2025. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Yilun Yao, Shan Huang, Elsie Dai, Zhewen Tan, Zhenyu Duan, Shousheng Jia, Yanbing Jiang, and Tong Yang. 2026. ARC: Active and reflection-driven context management for long-horizon information seeking agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 18644–18659. Association for Computational Linguistics.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Siheng Chen, Jingren Zhou, and Yong Jiang. 2025. AgentFold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2026. A survey on evaluation of LLM-based agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 26690–26714.

Lu Yi, Runlin Lei, Liuyi Yao, Yuexiang Xie, Yuyang Li, Wenhao Zhang, Zhewei Wei, Yaliang Li, and Jian-Yun Nie. 2026. Learning agent-compatible context management for long-horizon tasks. *Preprint*, arXiv:2605.30785.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. 2024. Gödel Agent: A self-referential agent framework for recursive self-improvement. *arXiv preprint arXiv:2410.04444*.

Ye Yu, Xiaopeng Yuan, Haibo Jin, Heming Liu, Yaoning Yu, and Haohan Wang. 2026. Do self-evolving agents forget? capability degradation and preservation in lifelong LLM agent adaptation. *arXiv preprint arXiv:2605.09315*.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, Dunjie Lu, Haikong Lu, Qi Zhen, Xinyuan Wang, Jiaqi Deng, Yuhao Yang, Cheng Chen, Boyuan Zheng, Alex Su, Xiao Yu, Hao Zou, Saaket Agashe, Xing Han Lu, Manpreet Kaur, Zhengyang Qi, Vincent Sunn Chen, Frederic Sala, Dayiheng Liu, Junyang Lin, Zhou Yu, Yu Su, Siva Reddy, Xin Eric Wang, Peng Qi, Tianbao Xie, and Tao Yu. 2026. OSWorld 2.0: Benchmarking computer use agents on long-horizon real-world tasks. *Preprint*, arXiv:2606.29537.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*.

Z.ai. 2026. GLM-5.3: Frontier coding with emergent cyber capabilities. `https://z.ai/blog/glm-5.3`. Product release, 14 August 2026; accessed 17 August 2026.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2024. Self-taught optimizer (STOP): Recursively self-improving code generation. In *Conference on Language Modeling*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pages 15476–15488.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. 2026a. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2025a. Darwin Gödel Machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. 2025b. SWE-bench goes live! In *Advances in Neural Information Processing Systems*, volume 38.

Linghao Zhang, Junhao Wang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Jiaheng Wen, Chengxing Xie, Maoquan Wang, Yufan Huang, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2025c. DI-BENCH: Benchmarking large language models on dependency inference with testable repositories at scale. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10134–10153.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2025d. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

Shaoqiu Zhang, Yuhang Wang, Jialiang Liang, Yuling Shi, Wenhao Zeng, Maoquan Wang, Shilin He, Ningyuan Xu, Siyu Ye, Kai Cai, and Xiaodong Gu. 2026b. SWE-Explore: Benchmarking how coding agents explore repositories. *Preprint*, arXiv:2606.07297.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. 2026c. DeepPlanning: Benchmarking long-horizon agentic planning with verifiable constraints. *arXiv preprint arXiv:2601.18137*.

Yunbei Zhang, Janet Wang, Yingqiang Ge, Weijie Xu, Jihun Hamm, and Chandan K. Reddy. 2026d. Stop comparing LLM agents without disclosing the harness. In *Second Workshop on Agents in the Wild: Safety, Security, and Beyond*.

Zehua Zhang, Ati Priya Bajaj, Divij Handa, Siyu Liu, Arvind S Raj, Hongkai Chen, Hulin Wang, Yibo Liu, Zion Leonahenahe Basque, Souradip Nath, Vishal Juneja, Nikhil Chapre, Yan Shoshitaishvili, Adam Doupé, Chitta Baral, and Ruoyu Wang. 2025e. BuildBench: Benchmarking LLM agents on compiling real-world open-source software. *Preprint*, arXiv:2509.25248.

Zhengxin Zhang, Ning Wang, Sainyam Galhotra, and Claire Cardie. 2026e. How far are we from true auto-research? *arXiv preprint arXiv:2605.19156*.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2024. Agent-SafetyBench: Evaluating the safety of LLM agents. *arXiv preprint arXiv:2412.14470*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Bingchen Zhao, Dhruv Srikanth, Yuxiang Wu, and Zhengyao Jiang. 2026. SpecBench: Measuring reward hacking in long-horizon coding agents. *Preprint*, arXiv:2605.21384.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. 2024. NATURAL PLAN: Benchmarking LLMs on natural language planning. *arXiv preprint arXiv:2406.04520*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024a. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning*, volume 235, pages 62138–62160. PMLR.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, Jianghao Lin, and Weinan Zhang. 2026a. Externalization in LLM agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*.

Qixing Zhou, Jiacheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, Dandan Tu, and Zhaoxiang Zhang. 2026b. FeatureBench: Benchmarking agentic coding for complex feature development. *Preprint*, arXiv:2602.10975.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024b. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

Wei Zhou, Xuanhe Zhou, Shaokun Han, Hongming Xu, Guoliang Li, Zhiyu Li, Feiyu Xiong, and Fan Wu. 2026c. Are we ready for an agent-native memory system? *Preprint*, arXiv:2606.24775.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. 2025. Where LLM agents fail and how they can learn from failures. *Preprint*, arXiv:2509.25370.

Xinyu Zhu, Yuzhu Cai, Zexi Liu, Bingyang Zheng, Cheng Wang, Rui Ye, Yuzhi Zhang, Linfeng Zhang, Weinan E, Siheng Chen, and Yanfeng Wang. 2026. Toward ultra-long-horizon agentic science: Cognitive accumulation for machine learning engineering. *arXiv preprint arXiv:2601.10402*.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. 2025. Agent-as-a-judge: Evaluate agents with agents. In *Forty-second International Conference on Machine Learning*, volume 267, pages 80569–80611.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. 2025. Self-adapting language models. In *Advances in Neural Information Processing Systems*.

A Survey Methods and Coverage

A.1 Search Process and Inclusion Criteria

We froze the systematic search on 4 August 2026 and verified targeted additions through 13 August 2026. We also froze the primary-source stage-ownership audit on 4 August 2026. The original Google Scholar search combined four terms: *long-horizon*, *LLM agent*, *reliability*, and *benchmark*. We paired these terms with terms for web use, computer use, software engineering, tool use, and planning. For the AI for AI (AI4AI) revision, we defined AI4AI as using AI to improve an AI artifact or research process. We queried the arXiv and OpenAlex APIs through eight recorded query families. These families covered AI4AI terminology, automated research, AI Scientist/autoresearch systems, self-improving and recursive agents, harness evolution, ML engineering, research benchmarks, and long-horizon reliability. Each query returned at most 50 records. We then found related records by following references and citations. The search returned 369 records after removing duplicate titles. One OpenAlex query timed out. Its corresponding arXiv query and all nine priority checks completed. Search results were capped, and one earlier log is incomplete. We therefore report how many records we screened, but not the total required for a full PRISMA flow diagram.

Core studies of long-horizon execution had to evaluate an interactive LLM agent. Eligibility required functional coupling across actions or rounds. This coupling means that earlier actions or feedback affect later actions or rounds and thereby influence success. Persistent state, delayed consequences, trajectory-level verification, and repeated-run reliability supported eligibility but did not replace functional coupling. AI4AI entries also required a repeated plan–execute–feedback–repair process that changed an AI artifact. Eligible artifacts included data, weights, a harness, an executable program, an evaluator, or a research process. We included self-improvement and recursive self-improvement studies that changed the mechanism producing later attempts or measured repeated improvement. We label non-coupled and non-AI-target comparisons as protocol-conditional or adjacent. Protocol-conditional entries depend on specific evaluation settings, whereas adjacent entries are included only for comparison. We excluded static QA, standard long-context processing, independent turns, one-call tools, and research proposals without execution or measured evidence from artifacts. We prioritized primary sources that disclosed tasks, environments, graders, budgets, or protocols over surveys.

A.2 Evidence Collection and Verification

For each component benchmark, we recorded its domain, scale, reported execution amount, coupling, persistent state, correctness signal, budget, and source. We kept each execution amount in the unit used

by its source. For the stage-ownership audit, we selected 35 primary publications and first-party reports covering AI4AI and related forms of self-adaptation. Each source contributes one record for a reported system release and identifies the main AI component that release could change. We coded the improvement target (D1), stage ownership (D3), and improvement-evidence profile (D5). Stage ownership records who controls the goal, plan, execution, feedback, and repair. The improvement-evidence profile ($G/R/H/T$) records measured gain, retention, a budget-matched human comparison, and held-out transfer. We also recorded the source type, where the evidence appears, why we assigned each code, and our confidence. We described self-reference (D2) and signal grounding (D4) in prose instead of assigning table codes. Self-reference asks whether the system running the improvement process is also a target of that process. Signal grounding identifies the evidence used to decide whether a change is an improvement. A human-created artifact fixed before the run affects only the stage for which it provides the main output or pass/fail rule. An external evaluator changes another stage's label only when that stage uses its feedback. We resolved disputed fields through a primary coding pass and an independent check of sources and possible contradictions. The only record based solely on a blog remains classified as low confidence. Because the audit lacks two completed human annotations, its counts describe this sample rather than how common the coded features are across the field.

We checked new records from logged searches and citation chaining against primary papers, proceedings, arXiv or publisher metadata, and first-party reports. We assign commercial material to a separate source tier and do not treat it as peer reviewed. Figure 7 summarizes the discovery, screening, extraction, coding, and verification process.

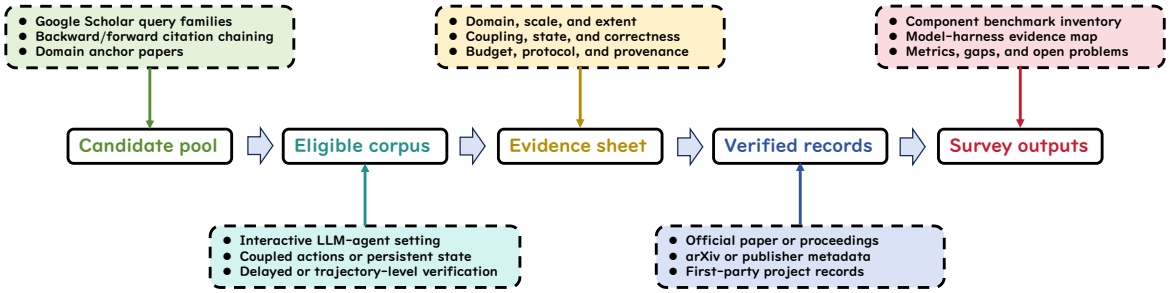


Figure 7: Survey methods and evidence trail. The pipeline combines the original Google Scholar search with the revised arXiv and OpenAlex search. It then screens records, extracts evidence in a standard format, codes stage ownership, and checks primary sources. We froze the systematic search on 4 August 2026 and verified targeted additions through 13 August 2026. The primary-source audit remained frozen on 4 August 2026. The component inventory contains 67 entries, and the stage-ownership audit contains 35 primary-source records. Because search results were capped, these counts are not the complete totals required for a PRISMA flow diagram.

A.3 Scope and Limitations

The corpus covers web and computer use, software and terminal work, tools, APIs, workplace workflows, planning, science, and model learning. It also covers AI-directed improvement of artifacts and research processes, harness evolution, and evaluation methods. Appendix E contains the 67 component rows. Appendix B reports the 35-record primary-source audit. These two views overlap because PostTrainBench reports both the scale of one benchmark component and evidence about an integrated system.

We gave more weight to stronger evidence but did not aim for exhaustive coverage. Four factors limit completeness: query caps, one recorded OpenAlex timeout, changing terminology, and the rapid publication of 2026 preprints. The documented search is therefore reproducible but incomplete. Stage-ownership labels may also depend on what we count as one system. For example, a reported release may operate autonomously at run time while a human still sets its goal during design. Finally, the stage-ownership audit has two independent coding passes but does not include two completed human annotations. We retain low-confidence and non-peer-reviewed sources for transparency. However, we exclude them from strong claims about recursive self-improvement.

B AI4AI Stage Ownership and Evidence of Improvement

We audited 35 primary publications and first-party reports. The sources cover AI for AI (AI4AI), self-improving systems, automated research, self-modifying agents, and related work on self-adaptation. Each source contributes one record for one reported system release. We code the main layer that the system can change. The audit corpus was frozen on 2026-08-04. Source verification was last updated on 2026-08-13. We selected sources for the evidence they provide, so this sample does not show how common these features are across the field. We include Self-Refine and Reflexion as related comparison systems. The audit directly codes the improvement target (D1), stage ownership (D3), and evidence of improvement (D5). We describe self-reference (D2) and signal grounding (D4) in prose instead.

How we assign stage ownership. HUM means that a human, or a human-created artifact fixed before the run, provides the stage's main output. The system only passes on or executes that output. MIX means that a human-written objective, test, template, or rule partly determines the stage, while the system creates the rest. SYS means that the system decides and carries out the stage from earlier inputs and general interface rules. A fixed external evaluator changes another stage's label only when that stage uses its feedback. Records use the year of first public release. Source and venue years remain in the bibliography.

How we record evidence of improvement. D5 records four separate checks, labeled G, R, H, and T. They are not combined into one score or ranking. G means that a change made by the system produced a measured gain. R means that a declared test confirmed retention, or that performance improved further across passes or generations. H means that the system was compared with a human under the same budget. T means that improvement transferred beyond the optimized metric, task, model, or domain under a controlled budget. Each result is coded as Y (demonstrated), N (tested but not demonstrated), or NR (not reported or not tested). Passing one check does not imply passing another.

Table 6: Who produces the main output at each stage in the 35 audited records. The counts describe this sample, not the whole field.

Stage	Human	Mixed	System
Goal	35	0	0
Plan	3	16	16
Execute	0	0	35
Feedback	16	19	0
Repair	0	12	23

All 35 records classify execution as system-owned. Under our design-time convention, no record classifies the final improvement goal or feedback as fully system-owned. Repair is system-owned in 23 records. These counts support a limited claim about this sample. The systems control execution and repair more often than they control the objective and criterion that define improvement.

Table 7: Evidence for improvement in the same 35 records. Y means demonstrated, N means tested but not demonstrated, and NR means not reported or not tested.

Evidence test	Y	N	NR
G: Measured gain	33	1	1
R: Retention / compounding	31	1	3
H: Budget-matched human comparison	1	0	34
T: Held-out transfer	14	1	20

Table 8: Coding details for the 35 systems. Each row identifies the main AI component being changed. It then shows who controls the goal, plan, execution, feedback, and repair stages. The row also reports the four evidence codes ($G/R/H/T$) and the primary source. The counts apply only to this sample.

System	AI component	Goal	Plan	Exec.	Feedb.	Repair	G/R/H/T	Source
STaR (Zelikman et al., 2022)	weights	HUM	HUM	SYS	HUM	MIX	Y/Y/NR/Y	peer-reviewed
Self-Rewarding Language Models (Yuan et al., 2024)	weights	HUM	MIX	SYS	MIX	MIX	Y/Y/NR/Y	peer-reviewed
Self-Refine (Madaan et al., 2023)	task output (adjacent)	HUM	HUM	SYS	MIX	MIX	Y/Y/NR/NR	peer-reviewed
Reflexion (Shinn et al., 2023)	agent memory (adjacent)	HUM	MIX	SYS	MIX	MIX	Y/Y/NR/NR	peer-reviewed
ExpeL (Zhao et al., 2024)	harness	HUM	MIX	SYS	HUM	MIX	Y/Y/NR/Y	peer-reviewed
SPIN (Chen et al., 2024)	weights	HUM	HUM	SYS	HUM	MIX	Y/Y/NR/NR	peer-reviewed
SEAL (Zweiger et al., 2025)	weights	HUM	MIX	SYS	HUM	MIX	Y/Y/NR/Y	peer-reviewed
DataEnvGym (Khan et al., 2025)	data/tasks	HUM	MIX	SYS	MIX	MIX	Y/Y/NR/NR	peer-reviewed
Autodata (Kulikov et al., 2026)	data/tasks	HUM	MIX	SYS	MIX	MIX	Y/Y/NR/Y	preprint
ASI-Evolve (Xu et al., 2026c)	research process	HUM	MIX	SYS	MIX	MIX	Y/Y/NR/Y	preprint
Frontis-MA1 (Yang et al., 2026c)	program/ infrastructure	HUM	MIX	SYS	HUM	MIX	Y/Y/NR/Y	technical
STOP (Zelikman et al., 2024)	harness	HUM	SYS	SYS	HUM	SYS	Y/Y/NR/Y	peer-reviewed
Gödel Agent (Yin et al., 2024)	harness	HUM	SYS	SYS	HUM	SYS	Y/Y/NR/NR	peer-reviewed
Automated Design of Agentic Systems (Hu et al., 2025b)	harness	HUM	SYS	SYS	HUM	SYS	Y/Y/NR/NR	peer-reviewed
Darwin Gödel Machine (Zhang et al., 2025a)	harness	HUM	SYS	SYS	HUM	SYS	Y/Y/NR/NR	peer-reviewed
AlphaEvolve (Novikov et al., 2025)	program/ infrastructure	HUM	SYS	SYS	HUM	SYS	Y/Y/NR/Y	technical
ShinkaEvolve (Lange et al., 2025)	program/ infrastructure	HUM	SYS	SYS	HUM	SYS	Y/Y/NR/Y	peer-reviewed
Recursive Harness Self-Improvement (Lee et al., 2026a)	harness	HUM	SYS	SYS	MIX	SYS	Y/Y/NR/NR	preprint
Meta-Harness (Lee et al., 2026b)	harness	HUM	SYS	SYS	HUM	SYS	Y/Y/NR/Y	preprint
Self-Harness (Zhang et al., 2026a)	harness	HUM	SYS	SYS	MIX	SYS	Y/Y/NR/NR	preprint
KernelEvolve (Liao et al., 2026)	program/ infrastructure	HUM	SYS	SYS	MIX	SYS	Y/Y/NR/N	peer-reviewed
Bilevel Autoresearch (Qu and Lu, 2026)	research process	HUM	SYS	SYS	HUM	SYS	Y/NR/NR/NR	preprint
AIDE^2 (Weco AI, 2026)	research process	HUM	SYS	SYS	HUM	SYS	Y/Y/NR/Y	first-party
AIDE (Jiang et al., 2025)	data/tasks	HUM	MIX	SYS	HUM	SYS	Y/Y/Y/NR	preprint
ML-Master (Liu et al., 2025)	data/tasks	HUM	MIX	SYS	HUM	SYS	Y/Y/NR/NR	preprint
ML-Master 2.0 (Zhu et al., 2026)	data/tasks	HUM	SYS	SYS	MIX	SYS	Y/Y/NR/Y	preprint
The AI Scientist v1 (Lu et al., 2024)	research process	HUM	MIX	SYS	MIX	SYS	Y/Y/NR/NR	preprint

System	AI component	Goal	Plan	Exec.	Feedb.	Repair	G/R/H/T	Source
The AI Scientist-v2 (Yamada et al., 2025)	research process	HUM	MIX	SYS	MIX	SYS	N/NR/NR/NR	preprint
AiScientist (Chen et al., 2026a)	program/ infrastructure	HUM	SYS	SYS	MIX	SYS	Y/Y/NR/NR	preprint
FARS (Tang et al., 2026)	research process	HUM	MIX	SYS	MIX	SYS	Y/N/NR/Y	preprint
AutoSOTA (Li et al., 2026f)	program/ infrastructure	HUM	MIX	SYS	MIX	SYS	Y/Y/NR/NR	preprint
AgentRxiv (Schmidgall and Moor, 2025)	harness	HUM	SYS	SYS	MIX	SYS	Y/Y/NR/NR	preprint
DeepScientist (Weng et al., 2026)	program/ infrastructure	HUM	SYS	SYS	MIX	MIX	Y/Y/NR/NR	peer-reviewed
AI-Researcher (Tang et al., 2025)	research process	HUM	MIX	SYS	MIX	SYS	NR/NR/NR/NR	peer-reviewed
CodeScientist (Jansen et al., 2025)	program/ infrastructure	HUM	MIX	SYS	MIX	SYS	Y/Y/NR/NR	peer-reviewed

C Detailed Evaluation Protocol

This appendix gives the reporting requirements summarized in Sections 2.3 and 3.1. A protocol-conditioned reliability surface shows how often a fixed system succeeds as tasks become more demanding under a fixed evaluation setup. Before estimating this surface, fix the system, protocol, task set, labels for task pressure, and rule for counting success. Task pressure describes the conditions that make an agent's task more demanding. Group tasks by the combinations of task-pressure factors that actually occur in the data. Then run repeated trials for each task. Report the number of observations in each cell, task-level success, repeated-run consistency, and uncertainty intervals. Release random seeds, complete trajectories or replayable state changes, grader versions, and logs of any interventions. Do not estimate reliability for groups with too few tasks by guessing from other groups.

Use one episode as the experimental unit. Each episode record should identify the task, initial state, model version, harness version, tool schema, decoding settings, and random seed. Include the action–observation trajectory, final state, grader output, and any human intervention. Also report token use, call counts, latency, computing resources, and monetary cost. Comparisons should use repeated rollouts and report uncertainty at the task level. Use paired analysis when systems run on the same task instances. These controls prevent extra search, retries, verifier calls, context, or a more favorable harness from being misinterpreted as model progress.

C.1 One-Dimensional Reliability When Difficulty Changes Unevenly

Consider a declared stress path $g : h \mapsto \mathbf{z}$ along which task pressure does not decrease. This path describes one way in which task difficulty can increase. Its scalar path horizon is

$$H_\alpha(S, Q; g) := \sup\{h : R_{S,Q}(g(h)) \geq \alpha\}. \tag{6}$$

When all coordinates except axis d remain fixed, the corresponding conditional horizon is

$$H_\alpha^{(d)}(\mathbf{z}_{-d}; S, Q) := \sup\{u : R_{S,Q}(\mathbf{z}_{-d}, z_d = u) \geq \alpha\}. \tag{7}$$

This threshold is meaningful only for a specified one-dimensional slice where reliability does not increase as task pressure rises. If stress path g rises and falls rather than changing steadily, report the reliable set or the robust prefix:

$$H_\alpha^{\text{prefix}}(S, Q; g) := \sup\Big\{h : \inf_{0 \leq u \leq h} R_{S,Q}(g(u)) \geq \alpha\Big\}, \tag{8}$$

The robust prefix is the longest initial segment that maintains reliability of at least α. It prevents one easier instance at a later value of h from making the reported horizon look larger than it is.

Table 9: Measures for evaluating long-horizon agents. The first six rows cover outcomes and reliability. The remaining rows explain why a system succeeds or fails and which resources it uses.

Measure group	Typical measures	Main question	Reporting guidance
Final outcome	Success rate; exact match; goal-state match; pass@1	Did the agent finish the task successfully?	Report success for each task and define the grading rules clearly. Prefer executable tests or checks of the final environment state.
Multiple attempts	pass@k; best-of-k success	Does the agent succeed when it receives several attempts?	Report k, the sampling settings, and the selection rule. Separate capability with search from performance in a single deployment run.
Repeated-run reliability	pass^k; trial consistency; success variance	How consistently does the agent succeed across repeated runs?	Run multiple rollouts for each task and report the variance or confidence intervals (Yao et al., 2025; Khanal et al., 2026).
Estimate uncertainty	Binomial interval; task bootstrap; paired interval	How precise is the reliability estimate?	State the resampling unit, task count, rollout count, and interval. Do not treat several rollouts of one task as independent tasks.
Reliability boundary	$R_{S,Q}(\mathbf{z})$; $\mathcal{H}_\alpha$; $H_\alpha(S, Q; g)$; robust prefix	Under which task pressures does reliability reach at least α?	Keep the system and protocol fixed. Report the stress path or conditional slice. If difficulty rises and falls, use a reliable set or robust prefix.
Partial progress	Subgoal completion; milestone accuracy; partial credit	How much progress does the agent make before failing?	When final success is rare, distinguish failure near the start from failure after almost complete execution.
Action trajectory	Action validity; tool-use correctness; loop rate; process score	Did the agent follow a valid and efficient sequence of actions?	Record complete trajectories and evaluate the quality of the process as well as the final outcome (Zhuge et al., 2025).
Quality of verification	Check coverage; false accept/reject; verification delay	Can the evaluation protocol detect both success and failure?	Prefer executable checks. When model-based grading is unavoidable, disclose the judge prompts and calibration procedure.
Error recovery	Recovery rate; rollback success; actions or time to recovery	Can the system recover a valid state after an error?	Report error detection, error localization, rollback, and successful repair as separate steps.
Safe execution	State-diff correctness; unintended actions; rollback success	Did the agent change the environment only as intended?	Record the initial and final states, permitted changes, and any irreversible or harmful side effects.
Resource use and cost	Tokens; tool calls; latency; verifier calls; monetary cost	Which resources did the system use?	Compare systems with matched budgets and report success normalized by cost (Kapoor et al., 2025b).

D Detailed Mechanism Catalogues

This appendix provides the detailed method comparisons summarized in Sections 4 and 5. The main-text figures place each conceptual map beside the argument it supports. The tables below provide a compact method-by-method reference.

Table 10: Model-side methods for long-horizon work, their benefits, and their remaining limits.

Method group	Example methods	How it helps	Main limitation
Inference-time reasoning	CoT, ToT, LATS (Wei et al., 2022; Yao et al., 2023a; Zhou et al., 2024a)	Produces intermediate reasoning steps, compares alternative branches, and searches using feedback from the environment.	Uses substantial computing resources at inference time and relies on accurate evaluators.
Grounded planning traces	Plan-and-Act (Erdogan et al., 2025)	Turns successful trajectories into training data that links executable plans to actions.	Plans can become outdated as the environment changes and must be updated.

Continued on next page

Table 10 (continued)

Method group	Example methods	How it helps	Main limitation
Tool-use learning	Toolformer, Gorilla (Schick et al., 2023; Patil et al., 2024)	Learns when to call tools and how to make calls based on their documentation.	Training data often cover fixed distributions and provide weak optimization for complete tasks.
Trajectory-level RL	LOOP (Chen et al., 2025c)	Optimizes the agent's full trajectory through direct interaction with a stateful environment.	Rewards may not show which actions caused success, and results can depend heavily on the chosen environment.
Step- and action-level credit assignment	GiGPO, SALT, HCAPO, CAPO (Feng et al., 2025; Li et al., 2026b; Tan et al., 2026; Wang et al., 2026b)	Uses RL advantage estimates to judge how much each step, state, or action contributes, rather than scoring only the full trajectory.	Often needs repeated states, hindsight information, or multiple rollouts.
Stability and scaling	SORL, SAO, AgentGym-RL, IterResearch (Li et al., 2025a; Hou et al., 2026; Xi et al., 2025; Chen et al., 2025a)	Supports more stable updates, longer training, or richer representations of agent state.	Requires costly infrastructure; gains can be domain-specific, and their causes can be hard to identify.

Table 11: Methods for managing context and memory. The first five rows manage information within one episode. The last three preserve information across episodes or tasks.

Method group	Examples	Main limitation
Within one episode		
Observation selection & pruning	SWE-Pruner, Squeez, LongCodeZip, ACON, LCoW, observation masking (Wang et al., 2026h; Kovács, 2026; Shi et al., 2025; Kang et al., 2025; Lee et al., 2025; Lindenbauer et al., 2025)	Removed content may be needed later.
Trajectory summarization & folding	ReSum, AgentFold, Context-Folding, Trajectory Reduction (Wu et al., 2025; Ye et al., 2025; Sun et al., 2025; Xiao et al., 2026)	Compression can remove precise instructions or constraints.
In-context memory construction	MEM1, ContextWeaver, proactive extraction (Zhou et al., 2025; Wu et al., 2026b; Yang et al., 2026a)	The method must predict which stored state will be useful later.
Evidence & plan externalization	WebWeaver, Step-DeepResearch, ContextBudget (Li et al., 2025b; Hu et al., 2025a; Wu et al., 2026c)	External records must be linked back to the current task state before the agent can use them reliably.
Learned context managers	CompactionRL, AdaCoM, ACM (Li et al., 2026g; Yi et al., 2026; Li et al., 2026e)	Performance does not transfer consistently across agents.
Across episodes		
Factual memory	Generative Agents, MemGPT, HippoRAG, Mem0 (Park et al., 2023; Packer et al., 2023; Gutiérrez et al., 2024; Chhikara et al., 2025)	Stored facts can become outdated, and retrieval may return irrelevant items.
Experiential memory	ExpeL, A-MEM, ACE (Zhao et al., 2024; Xu et al., 2025b; Zhang et al., 2025d)	Lessons from earlier experience may not transfer to new settings.
Across tasks		
Procedural memory & skills	AWM, Memp, Voyager, JARVIS-1, SkillOS (Wang et al., 2025e; Fang et al., 2026; Wang et al., 2024a, 2025f; Ouyang et al., 2026)	Stored skills must be checked and updated to remain valid.

Table 12: Decisions an agent loop must make when it runs without human help, with example methods and their main limitations.

Decision to make	Example methods	Main limitation
Verify	Agent-as-a-Judge, ReVeal, process reward models (Zhuge et al., 2025; Jin et al., 2026b; Choudhury, 2025)	Vague success criteria can let agents game the verification process, a problem known as reward hacking (Wang et al., 2026a; Zhao et al., 2026).
Stop or continue	Early-exit policies and bounded control flow (Lu et al., 2025b; Wei, 2026; Sypherd and Belle, 2024)	Misleading progress signals can trap the agent in an endless loop (Xu et al., 2026a).
Escalate	Approval gates and human-on-the-loop interfaces (Mozannar et al., 2025)	Even frontier agents often fail to ask for human help at the right time (Gulati et al., 2026).
Localize and repair	Debugging that reruns the task after each attempted fix (Ma et al., 2026a)	Failures without a clear error signal are hard to locate. Knowing when to stop and how to verify a fix remain major obstacles (Cemri et al., 2025; Zhu et al., 2025).

E Benchmark Inventory and Evidence Sources

Table 13 lists 67 candidate benchmarks for which our structured review found usable evidence about individual components. The set contains 55 core benchmarks of long-horizon execution. It also contains 7 protocol-conditional entries, whose steps depend on one another only under specific evaluation settings, and 5 adjacent comparators, which are related benchmarks included only for comparison. We use functional coupling to mean that one step creates, changes, or selects information or state that a later step needs. Inclusion does not show that a benchmark's standard evaluation has functional coupling.

Table 14 lists integrated AI R&D evaluations and systems for which the evidence covers several capabilities. The two views overlap because PostTrainBench reports component scale with our time, step, or budget codes and also evaluates improvement in a complete AI artifact. We omit a domain entry when no identified source reports scale for individual tasks. This omission does not put the entry outside our scope. Most such omissions come from the web and browsing domain.

A typed extent is a reported scale labeled by whether it comes from time, interaction count, or budget. We assign one only when a primary source or an identified external measurement supports both the value and where it came from. Otherwise, we report both the type and extent as `NR` (not reported). A repository or dataset update replaces a paper snapshot only when it identifies an exact release or commit.

The prefixes `T`, `S`, and `B` mean time, reported interactions, and protocol budget or execution window, respectively. The time bins `T1`–`T5` are < 1 min, 1–10 min, 10–60 min, 1–4 h, and > 4 h. The interaction bins `S1`–`S5` are < 10, 10–< 30, 30–< 100, 100–< 300, and ≥ 300 reported steps, states, turns, actions, or calls. The budget codes `B1`–`B5` use the matching five-level range in the unit reported by the source. Readers need that unit because budget codes do not measure task length. Levels can be compared only when they share a prefix and come from compatible measurements. For example, `T3`, `S3`, and `B3` are not interchangeable. We prefer human or reference trajectories as evidence, then reported execution counts, and finally official budgets. We explicitly label values that depend on a specific rollout or agent setup. The inventory reports aggregate performance separately as part of the evaluation protocol.

Table 13: The 67 component benchmarks in this survey, grouped by domain. NR means that a value was not reported or does not apply. Italic text marks cases that depend on the evaluation protocol or are included only as comparators.

Benchmark	Domain and reported size	Reported duration, steps, or budget	Why later steps depend on earlier ones, and what state is retained	How the benchmark checks success
Navigating and browsing the web				
MiniWoB	Web user interfaces 100 tasks	NR NR	**Why steps are linked.** *Protocol-conditional:* only compound tasks make later page states depend on earlier widget actions; the inventory excludes atomic one-step tasks **State retained.** The current page, widget values, and episode state	The environment's reward after the task is complete
WebShop	Web shopping 12,087 instructions	S2 11.3 states (mean; human expert); max 114	**Why steps are linked.** Product constraints link search, comparison, option selection, and purchase **State retained.** The query and page state, visited and current products, selected options, and session state	The task reward, with exact success when the reward equals 1
Mind2Web	Offline web navigation 2,350 tasks	S1 7.3 actions (mean; offline reference demonstrations)	**Why steps are linked.** *Adjacent comparator:* the standard evaluation predicts each action from the ground-truth history instead of executing a trajectory controlled by the model **State retained.** Recorded DOM snapshots and ground-truth previous actions, supplied for each offline prediction	Teacher-forced checks provide the correct history at each step, then score the element and operation independently
WebLINX	Offline dialogue-based web navigation 2,337 demonstrations (v1.0 snapshot)	S3 43 turns (mean; v1.0 reference demonstrations)	**Why steps are linked.** *Adjacent comparator:* the standard evaluation scores reference actions one turn at a time instead of executing a trajectory controlled by the model **State retained.** Recorded dialogue and action history, DOM snapshots, and screenshots supplied offline (v1.0 snapshot)	Turn-level checks of reference intent, elements, and text-action similarity
WebArena	Web navigation 812 tasks	T1 35.38 s (median; external 100-task sample)	**Why steps are linked.** Subgoals span pages and websites, so earlier actions determine the context for later ones **State retained.** The website, account, and database state	Task-specific functional checks of the final state or answer
WebCanvas	Live web navigation 542 top-level tasks (HF 160677b)	S1 8.40 annotated steps/task (derived mean; range 3–28; HF 160677b)	**Why steps are linked.** Page transitions and form edits create prerequisites for later actions **State retained.** The live page, DOM, and form state (no login or personal-account state)	Matches of key-node URLs, element paths, and values, checked at both step and task level
VisualWeb Arena	Visual web navigation 910 tasks	S1 9.6 steps (mean; official GPT-4V+SoM rollout; specific to this model and agent setup)	**Why steps are linked.** Each website action requires visual grounding in the state created by previous actions **State retained.** The website, account, database, and task-relevant visual-content state	Functional evaluators of the final state
WebVoyager	Open-web navigation 643 tasks	B2 15 steps (maximum; official evaluation budget)	**Why steps are linked.** Earlier web actions determine which page state later actions can access **State retained.** The live browser session, navigation history, current page, and website state	A multimodal GPT-4V judge that assesses task success from final answers and recent screenshots

Continued on next page

Table 13 (continued)

Codes: T = time, S = steps or actions, B = budget, NR = not reported or not applicable.

Benchmark	Domain and reported size	Reported duration, steps, or budget	Why later steps depend on earlier ones, and what state is retained	How the benchmark checks success
Using computers, mobile devices, and embodied environments				
OSWorld	Desktop computer use 369 tasks	T2 111.94 s (median; $n = 369$)	**Why steps are linked.** User-interface and file operations create preconditions for later actions **State retained.** Files, applications, documents, settings, and operating-system state	Execution-based checks of the final application, file, and operating-system state
OSWorld 2.0	Professional computer workflows 108 tasks	T4 About 1.6 h (median; annotated/estimated); 69.6% $>$ 1 h	**Why steps are linked.** Workflows span applications, depend on earlier work, and receive dynamic updates **State retained.** Workspace files, application and service records, user-profile data, messages, and injected updates	Detailed checkpoints on the final state, using functional checks and limited model-based judging
DeskCraft	Professional desktop workflows 538 tasks (386 standard / 152 interactive)	S3 62.2 executed steps (mean; successful Kimi-K2.6 L3 standard rollouts; specific to this model and agent setup)	**Why steps are linked.** Dependent graphical-interface edits, plus phase-triggered clarification, interruption, and feedback, constrain later actions **State retained.** Desktop and application state, project files, exported artifacts, active instructions, and interaction history	Programmatic checks of final states and artifacts in the application and runtime state
Windows Agent Arena	Windows computer use 154 tasks	S1 8.1 steps (mean; one-participant human evaluation)	**Why steps are linked.** Actions on windows, applications, and files create preconditions for later actions **State retained.** Windows applications, files, settings, and desktop state	Task-specific evaluators of the final state
OmniACT	Desktop and web control 9,802 tasks	S1 1 action (derived median; mean 1.66, range 1–11; official scripts)	**Why steps are linked.** *Adjacent comparator:* the standard evaluation maps one static screenshot to a complete action script, so model actions do not create later observations **State retained.** N/A (the model predicts an action sequence from a single-screen screenshot)	Static checks against reference action sequences; model actions do not change later task states during evaluation
AndroidWorld	Mobile computer use 116 task templates	S1 6 optimal steps (median); range 1–60	**Why steps are linked.** Earlier user-interface actions change the application and device state that later actions require **State retained.** The Android user interface, application databases, records, files, and settings	Programmatic rewards based on device and application state
B-MoCA	Mobile-device control 131 tasks (release/ver.3 de06497)	B1 8 steps (derived median task cap; range 3–19; official max_episode_steps)	**Why steps are linked.** Each device-control action depends on the screen state created by previous actions **State retained.** The Android user interface, application state, language, and settings	Checks of whether the goal state was reached
KnowU-Bench	Personalized mobile assistance 192 tasks (42 general /86 personalized /64 proactive)	B3 50 steps (maximum; official evaluation budget; repo c03a825)	**Why steps are linked.** Inferring preferences, asking for clarification and consent, and using the graphical interface make later actions depend on user responses **State retained.** Android and application state, timestamped behavior logs, dialogue history, current context, and consent state	Rule-based checks of tasks and trajectories, plus rubric-conditioned model judging of the evidence and dialogue

Continued on next page

Table 13 (continued)

Codes: T = time, S = steps or actions, B = budget, NR = not reported or not applicable.

Benchmark	Domain and reported size	Reported duration, steps, or budget	Why later steps depend on earlier ones, and what state is retained	How the benchmark checks success
ALFRED	Embodied household tasks 8,055 demonstrations	S3 50 action steps (mean); 7.5 subgoals (mean)	**Why steps are linked.** The Planning Domain Definition Language (PDDL) defines chains in which each action creates conditions for later actions **State retained.** The agent's pose, object locations, receptacles, and object states	Checks of overall task success and individual goal conditions
BEHAVIOR-1K	Embodied activities 1,000 activities	NR NR	**Why steps are linked.** The BEHAVIOR Domain Definition Language (BDDL) defines start and goal conditions that link changes in object state and position **State retained.** The scene graph, object poses, relations, and physical states	Checks of the stated goal conditions and overall task success
RoboCerebra	Robotic manipulation 1,000 human trajectories / 100 task variants; evaluation: 60 tasks / 600 rollouts	S5 2,972.4 simulation steps (mean; human trajectories)	**Why steps are linked.** Each manipulation subgoal depends on the completion of earlier operations **State retained.** The scene, object states, robot state, and subtask progress	Simulator predicates for object state, exact plan matching, and VideoQA checks
WeaveBench	Hybrid computer use 114 tasks	S3 76 tool calls (median; best official Hybrid rollout/task); mean 88, range 14–471	**Why steps are linked.** Graphical-interface, command-line, code, and filesystem actions create and reuse the same artifacts **State retained.** Desktop and application state, source and configuration files, artifacts, logs, and service status	An agent-based judge reviews the full action history, artifacts, screenshots, and logs, and checks for shortcuts
HeroBench	Planning in a virtual world 844 tasks (180 evaluated)	S3–5 Difficulty-stratum means 40–937 environmental actions (mean ± SD)	**Why steps are linked.** Resource dependencies and action order limit which choices remain available later **State retained.** The simulator location, inventory, equipment, skill levels, and resource counts	Checks of goal completion and partial progress
Software engineering and command-line work				
SWE-bench	Software engineering 2,294 tasks	T3 Estimated median bucket 15–60 min (official random annotation sample, $n = 1{,}699$)	**Why steps are linked.** Changes across files must work together to satisfy both issue-specific and regression tests **State retained.** The working tree, source code, dependencies, build state, and test state	FAIL_TO_PASS and PASS_TO_PASS test results
SWE-bench Verified	Software engineering 500 tasks	T3 Derived median bucket 15–60 min (official annotations, $n = 500$)	**Why steps are linked.** Changes across files must work together to satisfy both issue-specific and regression tests **State retained.** The working tree, source code, dependencies, build state, and test state	FAIL_TO_PASS and PASS_TO_PASS test results
SWE-Cycle	Issue-resolution cycle 489 tasks	B4 90 min (isolated) / 3 h (FullCycle), maximum	**Why steps are linked.** Reconstructing the environment, implementing code, and generating verification tests depend on outputs from earlier phases **State retained.** The repository, execution environment, reproduced issue, and test state	SWE-Judge static code review together with dynamic execution tests
SWE-bench-Live	Software engineering 1,319 tasks (initial release)	B3–4 60 iterations (OpenHands) / 100 LLM calls (SWE-agent), maximum	**Why steps are linked.** Changes across files must work together to satisfy new issue-specific and regression tests **State retained.** The repository snapshot, issue context, working tree, and tests	Executable tests written for each issue

Continued on next page

Table 13 (continued)

Codes: T = time, S = steps or actions, B = budget, NR = not reported or not applicable.

Benchmark	Domain and reported size	Reported duration, steps, or budget	Why later steps depend on earlier ones, and what state is retained	How the benchmark checks success
SWE-MERA	Software engineering 728 samples (2024-09-01–2025-06-23 leaderboard snapshot)	NR NR	**Why steps are linked.** Repairing an issue links fault localization, code edits, and validation **State retained.** The repository snapshot, source files, dependencies, and tests	Executable tests run at repository level
SWE-rebench V2	Software engineering 32,079 tasks	NR NR	**Why steps are linked.** Repair across programming languages depends on feedback from builds and tests **State retained.** The repository, language toolchain, dependencies, and test state	Executable tests written for each task
SWE-Bench Pro	Professional software engineering 1,865 tasks	B3 50 turns (maximum; official evaluation budget)	**Why steps are linked.** Professional requirements link implementation choices across files **State retained.** The large repository, dependencies, build artifacts, and tests	Acceptance and regression tests run at repository level
SWE-Bench ProMax	Multilingual code refactoring 170 instances /70 repositories /7 languages	B5 300 steps (maximum; official evaluation budget)	**Why steps are linked.** Large-scale refactoring requires coordinated changes across files while preserving existing behavior **State retained.** The repository, working tree, dependencies, build state, tests, and execution feedback	All task-specific tests must pass, and experts review the specifications and test suites
DeepSWE	Software engineering 113 tasks	B4 2.5 h (wall-clock timeout; no step or cost cap)	**Why steps are linked.** Implementations across files must work together to satisfy task-specific behavior verifiers **State retained.** The repository, source changes, build state, and test results	Hand-written program verifiers that determine task-specific acceptance
SWE-Marathon	Project-scale software engineering 20 tasks	T5 Expert-human estimate 40–400 h	**Why steps are linked.** Project-scale work repeatedly cycles through implementation, building, and testing **State retained.** The task workspace, source and output artifacts, build and runtime state, and measured results	Tests, behavioral checks, judges, and performance gates
FeatureBench	Feature development 200 tasks (full set)	B5 500 steps (maximum; OpenHands paper baseline)	**Why steps are linked.** Feature requirements link architectural choices, implementation, and tests **State retained.** The repository, feature artifacts, dependencies, and regression state	Acceptance and regression tests written for each feature
SWE-Explore	Repository exploration 848 tasks	B1–3 8 search calls (LocAgent) / 25–30 steps (Mini-SWE-agent/ AweAgent), maximum	**Why steps are linked.** The agent must select and rank related code regions together under a fixed line budget **State retained.** The repository snapshot, issue specification, and ranked code-region output (no code changes)	Ground-truth labels for relevant files and locations
ReCUBE	Repository-level code generation 366 masked-file instances	S2 20.3 turns (mean; GPT-5 Mini + CCE setting only)	**Why steps are linked.** *Protocol-conditional:* the Agent+CCE setting uses feedback from the repository; ReCUBE also has a single-turn condition with full context **State retained.** Unmasked repository files, dependency specifications, documentation, and the reconstructed file	Usage-aware executable tests of internal logic and integration across files

Continued on next page

Table 13 (continued)

Codes: T = time, S = steps or actions, B = budget, NR = not reported or not applicable.

Benchmark	Domain and reported size	Reported duration, steps, or budget	Why later steps depend on earlier ones, and what state is retained	How the benchmark checks success
SWE Atlas	Repository-level engineering 284 tasks	S3 56 tool calls (mean; Opus 4.6) / 58 (mean; GPT-5.4), both with mini-SWE-agent	**Why steps are linked.** Repository context and runtime behavior together constrain answers, tests, or refactored code **State retained.** The repository, retrieved context, working tree, and test state	Programmatic checks plus task-specific LLM judging based on rubrics
ProgramBench	Program reconstruction 200 tasks	B5 6 h / 1,000 steps (maximum; mini-SWE-agent paper baseline)	**Why steps are linked.** The agent must reconstruct observed behavior across modules that depend on one another **State retained.** The generated repository, module interfaces, build state, and execution state	Fuzzed end-to-end behavioral tests against reference executables
NL2Repo-Bench	Repository generation 104 tasks	S3 86.0 tool calls (derived mean; Claude-Sonnet-4.5/OpenHands official rollout)	**Why steps are linked.** Natural-language requirements jointly constrain repository components that interact **State retained.** The generated files, interfaces, dependencies, build state, and tests	A repository build and tests written for each task
RepoGenesis	Microservice generation 30 eval / 76 train repositories	B1–4 5 iterations (MetaGPT) / 100 chat rounds (MS-Agent), maximum	**Why steps are linked.** README requirements link service design, APIs, and implementation **State retained.** The generated services, API schema, dependencies, and deployment artifacts	API checks and executable integration checks
SWE-EVO	Software evolution 48 tasks	B4 100 iterations/calls (maximum; official evaluation budget)	**Why steps are linked.** High-level release requirements link coordinated changes across files with preservation of existing behavior **State retained.** The pre-release codebase, agent patch, dependencies, build state, and test results	Executable tests run after each stage of evolution
SWE-CI	Continuous integration 100 tasks	B2 20 turns (maximum; official evaluation budget)	**Why steps are linked.** Feedback from continuous integration guides each successive maintenance decision **State retained.** The repository, workflow configuration, continuous-integration logs, and test state	Continuous-integration jobs and executable test outcomes
SWE-Milestone	Continuous evolution 98 graded milestones	T5 1–3 days (average human-estimated development time per milestone)	**Why steps are linked.** Requirements for each milestone and edits from earlier milestones constrain later stages **State retained.** The persistent repository, milestone artifacts, commits, and tests	FAIL_TO_PASS and PASS_TO_PASS test results at each milestone
RoadmapBench	Version upgrades 115 tasks	S4 134 agent steps (fleet mean across official evaluations)	**Why steps are linked.** A roadmap with several targets jointly constrains coordinated changes across files from the source to the target version **State retained.** Repository versions, dependencies, migration state, and tests	Executable tests for each subtask against behavior in the target version
SWE-Chain	Release-level upgrades 155 version transitions	S5 1,057–2,008 tool calls (per-chain means across official agents)	**Why steps are linked.** A chain of package upgrades makes each release depend on changes made for earlier releases **State retained.** The package repository, dependency versions, release state, and tests	Build and test checks after the chained releases

Continued on next page

Table 13 (continued)

Codes: T = time, S = steps or actions, B = budget, NR = not reported or not applicable.

Benchmark	Domain and reported size	Reported duration, steps, or budget	Why later steps depend on earlier ones, and what state is retained	How the benchmark checks success
ChainSWE	Maintenance across multiple bugs 304 bug instances (100 chains)	B3–4 30 min / 100 turns (maximum per bug instance)	**Why steps are linked.** Bug fixes interact because they modify the same evolving codebase **State retained.** The persistent repository, previous fixes, issue state, and test results	Executable tests for each bug and for the final chain
TEBench	Test evolution 314 tasks	B3 30 min (default per-task timeout in official coding-agent runners; configurable)	**Why steps are linked.** Changes in source code determine which tests become stale, break, or are missing **State retained.** The updated production code, evolving test suite, generated test patch, and test results	Developer-provided ground-truth labels for affected tests, plus checks of executability, coverage overlap, and patch similarity
Terminal-Bench 2.0	Terminal and systems work 89 tasks	T4–5 Expert median bucket 1 h–1 day (74 timed tasks)	**Why steps are linked.** Commands, debugging, builds, and artifacts depend on the state created by earlier work **State retained.** The filesystem, packages, processes, artifacts, and logs	Executable tests written for each task and run in isolated containers
LongCLI-Bench	Command-line programming 20 tasks	T5 1,000+ min (mean; expert time)	**Why steps are linked.** Long command-line implementation sequences create and reuse artifacts from earlier steps **State retained.** The filesystem, source code, shell, processes, build state, and logs	Executable checks at individual steps and at the end
BuildBench	Building software 148 test repositories	S1 6.6 error-resolution attempts (mean; official agent rollout)	**Why steps are linked.** Commands that install dependencies and build software create prerequisites for later stages **State retained.** The source tree, dependencies, build artifacts, environment, and logs	Strict and flexible checks of build success
DI-BENCH	One-shot dependency inference 581 repositories	T3 20 min (maximum; official human study on 40 repositories; not agent runtime)	**Why steps are linked.** *Adjacent comparator:* dependency choices across the repository affect installation and execution, but the model predicts them in one shot **State retained.** The repository and manifests are supplied statically, with no state from iterative agent feedback	Checks of dependency matching, installation, and execution
CI-Repair-Bench	Continuous-integration repair 567 tasks	S2 28.4 LLM API calls/task (derived mean; GPT-5-mini official agent pipeline)	**Why steps are linked.** A full continuous-integration rerun jointly constrains code, workflow, environment, and dependency artifacts **State retained.** The repository, workflow files, continuous-integration logs, patches, and test state	Reruns of the continuous-integration workflow and tests
CLI-Tool-Bench	Command-line tool generation 94 repositories (paper v2)	S3 49.38 steps (mean; Kimi-k2.5/Mini-SWE-agent official rollout; paper v2)	**Why steps are linked.** The specification links implementation, packaging, and invocation of the tool **State retained.** The generated repository, package metadata, build artifacts, and file and system state visible in the sandbox	Black-box tests that execute commands

Continued on next page

Table 13 (continued)

Codes: T = time, S = steps or actions, B = budget, NR = not reported or not applicable.

Benchmark	Domain and reported size	Reported duration, steps, or budget	Why later steps depend on earlier ones, and what state is retained	How the benchmark checks success
SWE-Infra Bench	Infrastructure code 100 tasks	B1 1–2 LLM turns (official solver configurations; only the two-turn condition uses test feedback)	**Why steps are linked.** *Protocol-conditional:* in the two-attempt condition, test feedback links successive infrastructure edits; the one-turn baseline has no such link **State retained.** The AWS CDK repository, resource definitions, dependency graph, and test state	Provided tests of synthesized CloudFormation templates
FrontierSWE	Frontier technical work 17 tasks	B5 20 h/task (protocol budget)	**Why steps are linked.** Open-ended technical requirements link implementation to task-specific functional, research, or performance objectives **State retained.** The task workspace, source code and checkpoints, experimental variants, and build and benchmark results	Task-specific functional and performance checks, with correctness and anti-cheat gates
PostTrainBench	AI R&D and post-training 28 task configurations	B5 10 h on one H100/ configuration (protocol budget)	**Why steps are linked.** The work cycles through data preparation, training, evaluation, and tuning **State retained.** Training data, scripts and configurations, checkpoints, logs, and evaluation results	Evaluation on a held-out target benchmark, plus an LLM judge for contamination and model substitution
Workflows with tools, APIs, applications, and workplace systems				
ToolBench	Using tools and APIs 1,200 test instances (six official subsets, 200 each; derived)	NR NR	**Why steps are linked.** *Protocol-conditional:* in subsets with several rounds or tools, later calls use earlier outputs and argument choices **State retained.** The trajectory context, including call history, tool outputs, and selected tools; the benchmark manages no mutable external state	LLM-based ToolEval judgments of task success and pairwise preference over action sequences
BFCL	Calling functions 5,551 question–function–answer pairs (ICML 2025 paper snapshot)	NR NR	**Why steps are linked.** *Protocol-conditional:* in stateful agentic subsets, later calls depend on earlier outputs, user turns, and backend state **State retained.** For stateful subsets only, the backend API state, conversation history, available functions, and tool responses	Executable checks of function calls and structured matches
AppWorld	Application and API workflows 750 tasks	S3 42.5 validation-solution API calls (mean; Test-N* = Train+ Dev+Test-N) / 46.8 (mean; Test-C)	**Why steps are linked.** API calls across applications use and change results from earlier calls **State retained.** Application databases, stateful execution-shell variables and results, and simulated date and time	State-based unit tests, including checks for collateral damage
WorkArena	Enterprise web workflows 19,912 instances across 33 task types	S2 13.7 oracle actions (derived instance-weighted mean)	**Why steps are linked.** Changes to pages and records link ServiceNow navigation with operations on forms and lists **State retained.** The browser session, ServiceNow records, forms, and task state	Task-specific evaluators of final records and interface state
OfficeBench	Office automation 300 tasks	B3 50 steps (maximum; official evaluation budget)	**Why steps are linked.** Document operations across applications create and reuse intermediate artifacts **State retained.** The filesystem, office documents, emails, calendar events, and application state	Exact-match, fuzzy-match, and execution-based checks of the task

Continued on next page

Table 13 (continued)

Codes: T = time, S = steps or actions, B = budget, NR = not reported or not applicable.

Benchmark	Domain and reported size	Reported duration, steps, or budget	Why later steps depend on earlier ones, and what state is retained	How the benchmark checks success
OdysseyBench	Office-application workflows 602 tasks (300 OdysseyBench+ / 302 OdysseyBench-Neo)	S1–2 Most tasks require 3–15 execution turns (no median reported)	**Why steps are linked.** Execution uses facts retrieved from supplied multi-day histories and outputs produced earlier in other applications **State retained.** The supplied multi-day dialogue history, plus changing files, documents, emails, and calendar state	Exact, fuzzy, and execution-based checks of the saved final filesystem
TheAgent Company	Workplace simulation 175 tasks	S2 27.2 steps (mean LLM calls; OpenHands 0.28.1 + Gemini-2.5-Pro official rollout)	**Why steps are linked.** Browsing, coding, communication, and interactions with coworkers contribute to shared subgoals **State retained.** Organizational services, artifacts, messages, and coworker state	Deterministic or LLM-based checks of results, with subcheckpoint checks for partial credit
Planning and scientific work with many constraints				
TravelPlanner	Travel planning 1,225 planning queries	T3 About 12 min/reference plan (mean human annotation; not agent runtime)	**Why steps are linked.** Budget, time, route, and preference constraints link choices across the itinerary **State retained.** *Protocol-conditional:* the two-stage mode retains retrieved records and a partial itinerary; the sole-planning mode is static	Rule-based checks of explicit and commonsense constraints
NATURAL PLAN	One-shot planning 3,600 test examples	NR NR	**Why steps are linked.** *Adjacent comparator:* several constraints jointly shape the output plan, but the model follows no interactive trajectory **State retained.** N/A: the tool outputs and constraints are static	An exact match to the golden plan
DeepPlanning	Travel and shopping 360 language-specific cases (240 unique tasks)	B5 400 tool calls (maximum; official evaluation budget)	**Why steps are linked.** Gathered information and local choices interact with constraints on the complete solution **State retained.** The sandbox data, tool results, constraints, and partial plan	Code-based checks of itinerary constraints and matches between requested and selected cart items
MinePlanner	Planning in a symbolic world 45 tasks	S3 42 actions (derived median of 45 human-expert plan lengths); range 3–1,268	**Why steps are linked.** *Protocol-conditional:* open-loop PDDL links action preconditions and effects, but it has no interactive LLM trajectory **State retained.** The symbolic world, inventory, object relations, and goal state	Checks that the plan can execute and satisfies the goal predicates
Robotouille	Interactive multi-agent planning 300 instances (100 per setting)	S2–3 10–82 optimal steps (20 default single-agent tasks; multi-agent extent NR)	**Why steps are linked.** Delays and interruptions link subplans that agents carry out in an interleaved order **State retained.** Objects, cooking state, timers, the action queue, and subtask progress	Simulator checks of the goal state and timing
ALE-Bench	Algorithm engineering 40 problems; Lite is a 10-problem subset	B4 Contest-window median 4 h (range 3.5–367 h); experiment cap 4 h/problem	**Why steps are linked.** The agent proposes, implements, runs, receives score or error feedback, and then refines the algorithm **State retained.** The current and best code, scores, feedback, and execution logs	An executable score for each run, with hidden or private cases grading the final best code

E.1 Evidence from Integrated AI R&D Evaluations and Systems

Table 14: System-level AI4AI evaluations that combine several capabilities. This table is separate from the 67 component benchmarks. PostTrainBench appears in both tables because the evidence supports both views.

Evaluation or system	Part of the AI system that changed	Types of work combined	How the evaluation measured improvement	How the evaluation separated possible causes
MLE-bench (Chan et al., 2025)	The competition model and pipeline	End-to-end machine-learning engineering across 75 Kaggle competitions	Scores on held-out competitions	The agent wrapper and tools are fixed, but the evaluation does not separate the effects of the model and budget
RE-Bench (Wijk et al., 2025)	AI R&D artifacts	Open-ended R&D in seven environments	Executable scores specific to each task	Comparisons with expert-human attempts given the same amount of time
PaperBench (Starace et al., 2025)	The research implementation	Reproduction of 20 ICML papers from descriptions	Hierarchical rubrics validated by the authors	Artifacts and rubrics are visible, but the effects of the model and harness remain coupled
MLR-Bench (Chen et al., 2025b)	The machine-learning research result	Work from forming ideas through producing the report	Task-specific scores for artifacts and research	End-to-end comparisons, with few tests that vary one factor at a time
PostTrainBench (Rank et al., 2026)	The model checkpoint	Data and recipe selection, training, and evaluation	A held-out benchmark, using one H100 for ten hours	A shared sandbox and budget, with logs that reveal shortcuts
AiScientist (Chen et al., 2026a)	The state of the research project	Long-running refinement by specialized roles that hand files to one another	Outcomes on MLE-bench Lite and PaperBench	Tests that remove or change state and coordination, without evaluating every combination of base model and setup
Frontis-MA1 / OpenMLE (Yang et al., 2026c)	The MLE program, operators, and search harness	Operator training together with experience-guided evolutionary search	MLE-bench Lite (12 hours) and transfer to NatureBench Lite	Comparisons that match the model, match the harness, or evaluate the combined system
Kimi K3 (Kimi Team, 2026b)	The model, agent policy, and harness distribution	Million-token reinforcement learning with persistent rollout and sandbox state	Domain-specific executable metrics and judge-based metrics	An industrial system designed as a whole, with no public experiment that varies each full-stack component separately

Table 15: What AI4AI studies published in 2026 have shown, and what their evaluations do not yet establish.

Study	Evaluation setup	Strongest reported result	What the result does not establish
AIRS-Bench (2026)	20 research tasks, with no baseline code	The reported human SOTA was exceeded on 4/20 tasks	Fixed tasks and metrics test execution ability, not the ability to choose a research direction
NatureBench (2026i)	90 containerized tasks from Nature-family papers, with the source methods hidden	The strongest agent surpasses the published SOTA on 17.8% and matches it on 47.8%	The tasks mostly test translation of a method into an implementation, not demonstrated invention
RSIBench-Data (2026a)	A fixed stack for iteratively developing data strategies	58.33% improve initially; 78.26% of searches after the peak end below that peak	Scores can rise and then fall, and the study does not resolve when to stop searching
The AI Scientist / Nature (2026a)	Template-based and free-form systems that work from an idea to a paper	One of three selected papers reaches workshop level	The process still uses human filtering, and no paper reaches the main-conference bar

Continued on next page

Table 15 (continued)

Study	Evaluation setup	Strongest reported result	What the result does not establish
FARS (2026)	An unattended multi-agent deployment that works from an idea to a paper	166 papers/67 topics; 282 reviews cover 140 outputs	The result demonstrates scale and allows outputs to be inspected, but it does not show quality improving steadily
AutoSOTA (2026f)	Reproduction and optimization from a paper to a repository	Improved models are reported for 105 executable papers	Criteria derived from each paper guide both search and evaluation, and the study does not isolate transfer
ResearchArena (2026e)	117 papers, reviewed from the manuscript alone and with access to artifacts	Text scores are competitive, but no paper reaches the top-tier bar after audit	The audit finds fabrication, weak experiments, and mismatches between plans and execution
Shadow evals. (2026)	Two unpublished NeurIPS questions, six days, and expert grading	The engineering work is completed, but there is no meaningful research progress	The evaluation reveals weak judgment, backtracking, resource use, and choice of direction