Tencent Hy

Diving into Reliable Self-Evolving Agents: A Survey

Kaiqi Wang[1,*,‡], Wenjin Hou[1,2,*], Yuchen Yan[1,2,*], Hongrui Jia[1,3,*], Zhisheng Zhong[1,*], Botao Ren[1,4,*]
Yifei Chen[1], Songyang Zhang[1,§], Yongliang Shen[2], Jun Xiao[2], Yi Yang[2], Yueting Zhuang[2], Hehe Fan[2,†]

[*]Core Contributors [‡]Project Lead [§]Project Supervisor [†]Corresponding Author

[1]Tencent Hunyuan [2]Zhejiang University [3]Peking University [4]Tsinghua University

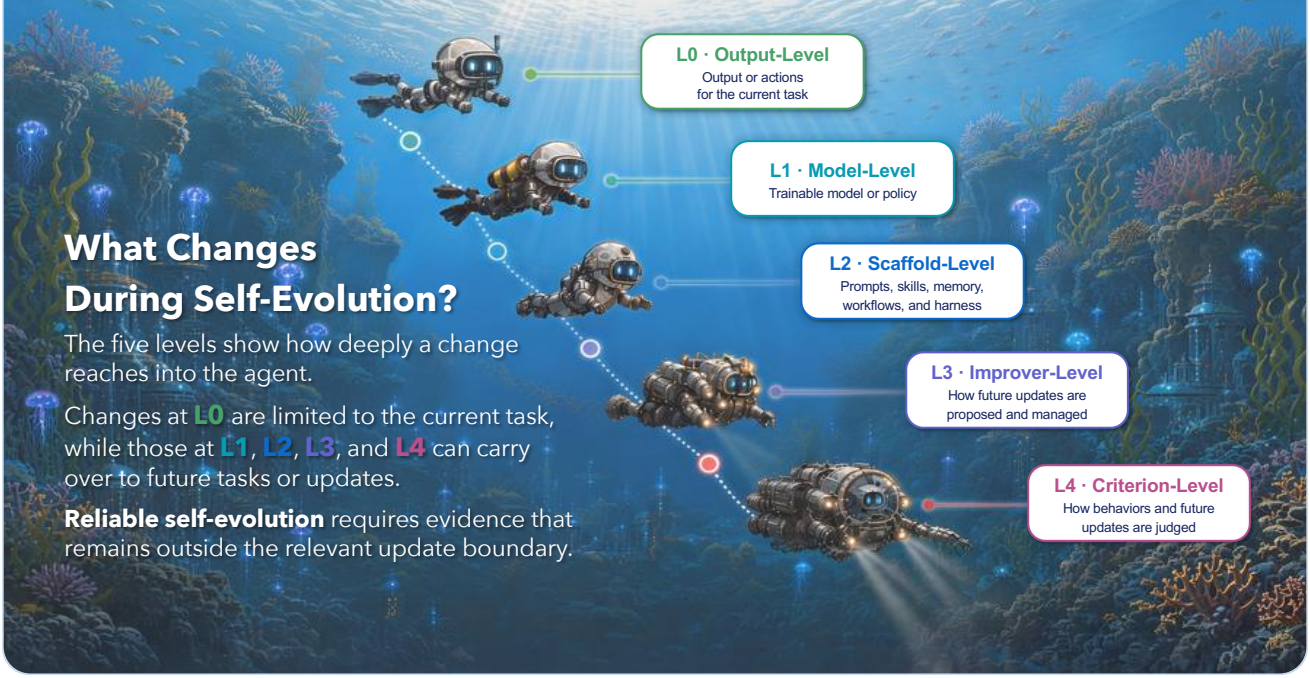
wkqscut@gmail.com, {wenjinhou,yanyuchen,hehefan}@zju.edu.cn

Awesome-Reliable-Self-Evolving-Agents

Abstract

Self-evolving agents use information produced during their own execution to revise current outputs or modify retained agent components that shape later behavior and future updating. As these systems gain broader and more persistent self-modification capabilities, **reliability** becomes a central concern. Two questions guide this survey: what changes during self-evolution, and what evidence can support claims of improvement. The literature is organized by **self-evolution depth**, defined by the deepest evolution target whose change takes effect—from the current output to retained components that shape future behavior, updates, or judgments. Accordingly, we classify self-evolution into five levels: **Output-Level** Self-Evolution (L0), **Model-Level** Self-Evolution (L1), **Scaffold-Level** Self-Evolution (L2), **Improver-Level** Self-Evolution (L3), and **Criterion-Level** Self-Evolution (L4). For each level, we survey representative systems and compare their evolution mechanisms, changed objects, persistence conditions, and reliability challenges. We then provide a cross-level synthesis of these concerns and develop a reliability ladder that pairs each evolution target with evidence and controls outside the corresponding update boundary. Reliable self-evolution thus depends not on self-evolution depth alone, but on whether evaluation and oversight remain independent of the update and cover the relevant tasks, conditions, and constraints. Finally, we discuss key challenges and future research directions. We aim for this survey to serve both as a structured reference for existing work and as guidance for developing the next generation of capable and reliable self-evolving agents.

Keywords self-evolving agents; recursive self-improvement; reliability; self-evolution depth; external audit; reliability ladder



Contents

1 Introduction

Large language models (LLMs) are increasingly embedded in agentic systems that can plan, use tools, maintain memory, interact with environments, and coordinate multi-step actions [1–3]. Recent work emphasizes that the capabilities of an agentic system depend not only on its base model but also on the surrounding harness, which organizes execution, tool use, context, persistent state, and evaluation [4]. As these model–harness systems are considered for open-ended, knowledge-intensive settings, their ability to specialize and adapt after deployment becomes an important design requirement [5]. Conventional improvement pipelines depend heavily on human-generated data, expert feedback, manually designed workflows, and repeated engineering effort. Although these external inputs and engineering resources remain indispensable, their limited availability and the difficulty of covering open-ended deployment conditions can constrain the scalability of conventional improvement pipelines [5]. At the same time, agents can generate training data [6], critique candidate solutions [7], write and test code [8, 9], retain reusable plans or skills [3], and evaluate candidate system components [10, 11]. These capabilities shift improvement from a purely external engineering process toward an increasingly endogenous one, in which agents help produce the knowledge, feedback, and mechanisms that shape their future behavior. When this participation produces changes that enhance the system's capacity for further improvement, self-evolution becomes recursive, making recursive self-improvement (RSI) an operational research problem for LLM-based agents. We therefore situate RSI within the broader landscape of agent self-evolution rather than treating it as an isolated category.

To describe ordinary self-evolution and its recursive forms within a common framework, we organize this landscape into five levels according to self-evolution depth. Self-evolution depth is determined by the deepest evolution target whose semantic modification causally affects the functional chain that produces behavior, collects evidence, proposes later changes, and judges those changes. Representative systems illustrate the progression from task-local output refinement in Self-Refine [7], through retained model updates in WebEvolver [12] and persistent scaffold changes such as PromptAgent's optimized prompts and SkillWeaver's reusable skills [11, 13], to changes that reach the procedures and criteria governing subsequent updates. The Darwin Gödel Machine retains self-modifications that affect how later descendants are generated [14], while the Red Queen Gödel Machine promotes evaluators that govern subsequent search [15]. **L0: Output-Level Self-Evolution** changes only a task-local output or trajectory, whereas **L1: Model-Level Self-Evolution** writes experience produced during agent execution into trainable model state and retains the resulting parameter update across later independent tasks. **L2: Scaffold-Level Self-Evolution** retains changes to structures surrounding the model, including prompts, tools, memory operations, workflows, runtime harnesses, and multi-agent topology. **L3: Improver-Level Self-Evolution** brings the procedure that proposes, selects, commits, or rolls back future updates inside the update boundary. **L4: Criterion-Level Self-Evolution** retains changes to the evaluator protocol, judgment tasks, rewards, metric semantics, constraints, or values that determine how later updates are judged. These levels are not parallel method categories: each transition is classified by its self-evolution depth, with L0 remaining task-local and L1–L4 requiring a retained change that affects later independent tasks or future updating. Under this structural definition, RSI begins at L3, where a retained change reaches the improver, and extends at L4 to the criterion. This boundary does not by itself imply that the system improves or that improvement accelerates. Accordingly, the hierarchy describes the functional position of a modification rather than its magnitude, complexity, or reliability.

A growing body of surveys has examined parts of this landscape from complementary perspectives. General surveys of LLM self-evolution organize the process as a cycle of experience acquisition, refinement, model updating, and evaluation, or classify methods by their evolution objectives [16]. A recent system-level account complements these process views by separating foundation-model improvement from scaffold improvement and further organizing methods by the object updated and the source of the improvement signal [17].

Experience-centered surveys trace how interaction histories become memory, skills, model behavior, and meta-level control [18], while self-feedback surveys examine the interaction between self-evaluation and self-update [19]. Related work distinguishes internal model adaptation from retrieval- and tool-based lifelong learning [20], and frames long-horizon agency through the coupling of externalized harness engineering and internalized model optimization [21]. Component-centered surveys analyze memory mechanisms [22] and agentic environments [23], while surveys of trustworthy agentic AI organize risks around safety, robustness, privacy, and system security [24]. Together, these works provide process taxonomies, system and component analyses, evaluation resources, and accounts of individual reliability challenges.

Existing surveys have discussed aspects of **reliability** by examining evolution processes, system components, and trustworthiness concerns, but they do not generally treat reliability as a unified axis for analysis across levels. In particular, existing work has not yet fully developed a cross-level synthesis linking self-evolution depth, the persistence of modifications, audit-evidence boundaries, promotion conditions, and characteristic reliability failures. This gap becomes more consequential as self-evolution expands from task-local revisions to persistent structural changes. Broader update processes can create longer causal chains and wider downstream effects, increasing opportunities for errors or misaligned objectives to propagate across rounds.

Reliability therefore becomes increasingly central, not because deeper self-evolution is necessarily less reliable, but because evaluating improvement claims and assessing robustness, traceability, controllability, and safety become more demanding parts of system design and evaluation. In this survey, reliability depends on how well the evidence supports the claimed improvement. The external target used to evaluate that improvement must remain fixed and cannot be changed by the update. Accordingly, this survey examines self-evolving agents as its broader subject, situates RSI within that landscape, and adopts reliability as a unified analytical perspective. Across the L0–L4 taxonomy, we ask how reliability requirements vary with the evolution target, the persistence of the modification, and the degree of recursion. A central concern is what evidence remains outside each update's control and can therefore support a trustworthy improvement claim. We assess whether claimed improvements are supported by evidence outside the optimization process, whether they generalize across tasks and environments, and whether their origins and downstream effects remain traceable and independently verifiable. We further examine whether improvement processes preserve safety constraints, support monitoring and intervention, and permit rollback or termination when failures occur. Reliable self-evolution therefore depends not on self-evolution depth alone, but on whether evaluation and oversight remain independent of the update being assessed and adequately cover the scope of the improvement claim. Reliability is not a final check at the end of an evolution pipeline. It is a requirement that extends across self-evolution and its recursive forms.

The remainder of this survey is organized into three parts. Figure 1 maps these three parts onto the sections that follow and shows the audit decision that recurs at every level. **Part I** establishes the foundations of agentic self-evolution in §2 by introducing the key concepts, formalizing the agent and improvement loop, and clarifying the relationships among adaptation, self-evolution, and RSI. **Part II** develops the L0–L4 taxonomy across §3–§7 and examines the evolution targets, feedback sources, update mechanisms, and reliability requirements at each level. Figure 3 maps the mechanism sections and representative works reviewed under these five evolution targets. **Part III** provides a cross-level reliability synthesis and research outlook. Section 8 synthesizes shared failure modes, examines external audit signals and promotion reliability, and analyzes boundary conditions under adversarial pressure, safety, and trust. Section 9 examines foundational limits and early-warning signals, the evaluation paradox under adaptive benchmarks, applications and reliability challenges, and the limits of external oversight under uncertainty, alignment, and human coordination. Section 10 concludes with a reliability-centered synthesis and a research agenda for agent-centric self-evolution. Together, these parts provide a unified taxonomy of how agents improve themselves and a framework for determining when such improvements can be trusted.

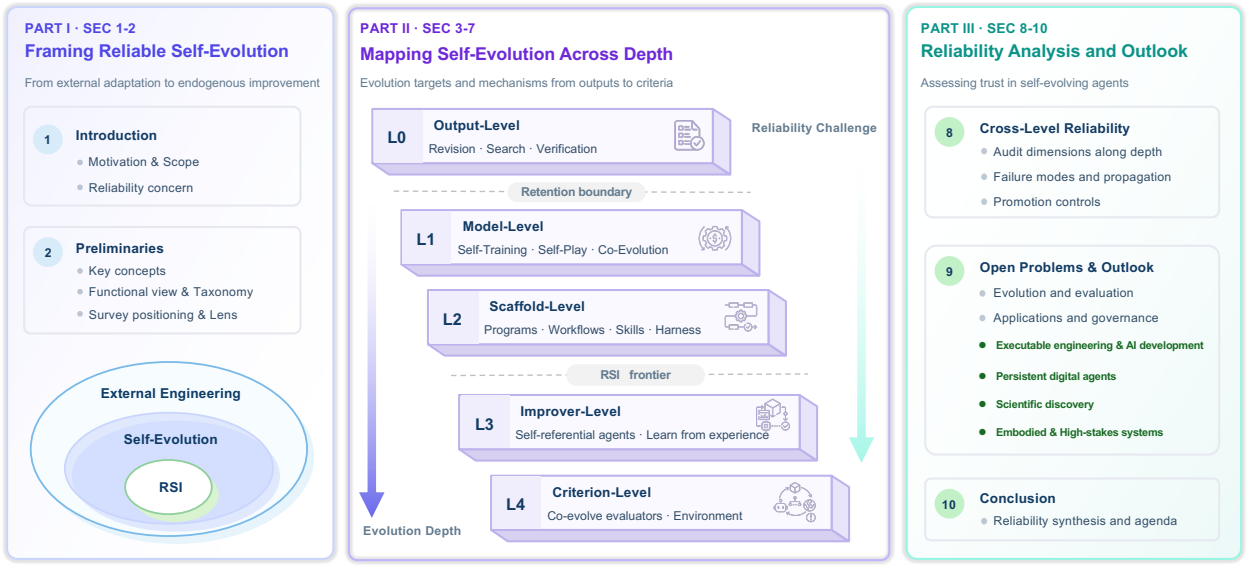


Figure 1 | **Organization of the survey.** The survey frames self-evolution and RSI (Part I), organizes methods from L0 to L4 by self-evolution depth (Part II), and analyzes reliability and open problems (Part III).

2 Preliminaries

This section introduces the concepts and compact agent-level model used throughout the survey. We first separate occurrences of self-evolution from claims of improvement, then define self-evolution depth and reliability. We next describe the agent state and update loop. Finally, we order the five evolution targets by self-evolution depth and position our survey relative to related surveys.

2.1 Definitions of Key Concepts

Agents may use their own outputs, interaction experience, or environmental feedback to improve their behavior or internal components. Prior work uses different terms for such processes [16, 18, 25, 26]. Throughout this survey, we consistently use *self-evolution*.

Self-evolving agents. We call an agent self-evolving when information generated during execution is fed back into the system. This feedback must revise either the current output or a retained component that affects later behavior or updating. The defining feature is not the number of execution steps, but whether information gathered during those steps leads to a further change. Within a single task, producing an answer is not enough. The agent must use feedback, evaluation, or search to revise, replace, or select candidate outputs [7, 27]. This loop, whether feedback and revision or verification and acceptance, does not by itself determine whether the resulting output is reliable. For a change to persist across tasks, storing a trace or a proposed update is not enough. The stored change must affect later behavior or future updates [28, 29].

Evolution target and self-evolution depth. The *evolution target* is the output or agent component changed by a transition. *Self-evolution depth* is determined by the deepest active semantic modification in a transition. This is the deepest target whose changed semantics affect a decision-relevant output, update, or judgment. For concise level-boundary equations, we write $A' \equiv_{\text{act}} A$ when two versions are equivalent in their active, decision-relevant semantics. The negation $A' \not\equiv_{\text{act}} A$ denotes a semantic change with the causal effect just

described. Depth therefore describes where a change occurs, not how large, irreversible, or reliable it is. When one transition changes several targets, we assign it to the deepest target whose semantic modification has such a causal effect. At L0, the change revises the task-local output or trajectory. At L1–L4, the change should affect later independent tasks or future updates. We call this the retention condition.

Reliability and improvement claims. A self-evolution event and an improvement claim are separate. The first states that a change occurred, while the second states that a measured aspect improved relative to the pre-update state. The evolution target asks what the system changed. The *external target* asks how we judge whether that change is better. The external target specifies the measured outcome, baseline, evaluation horizon, constraints, and matched resource conditions. The transition under evaluation does not modify these semantics. In this survey, *reliability* asks how strongly the available evidence supports the improvement claim under that target. A higher score alone is not enough because the loop may influence the proposed change, the evidence, or the retention rule.

External audit. An *external audit* assesses the improvement claim against the external target using evidence and a decision procedure outside the relevant update boundary. The *evidence source* produces results under declared data and procedures, while the *acceptance policy* maps those results to accept, reject, or escalate. The external target, evidence source, and acceptance policy should be versioned and remain outside the update boundary defined below [30]. This role differs from both the environment, which supplies observations and raw feedback, and the internal criterion, which uses rewards, rankings, metrics, or constraints to guide the update. The external audit assesses the resulting change using evidence such as held-out tests, executable checks, controlled experiments, formal verification, or human review. An audit is external because of how it is controlled, not where it runs. Its independence can weaken if the update can choose the evidence or repeatedly query the results.

2.2 A Functional View of Self-Evolving Agents

We now use a compact functional model to compare systems with different implementations. This view follows recent component-based accounts of self-evolving and lifelong agents [18, 25, 29]. The model keeps only the state, execution, retained updates, and audit needed for the taxonomy and reliability analysis.

Agent state and evolution targets. We model the retained agent state at iteration k with four components:

$$X_k = (\theta_k, \sigma_k, \mathcal{U}_k, C_k)\,. \tag{1}$$

The *update boundary* describes what the loop may modify, what evidence it may access, what resources it may use, and what actions it may authorize. Two systems may modify the same components yet differ in how strongly evidence can support their improvement claims, because their evidence access, resource limits, or authority differ. Here, θ_k denotes the trainable model or policy state. The scaffold σ_k is the retained, nonparametric runtime structure around the model. It includes prompts, tools, memory, workflows, and the runtime controls that shape execution and retained evidence. An object in this runtime structure belongs to σ_k by default. If its active role is to produce later updates or define later judgments, it belongs to the improver or criterion instead. We use $\sigma_k^{\mathrm{rt}} \subseteq \sigma_k$ to denote memory contents and indices that execution may update directly, while the outer loop controls the memory policy and schema. The improver $\mathcal{U}_k$ comprises the mechanisms that propose, select, commit, or roll back candidate modifications. The internal criterion C_k specifies the objectives, rewards, evaluation protocols, comparison rules, and constraints used to judge

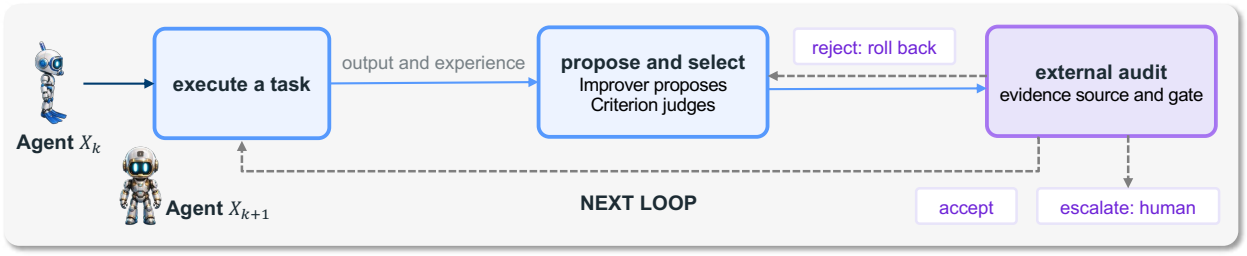


Figure 2 | **The self-evolution loop of an agent.** The agent runs a task and then proposes and selects a candidate change, which corresponds to Equations (3) and (4). The external audit of Equation (5) then decides among three outcomes: accept the candidate as X_{k+1}, reject it and roll back, or escalate it to a human. Its evidence source and acceptance gate stay outside the update boundary, so the loop cannot rewrite them. The dashed arrow closes the loop into the next round.

behavior and candidate states. Data and experience do not add a separate coordinate to X_k. They are classified with the retained component they modify. Learned router or evaluator parameters belong to θ_k, while nonparametric workflows and memory mechanisms belong to σ_k. Rubrics or evaluation protocols that govern later judgments belong to C_k.

The current output y_k is temporary and therefore sits outside the retained state X_k. For the taxonomy, we add this output to the four retained components and denote the five possible targets by

$$T_k^{\mathrm{evo}} = (y_k, \theta_k, \sigma_k, \mathcal{U}_k, C_k)\,. \tag{2}$$

Equation (2) lists the five taxonomy targets in order: output, model, scaffold, improver, and criterion.

Execution, proposal, and selection. At iteration k, the agent runs task q_k from state X_k under a declared resource budget b_k. Execution returns trajectory τ_k, output y_k, and a temporary experience summary e_k:

$$(\tau_k, y_k, e_k) \sim \mathrm{Exec}_{X_k}(\,\cdot \mid q_k, b_k). \tag{3}$$

An output-only revision changes y_k or its trajectory while leaving retained state unchanged, and is therefore L0. Execution may instead write to runtime memory σ_k^{rt} directly. Let X_k^{exec} denote the resulting retained state. If no runtime-memory update occurs, $X_k^{\mathrm{exec}} = X_k$. A runtime-memory update reaches Scaffold-Level Self-Evolution only when it stays active and changes behavior on a later independent task. Likewise, e_k affects later rounds only if it is stored in runtime memory or used in an accepted update. When a runtime-memory update is the only retained change, the round may end at $X_{k+1} = X_k^{\mathrm{exec}}$ without an outer update.

For an outer retained update, under the same implicit resource budget b_k, the current improver proposes candidates from X_k^{exec}, e_k, and evolution history $\mathcal{H}_k$. The history is maintained outside the update boundary, and the current criterion supplies the internal selection rules:

$$\begin{aligned} \mathbf{X}_k^{\mathrm{cand}} &\sim \mathrm{Propose}_{X_k^{\mathrm{exec}}}(\,\cdot \mid e_k, \mathcal{H}_k), \\ X_{k+1} &= \mathrm{Select}_{X_k^{\mathrm{exec}}}(\mathbf{X}_k^{\mathrm{cand}}; \mathcal{H}_k). \end{aligned} \tag{4}$$

Here $\mathbf{X}_k^{\mathrm{cand}}$ is a set of candidates that may modify any subset of the retained components. The selected X_{k+1} is provisional: internal selection determines which candidate proceeds to audit, not whether it is promoted.

External assessment instead uses evidence source $\mathcal{S}$ and acceptance policy $\mathcal{G}$ outside the agent state:

$$\begin{aligned} z_k^{\mathrm{ext}} &\sim \mathrm{Evidence}_{\mathcal{S}}(\,\cdot \mid X_k, X_{k+1}), \\ a_k^{\mathrm{ext}} &= \mathrm{Gate}_{\mathcal{G}}(z_k^{\mathrm{ext}}) \in \{\mathrm{accept}, \mathrm{reject}, \mathrm{escalate}\}. \end{aligned} \tag{5}$$

Table 1 | The L0–L4 taxonomy. Each level is defined by the deepest evolution target whose semantic modification causally affects a decision-relevant output, update, or judgment.

Level	Evolution target and boundary
Output-Level Self-Evolution (L0)	A task-local revision or selection changes the current output or trajectory, while the retained state X_k remains unchanged across independent tasks.
Model-Level Self-Evolution (L1)	A retained modification changes the trainable model or policy state θ_k and affects later task execution.
Scaffold-Level Self-Evolution (L2)	A retained modification changes the scaffold σ_k—such as its prompts, tools, memory operations, workflows, topology, or runtime harness—and affects later execution or retained evidence.
Improver-Level Self-Evolution (L3)	A retained modification changes the improver $\mathcal{U}_k$, altering how later candidates are proposed, selected, committed, or rolled back.
Criterion-Level Self-Evolution (L4)	A retained modification changes the criterion C_k—such as its objectives, rewards, evaluation protocols, comparison rules, or constraints—and governs later judgments after activation.

Here z_k^{ext} is the external evidence and a_k^{ext} is the resulting gate decision. The external target, $\mathcal{S}$, and $\mathcal{G}$ are controlled outside the update boundary of the transition being assessed. The audit uses the pre-execution state X_k as its baseline. It therefore covers both direct memory and outer updates unless they are declared and assessed separately.

Persistence, audit, and loop closure. Internal installation establishes a state transition, but does not by itself support an improvement claim. Without an external audit, the transition still counts as self-evolution, but no independent evidence supports an improvement claim. We use *promotion* for acceptance by the external gate, after which the candidate becomes the retained state for the next round. The audit may occur before installation or certify a provisional installation. An escalated case counts as accepted only after an authorized reviewer approves it. Acceptance means that the evidence meets the declared audit requirements. It is not a claim that the system improved in every respect. Rejection may trigger rollback. The state retained after any required rollback then conditions the next execution round and closes the self-evolution loop. Figure 2 summarizes this process.

2.3 The L0–L4 Taxonomy

We classify each transition by the deepest evolution target whose active semantic change affects a decision-relevant output, update, or judgment. Table 1 orders these targets by self-evolution depth. The levels are neither a temporal sequence nor a ranking of capability, sophistication, or reliability.

Classification rules. Classification follows the changed object rather than the algorithm that produces the change. Reflection, training, search, reinforcement learning, and evolutionary algorithms may therefore occur at several levels, and runtime use or storage alone does not establish a rewrite. When a transition changes several targets, the deepest causally active semantic change determines its level. If the available evidence does not establish that effect, we report the assignment as conditional. L0 is task-local, whereas L1–L4 must satisfy the retention condition. Retained data are classified by the deepest component whose later behavior they causally change. We call a system *L4-facing* when its core realized transition remains below L4 but it generates or uses criterion-related artifacts without changing the active judgment semantics.

Table 2 | Positioning our survey relative to related surveys. L0 denotes task-local output evolution. L1–L4 denote retained changes to the model, scaffold, improver, and criterion that affect later independent tasks or future updates. "Reliability" denotes explicit discussion of what improved and what evidence supports that claim. "External audit" denotes evaluation using evidence outside the control of the change being assessed. "Agent-level analysis" indicates whether a survey takes the agent as its unit of analysis. A dimension is marked ✓ when it is systematically covered or serves as a primary focus. The symbol △ indicates partial but explicit coverage, while ✗ indicates little or no coverage.

		Self-evolution depth					Reliability perspective		Agent-level
Survey	**Year**	**L0**	**L1**	**L2**	**L3**	**L4**	**Reliability**	**External audit**	**analysis**
Ours	–	✓	✓	✓	✓	✓	✓	✓	✓
General self-evolution surveys									
Ren et al. [17]	2026	✗	✓	✓	△	✗	✓	✓	✓
Jiang et al. [18]	2026	△	✓	✓	✓	△	△	△	✓
Gao et al. [26]	2026	✓	✓	✓	△	△	△	△	✓
Fang et al. [25]	2025	△	✓	✓	✗	△	△	△	✓
Tao et al. [16]	2024	✓	✓	△	✗	✗	△	△	△
Focused and related surveys									
Li et al. [23]	2026	△	△	✓	✗	△	✗	△	✓
Qi et al. [34]	2026	△	△	✓	✗	✗	△	△	✓
Zhang et al. [35]	2026	△	✓	✓	✗	△	△	△	✓
Du [36]	2026	✗	✗	✓	✗	✗	△	△	✓
Deng et al. [32]	2025	△	✓	✗	✗	✗	△	△	✗
Dong et al. [33]	2024	✓	✗	✗	✗	✗	△	△	✗
Liang et al. [19]	2024	✓	△	✗	✗	✗	△	✗	✗
Zheng et al. [20]	2024	✗	✓	✓	✗	✗	△	✗	✓
Zhang et al. [22]	2024	✗	✗	✓	✗	✗	✗	✗	✓

Relation to RSI. In this survey, *recursive self-improvement (RSI)* denotes a retained transition whose deepest active change reaches the improver or criterion. L3 changes how later modifications are proposed, selected, committed, or rolled back, while L4 also changes how behavior or candidate states are judged. This structural convention is not field-wide and neither establishes improvement nor implies accelerating gains. At these levels, an audit must establish which evidence remains outside the expanded update boundary.

2.4 Positioning Our Survey

Existing surveys characterize self-evolution research along several complementary dimensions. Broad surveys organize the literature by the evolved component, training stage, system architecture, feedback mechanism, or model–environment relation [16–18, 25, 26, 31]. Focused surveys instead examine topics such as inference-time self-evolution, multimodal models, agentic reinforcement learning, memory, environment engineering, and lifelong learning [19, 20, 22, 23, 32–36]. We add two questions: how deep does the change reach, and what evidence outside the relevant update boundary supports the improvement claim? These questions permit comparison across algorithms and architectures without treating depth as a reliability ranking. Table 2 compares the coverage of these two perspectives in our survey and related surveys.

Together, the taxonomy identifies what changes, while the reliability analysis asks what evidence supports the improvement claim.

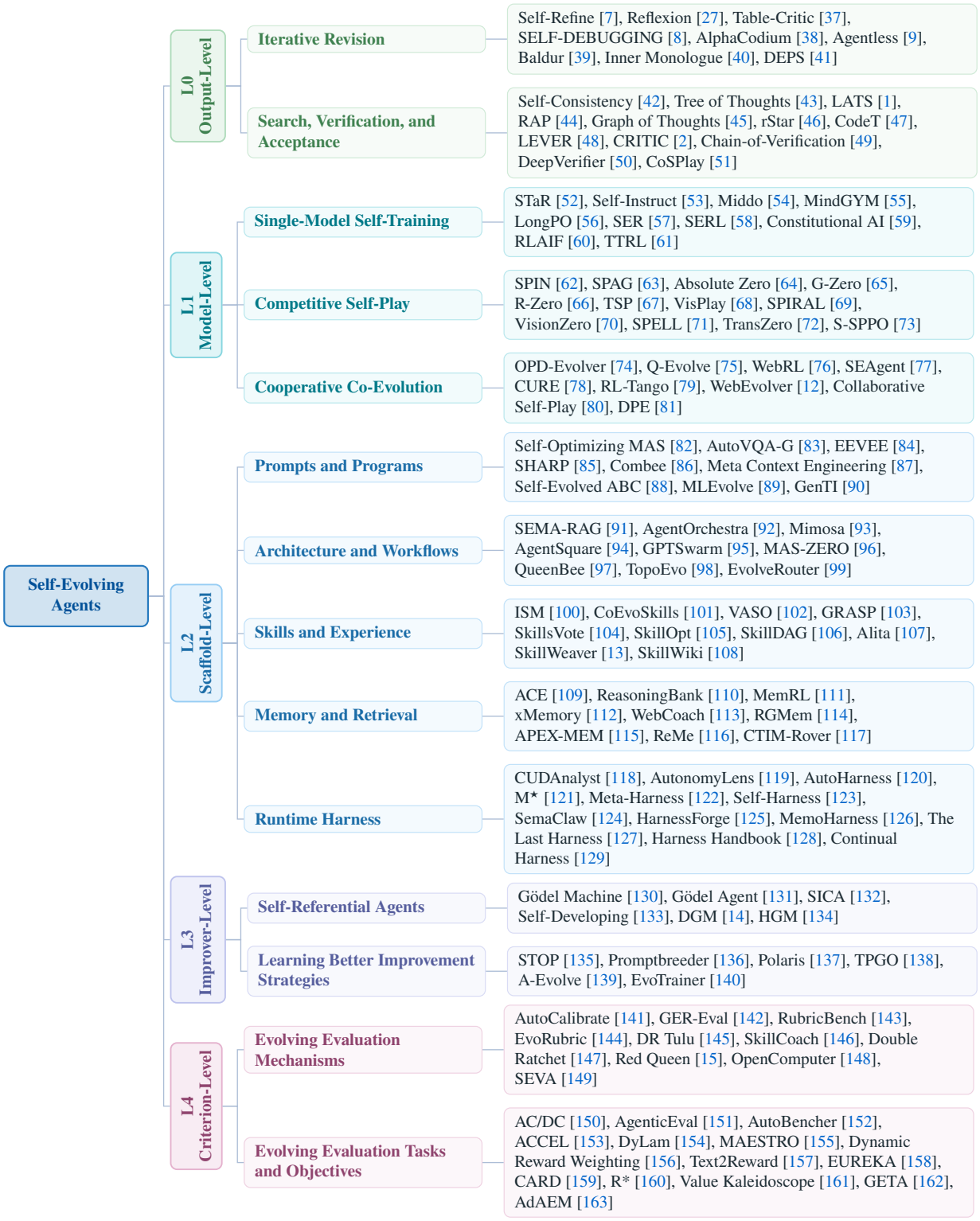


Figure 3 | **The L0–L4 taxonomy of agent self-evolution.** One branch per mechanism section of §3–§7, listing representative works reviewed under each evolution target; list length implies no ranking.

3 L0: Output-Level Self-Evolution

L0 at a Glance

Boundary. L0 changes only the current output y_k or its task-local trajectory τ_k. The retained agent state X_k remains unchanged across independent tasks, so any audit-supported improvement claim is task-local.

Roadmap. We follow the task-time control loop from iterative revision (§3.1) to search, verification, and acceptance (§3.2), then examine shared errors, coverage, stopping, and claim scope (§3.3).

Reliability focus. Revision and search can alter or enlarge the candidate set, but they do not establish improvement by themselves. Acceptance should rely on an external audit whose evidence source and acceptance policy remain outside the update boundary. The evidence must remain informative about the external target. Repeated access or shared blind spots can otherwise make the loop self-confirming.

Output-Level Self-Evolution is the task-time layer of the taxonomy: the agent may revise, replace, or select the current answer, rollout, code candidate, reconstruction, or reasoning trace, but the retained state used for later independent tasks remains unchanged. This section asks how task-time mechanisms produce candidate outputs, how evidence selects among them, and what scope of improvement that evidence can justify. We organize the section into three parts: *iterative revision* repairs one incumbent, *search, verification, and acceptance* expands and evaluates alternatives, and the *reliability and persistence* analysis defines the resulting claim boundary. Table 3 compares these three categories by their task-local role, representative mechanisms and systems, and reliability focus. Figure 4 follows the task-time path from a draft through reflection, exploration, and verification while retained state remains fixed.

Boundary and persistence. Writing the current trajectory, output, and retained agent state as (τ_k, y_k, X_k), the L0 boundary is

$$(\tau_k, y_k, X_k) \longmapsto (\tau'_k, y'_k, X'_k), \qquad (\tau'_k, y'_k) \not\equiv_{\text{act}} (\tau_k, y_k), \qquad X'_k \equiv_{\text{act}} X_k, \tag{6}$$

This is the L0 case: the active output or the trajectory that produces it changes, but no active retained state does. Critique, self-play, search, verification, and substantial temporary working state can all operate inside this boundary. For example, Focus compresses a long software-engineering trajectory into a compact Knowledge block and prunes earlier interactions so that the agent can continue solving the current task [164]. Such state can extend the effective reasoning horizon without becoming persistent self-evolution. The absence of persistence is both a containment property and a capability limit: a harmful revision is usually discarded with the episode, while a valid correction must be rediscovered on a later task.

3.1 Iterative Revision

Iterative revision maintains one incumbent candidate and repeatedly applies a feedback–repair loop. Its main design choice is the diagnostic signal: self-critique and role separation remain model-mediated, whereas execution, formal checks, environment observations, and localized evidence expose additional task structure.

3.1.1 Self-Critique and Role-Separated Feedback

The most direct revision loops generate feedback from the same model or from several model-mediated roles. Self-Refine repeatedly generates, critiques, and rewrites one candidate, with gains that vary substantially

Table 3 | Task-time mechanisms and limits of Output-Level Self-Evolution. Iterative revision works on one candidate, while search and verification generate and check alternatives. The final row summarizes the reliability concerns common to both and the task-local claims that L0 evidence can support. Examples are illustrative rather than exhaustive. Retained changes are classified separately by their deepest active semantic change.

Method	Task-local role	Representative mechanisms and systems	Reliability focus
§3.1 **Iterative Revision**	Rewrites one incumbent from model-mediated or task-grounded feedback.	• Self-critique and role-separated feedback: Self-Refine; Table-Critic [7, 37] • Execution-, test-, and proof-guided repair: SELF-DEBUGGING; Agentless; Baldur [8, 9, 39] • Environment and localized evidence: Inner Monologue; Kestrel [40, 165]	Shared blind spots or incomplete diagnostics can make revision preserve or introduce errors.
§3.2 **Search, Verification, and Acceptance**	Expands complete or partial candidates, then checks whether to return, reject, or escalate them.	• Sampling and structured search: Self-Consistency; Tree of Thoughts; LATS [1, 42, 43] • Consequence- and compute-aware search: WAC; START; CodeMonkeys [6, 166, 167] • Source-, tool-, and execution-guided verification: RARR; CRITIC; CodeT; LEVER [2, 47, 48, 168]	Candidate diversity helps only when the check covers the claimed property and does not reproduce the candidates' shared error.
§3.3 **Reliability and the Persistence Limit**	Bounds the claim to the selected output under the declared external target; retained state X_k remains unchanged.	• Intrinsic self-correction and self-confirmation [169] • Matched-budget self-repair and stronger feedback [170] • Evidence coverage, stopping, abstention, and escalation	The evidence can support a task-local output claim, not persistent agent improvement across independent tasks.

across code, dialogue, and mathematics [7]. Table-Critic instead separates error localization, critique, and revision across a Judge–Critic–Refiner cycle [37]. These designs make the repair process explicit, but neither repeated prompting nor role separation guarantees an independent diagnosis when the participants share the same criterion and blind spots.

3.1.2 Grounded Repair

Execution, tests, and proofs. Executable and formal systems narrow the repair set by returning behavioral or symbolic diagnostics. SELF-DEBUGGING revises programs from execution results, while AlphaCodium and Agentless combine reflection, localization, test construction, execution, and patch validation [8, 9, 38]. Baldur uses proof-assistant errors to repair Isabelle proofs, and AgentCoder separates programming, test design, and execution roles [39, 171]. These checks are more discriminating than unconstrained critique, although incomplete tests and specifications can still admit incorrect artifacts.

Environment and localized evidence. Interactive and multimodal systems ground revision in observations from the current task. Inner Monologue and DEPS revise plans from success signals, scene descriptions, human feedback, and observed failures, while Kestrel and Reflect-R1 retrieve localized visual or temporal evidence before revising an answer [40, 41, 165, 172]. Such evidence can expose errors that verbal self-critique misses, but its reliability still depends on whether observation and retrieval cover the claimed property. When one evolving trace cannot recover from an early commitment, L0 systems broaden the task-time state from an incumbent candidate to a set or structure of alternatives.

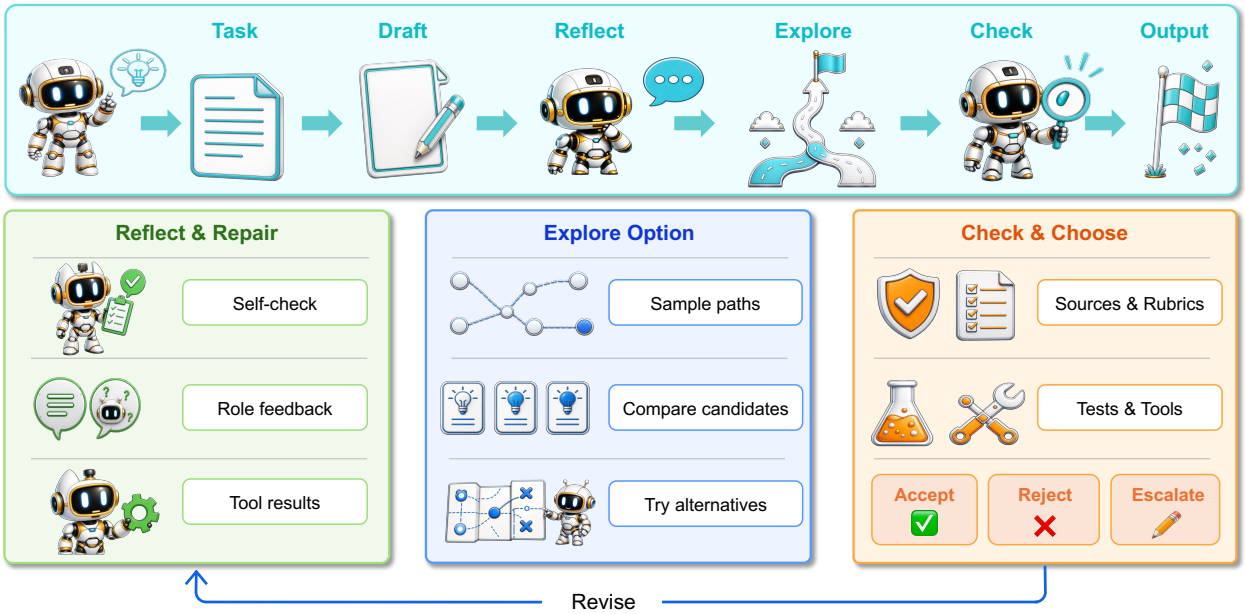


Figure 4 | **Task-local workflow of Output-Level Self-Evolution.** Starting from a task draft, reflection and repair use self-critique, role feedback, and tool results; exploration samples paths, compares ideas, and tries alternatives; and verification uses sources, rubrics, tests, and tools to accept, reject, or escalate the candidate.

3.2 Search, Verification, and Acceptance

Search expands the task-local candidate set, while verification and acceptance determine whether evidence warrants revision, rejection, or return. Flat sampling aggregates complete candidates, structured search controls intermediate states, and evidence-guided gates compare candidates against rubrics, retrieval, tools, execution, or internal proxies. These operations form one decision pipeline: candidate diversity is useful only when the acceptance signal can recognize a beneficial alternative without reproducing the search's shared error. The second row of Table 3 summarizes this pipeline from candidate generation through evidence-guided task-local decisions.

3.2.1 Candidate Generation and Structured Search

Sampling and search topologies. Sampling broadens the candidate pool without an explicit topology, while tree, graph, beam, and planning methods expose intermediate states for scoring, pruning, backtracking, or aggregation. Self-Consistency improves answer selection by marginalizing across sampled reasoning paths, whereas Tree of Thoughts and LATS organize explicit search over reasoning or agent trajectories [1, 42, 43]. Beam search, RAP, Graph of Thoughts, and rStar vary the topology and division of proposal and evaluation [44–46, 173]. These structures can diversify exploration and localize decisions, but agreement and model-derived value functions can still amplify errors shared across candidates.

Consequences, compute, and reuse. World-model systems compare predicted futures, whereas systems that execute tools can use observed consequences. WAC and WebDreamer simulate future states, while START executes and debugs tool-augmented traces [6, 166, 174]. Adaptive methods instead decide where inference should be spent. Fast–slow reasoning, difficulty-aware test-time scaling, CodeMonkeys, and S* allocate computation across uncertain cases or parallel edit–test trajectories [167, 175–177]. Buffer of

Thoughts and Forest-of-Thought reduce search cost through templates or sparse ensembles [178, 179]. Across these variants, additional computation helps only when predicted consequences are accurate, reused structure matches the task, and the selector recognizes informative alternatives.

3.2.2 Evidence-Guided Verification and Acceptance

Verification forms the outer task-time gate: it uses rubrics, retrieved evidence, tools, execution, tests, or internal consistency to return, reject, or escalate a candidate. Escalation transfers an unresolved case to an authorized reviewer or a stronger external verification process. The relevant distinction is not whether a check exists, but who controls its evidence and which errors it can observe.

Rubrics, sources, and tools. DeepVerifier and CoSPlay make acceptance criteria explicit through rubrics, critics, generated tests, and execution outcomes [50, 51]. RARR and Chain-of-Verification ground revision in separately retrieved or elicited factual evidence, while CRITIC generalizes tool-interactive checking to search engines and interpreters [2, 49, 168]. These channels can expose evidence absent from the initial output, but self-generated rubrics, tests, queries, and stopping rules remain vulnerable to shared omissions and adaptive overuse.

Execution and internal proxies. Execution-based methods compare observable behavior: CodeT ranks program–test consensus groups, MBR-EXEC groups programs by sampled behavior, LEVER combines execution with a learned verifier, and code-based self-verification uses an interpreter to check mathematical answers [47, 48, 180, 181]. When no executable or source-grounded check is available, Key Condition Verification, backward self-verification, and bidirectional manifold consistency construct reconstruction or consistency tests from the model's learned distribution [182–184]. Execution provides a checkable behavioral relation but remains limited by test coverage, while internal proxies can reject inconsistency without establishing correctness when the checking process shares the candidate's misconception.

3.3 Reliability and the Persistence Limit

The three task-time components expose a common reliability problem. Revision can repeat the generator's diagnosis, search can aggregate correlated candidates, and verification can accept a test or proxy that shares the proposer's blind spot. An L0 improvement claim is credible only if the audit measures the intended task outcome. The evidence and the decision rule must also remain outside the update's control.

Self-confirmation and negative evidence. The characteristic L0 failure is self-confirmation: additional reasoning endorses an error already present in the candidate, critic, candidate pool, generated test, or consistency score. Without oracle labels or external feedback, intrinsic self-correction reduces reported GPT-4 accuracy from 95.5% to 89.0% on GSM8K and from 49% to 43% on HotpotQA [169]. After accounting for model calls, self-repair gains are often modest or absent, while replacing self-feedback with human feedback raises the reported repair success rate from 33.3% to 52.6% [170]. These results do not show that task-time inference is ineffective. They show that more inference cannot substitute for evidence that distinguishes a repair from a newly introduced error.

Coverage and stopping. Evidence externality is necessary but not sufficient because every task-time check has a coverage boundary. Execution observes only tested behavior, retrieval observes only located sources, environment feedback observes only visited states, and fixed rubrics observe only encoded criteria.

Repeated access can turn a nominally external check into development feedback, so a defensible protocol couples each evidence source to stopping, abstention, escalation, and matched-budget rules. Candidate diversity and selector quality then determine whether additional search improves the returned artifact rather than increasing opportunities to overfit the check. The relevant comparison is between the accepted candidate and the incumbent under the same task, budget, and evidence protocol, not between systems that receive different amounts or kinds of feedback.

Claim scope and persistence handoff. When coverage and stopping are adequate, L0 evidence can support a claim about the selected output on the current task. Agreement and model-based critique are inexpensive but can reproduce a shared error, whereas execution, retrieved sources, environment observations, and fixed rubrics can provide more independent checks when their coverage is declared. None of these task-local checks by itself establishes that the retained agent will perform better on later independent tasks because Equation (6) leaves X_k unchanged. This absence of cross-episode retention contains the downstream effect of a harmful edit but also prevents a verified correction from accumulating. The next section turns to Model-Level Self-Evolution, where selected experience changes trainable state and thereby carries both improvements and errors into later tasks.

4 L1: Model-Level Self-Evolution

L1 at a Glance

Boundary. L1 persistently changes the trainable model or policy state θ_k, and the change affects later independent tasks. The scaffold σ_k, improver $\mathcal{U}_k$, and criterion C_k remain fixed. Any audit-supported improvement claim therefore concerns later behavior under the declared external target. The training signal may come from the agent's outputs, interaction experience, other models, or environmental feedback, but its source does not determine the level.

Roadmap. We organize the level by the structure of the training relation, moving from single-model self-training (§4.1) through competitive self-play (§4.2) to cooperative co-evolution (§4.3).

Reliability focus. Self-generated supervision, rewards, and peer signals can guide a model update, but they do not establish improvement under the external target. Promotion should compare the candidate with the incumbent under matched resource conditions, using an external audit outside the update boundary. Otherwise, low-quality labels, stalled self-play, or shared population bias can compound across rounds and contribute to collapse or drift.

Model-Level Self-Evolution is the first level whose retained rewrite lands on the trainable model state θ_k. The model may evolve its own weights or refine its own outputs, but it cannot evolve the structure or workflow around it, because the scaffold σ_k, improver $\mathcal{U}_k$, and criterion C_k stay fixed (Table 1). Because the update is kept in the weights rather than in the task-local output of L0, both useful changes and errors induced by self-generated supervision can carry over into later independent tasks. The core question is therefore under what conditions this signal remains informative about the external target and can support an improvement claim about the weight update. We organize the level by the structure of the training relation—who emits the signal and how that role differs from the trainee—rather than by the signal's surface form. Figure 5 shows this progression in signal-producing roles. Table 4 then indexes representative systems relation by relation, giving for each one the mechanism family it belongs to, the retained move it makes, and a public implementation where one exists.

Table 4 | **Representative works in Model-Level Self-Evolution.** Rows are grouped by the three training relations of §4.1–§4.3 and are representative rather than exhaustive. **Type** names the mechanism family, **Key mechanism** the retained move, and **Resource** a public implementation, where a dash means none was found.

Method	Type	Key mechanism	Resource
§4.1 **Single-model self-training**			
STaR [52]	Filtered self-output	Bootstraps reasoning from self-rationalized answers	GitHub
Self-Instruct [53]	Filtered self-output	Grows instructions from the model's own generations	GitHub
Middo [54]	Filtered self-output	Rewrites its fine-tuning set on competence axes	GitHub
MindGYM [55]	Synthesized supervision	Composes multi-hop questions for fine-tuning	GitHub
LongPO [56]	Synthesized supervision	Pairs long- and short-context responses to self-supervise	GitHub
SER [57]	Self-reward	Reward model relabels data to improve itself	GitHub
SERL [58]	Self-reward	Copeland-style pairwise self-ranking as reward	GitHub
Constitutional AI [59]	Self-generated alignment	Critiques and revises against fixed written principles	GitHub
RLAIF [60]	Self-generated alignment	Replaces the human labeler with an off-the-shelf model	–
TTRL [61]	Self-generated alignment	Majority-vote reward over unlabeled test-time samples	GitHub
§4.2 **Competitive self-play**			
SPIN [62]	Comparative self-play	Distinguishes own past generations from reference data	GitHub
SPAG [63]	Comparative self-play	Attacker and defender in an adversarial language game	GitHub
Absolute Zero [64]	Proposer-solver	Reasons from zero external data via a code executor	GitHub
G-Zero [65]	Proposer-solver	Drives reasoning from zero data under a code executor	GitHub
R-Zero [66]	Proposer-solver	Challenger and Solver co-evolve from zero human data	GitHub
TSP [67]	Proposer-solver	Turns unit tests into the solver's training signal	GitHub
VisPlay [68]	Proposer-solver	Questioner and reasoner co-evolve from unlabeled images	GitHub
SPIRAL [69]	Proposer-solver	Zero-sum reasoning games as the training signal	GitHub
VisionZero [70]	Proposer-solver	Competitive visual games over hidden state	GitHub
SPELL [71]	Proposer-solver	Self-play reinforcement for long-context evolution	GitHub
TransZero [72]	Proposer-solver	Self-play translation without parallel data	GitHub
S-SPPO [73]	Proposer-solver	Anneals the win-rate target toward maximum entropy	GitHub
§4.3 **Cooperative co-evolution**			
OPD-Evolver [74]	Privileged teacher	Internalizes memory competence by hindsight distillation	GitHub
Q-Evolve [75]	Co-evolving verifier	In-distribution critic emits per-step process rewards	GitHub
WebRL [76]	Co-evolving verifier	Outcome-supervised reward model drives the web agent	GitHub
SEAgent [77]	Co-evolving verifier	World-state critic emits the step-level reward	GitHub
CURE [78]	Two co-trained models	Coder and unit tester learn from each other's errors	GitHub
RL-Tango [79]	Two co-trained models	Generator and process verifier reinforced together	GitHub
WebEvolver [12]	Two co-trained models	World model predicts the feedback the policy learns from	GitHub
Collaborative Self-Play [80]	Population signal	Team reward internalized into one agent's policy	–
DPE [81]	Population signal	Multi-agent pipeline builds weakness-targeted samples	GitHub

Boundary and persistence. Model-Level Self-Evolution is a retained transition whose deepest active semantic change modifies the trainable model or policy state and affects later task execution. Within this boundary, we organize the surveyed methods by how the evolving system participates in producing the training signal. Writing the retained state as $X_k = (\theta_k, \sigma_k, \mathcal{U}_k, C_k)$, the L1 boundary is

$$X_k \longmapsto X_{k+1}, \qquad \theta_{k+1} \not\equiv_{\text{act}} \theta_k, \qquad (\sigma_{k+1}, \mathcal{U}_{k+1}, C_{k+1}) \equiv_{\text{act}} (\sigma_k, \mathcal{U}_k, C_k), \tag{7}$$

so a method belongs here whenever its retained update lands on the model parameters θ_k, whatever mechanism produces the signal. The training relation then forms one progression. It begins with a single self-training role, where producer and trainee coincide, and moves to two competing copies whose relative outcome may be more informative than a self-asserted label. It ends with cooperative helping roles that range from a privileged teacher to a whole population and differ from the trainee in information, function, parameters, or membership. Self-play that instead moves the target distribution is deferred to §7, since the deepest active change then reaches the criterion (§2.3). For the families surveyed here, the internal training signal may help produce a candidate update, but it does not by itself support an improvement claim under the external target. That claim requires an audit whose evidence source and acceptance policy remain outside the update boundary.

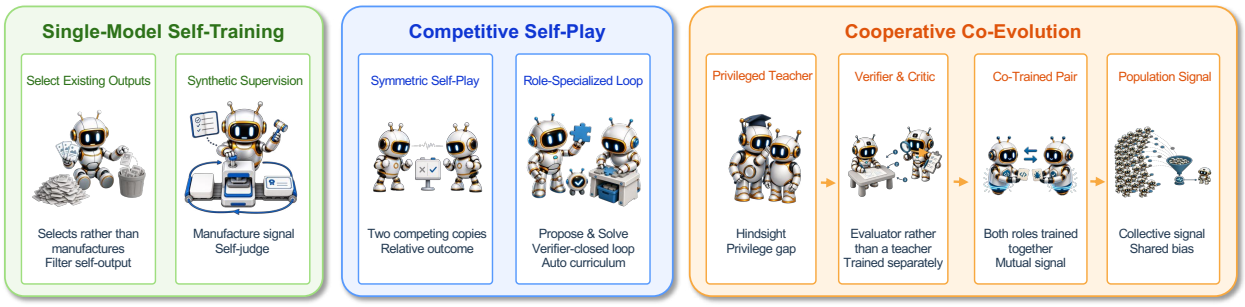


Figure 5 | The three training relations of Model-Level Self-Evolution. The panels distinguish a single self-training role, competing copies, and cooperative signal-producing roles. The left-to-right arrangement organizes who emits the training signal. It does not rank reliability or establish audit externality.

4.1 Single-Model Self-Training

This section collects the methods in which one model in a single role produces and consumes its own training signal, so producer and trainee coincide. We divide it by whether the signal is selected or created, separating methods that filter and curate outputs the model already produced from methods that must manufacture the supervision, reward, or alignment signal a task does not supply. The order distinguishes whether the signal comes from one role, competing roles, or cooperative roles. These arrangements expose different failure modes, but they do not by themselves determine reliability.

4.1.1 Filtering and Curating Existing Output

This part covers the base case, in which the model selects rather than manufactures its signal. The same model both produces and consumes the signal, and it retains only the high-confidence outputs it can filter and the data it can repair, so nothing enters the loop that the model did not already generate.

Bootstrapping from filtered self-output. The oldest and simplest way to close the loop on the weights lets a single model filter its own high-confidence outputs or distill its own trajectories and then write the result back into its parameters [52, 53, 185, 186]. STaR bootstraps reasoning by rationalizing the model's own correct answers, Self-Instruct grows an instruction-tuning corpus from its own generations, and ReST alternates growing a self-generated sample pool with reward-filtered fine-tuning [52, 53, 186]. Recent systems keep this single-role structure but vary the part of the training pipeline the self-generated signal targets [54, 187–189]. One line targets the *training data* itself. Middo rewrites its own fine-tuning set along complexity, diversity, and quality axes so that the data tracks the model's competence [54], and PolicyLong keeps the pool on-policy by re-filtering it with the current model before each update [187]. Another line distills a *persisted capability* into the weights without ground truth. SelfEvo cross-distills depth and pose from unlabeled video [188], while a further study asks when context-level experience can be internalized into parameters without the capability collapse that repeated internalization tends to induce [189].

Recurrence in label-scarce domains. The same filter-and-retrain loop is not confined to general reasoning. It recurs wherever a domain lacks cheap labels and the model must supply its own [190–192]. It appears in assembly-code comprehension under obfuscation [190], in streaming-video adaptation where the base model is its own data generator and annotator [191], and in medical slow-thinking bootstrapped from verifiable MCTS traces [192]. The most recent instances carry the same idea into still narrower settings, from embodied agents that internalize experience through inverse dynamics or idle-time updates [193–196] to self-collected

curricula for perception, coding, and retrieval [197–203]. Across these settings the grounding varies with the domain, but the loop's single-role structure does not.

Dynamics of the single-role loop. Because this loop is so widely reused, a methodological line studies its dynamics in their own right. This work examines how self-evolving training saturates and can be rebalanced [204], how a one-way policy update forms a ratchet without an external reference [205], and how self-consistency under geometric or visual invariance regularizes the signal [206, 207].

4.1.2 Synthesizing Supervision, Reward, and Alignment

This part covers the case in which no external label or teacher exists, so the single model manufactures the signal it will train on rather than selecting it from prior output. It draws that signal from synthesized tasks, self-generated critiques, and its own hindsight [55–57, 208–216], and the paragraphs below run outward from task supervision, through a self-produced reward, to a self-generated stand-in for human alignment feedback. Alignment is placed last because its manufactured signal targets values rather than task correctness, which puts it at the outer edge of what a single role can safely produce.

Synthesizing supervision targets. One line synthesizes explicit supervision targets. MindGYM composes cognitive multi-hop questions for thinking-centric fine-tuning [55], CoTEvol evolves chain-of-thought traces by genetic search before training on them [208], and LongPO pairs a model's long- and short-context responses to bootstrap long-context ability [56]. DITTO reports that the same self-comparison suffices for personalized alignment from fewer than ten demonstrations [209], and Learning-to-Label reinforces a self-evolving labeler that manufactures its own supervision targets [210].

Self-evaluation and retrospective reward. Another line draws the signal from the model's own evaluation or retrospection. Self-Rewarding Language Models use the evolving model as its own judge to build preference pairs for iterative alignment, and process-based self-rewarding extends this to step-wise judgments [211, 212]. SCRIT trains critique ability on self-generated contrastive critiques [213], and SER lets a reward model relabel data to improve itself [57]. RetroAgent turns hindsight into intrinsic numerical and language feedback for online updates [214], retrospective in-context learning converts sparse outcomes into dense advantage estimates for on-policy refinement [215], and reward-free world-knowledge exploration scores self-generated knowledge by its downstream success [216]. SERL closes the loop with the model as both actor and judge, and it derives its reward from Copeland-style pairwise self-ranking of its own generations with no external signal [58].

Shaping and stabilizing the self-reward. Because a self-produced reward is cheap but fragile, a further line shapes it to lower cost and collapse risk. Test-time synthesis and offline iteration reduce the compute a round demands [217, 218], latent-logic reward decomposition sharpens the credit signal [219], and easy-sample warm-up curricula stage what the model trains on first [220]. DARE adds a difficulty-adaptive curriculum whose difficulty signal is a self-normalized statistic over the model's own rollouts rather than a separate model, and the weight update is still driven by a fixed verifier [221].

Self-generated alignment. A parallel line applies the same single-role idea to alignment, replacing human preference labels with signal the model or a fixed rule produces [59–61, 222]. Constitutional AI critiques and revises its own responses against a fixed set of written principles and then trains a preference model

from the resulting AI feedback [59]. RLAIF shows that an off-the-shelf model can replace the human labeler in the RLHF pipeline [60]. TTRL pushes the idea to test time by deriving a reward from majority vote over the model's own samples on unlabeled data [61], whereas RLHI instead anchors the signal in real user interactions rather than the model's own distribution [222]. These methods update θ_k under a fixed criterion C_k, so they remain Model-Level even when the signal is evaluative. Activating a rubric or evaluator protocol whose judgment semantics have changed would instead make the realized transition Criterion-Level Self-Evolution (§7).

Across the whole single-role family the defining feature is the structural absence of a second party. Producer and trainee coincide, so nothing in the loop can contradict a confident error. This is the mechanism behind the self-consuming dynamics of §4.4, and it is the weakness that the next groups repair by reintroducing a second party into the loop.

4.2 Competitive Self-Play

This section covers the methods that reintroduce a second party as a competing copy, so the signal comes from a relative outcome rather than a self-asserted label. We divide it by how the competition is structured, starting with the symmetric comparative case and then the asymmetric proposer-solver schema that specializes the two roles. The second case is placed after the first because it adds a mechanism for keeping the competition informative rather than changing the underlying relation.

4.2.1 Symmetric Self-Play

The most direct way to reintroduce a second party is to let the model compete with a copy of itself and to take the training signal from the relative outcome of the competition [62–64, 69]. Self-play breaks the single-role closure by instantiating the model as two *competing* copies and reading the signal from a win or loss, a passed or failed test, or a preference between two responses. Because this relative signal is harder to fabricate than a label the model asserts about a single output, it is the first step away from the self-confirming loop. Two anchor cases make the mechanism explicit. SPIN converts a weak model into a strong one purely by having it distinguish its own earlier generations from reference data, with no extra annotation [62]. SPAG sharpens reasoning by casting the model as both attacker and defender in an adversarial language game [63].

4.2.2 The Proposer–Solver Schema

From the comparative base, the dominant instantiation specializes the two symmetric copies into a proposer rewarded for posing hard problems and a solver rewarded for answering them, with a verifier or consistency check closing the loop [64–66, 68, 223–225]. Its appeal is that the proposer manufactures the solver's curriculum automatically. The schema then recurs across domains according to what can play the verifier.

Executable verifiers. Where a code executor can adjudicate answers, Absolute Zero and G-Zero drive reasoning from zero external data [64, 65]. TSP and ACE instead turn unit tests or adversarial tests into the solver's signal [67, 223]. A pure instance of the schema is R-Zero, which starts from zero human data and co-evolves a Challenger rewarded for posing frontier-difficulty problems with a Solver trained on majority-vote pseudo-labels [66]. It updates both roles by reinforcement on the weights under a fixed reward.

Perceptual and cross-domain verifiers. Where the judgment is perceptual rather than executable, the same loop is closed by self-consistency or a learned check in multimodal and video understanding [224, 226–228]. VisPlay illustrates the perceptual case, pairing an image-conditioned questioner with a multimodal

reasoner and co-evolving them from unlabeled images alone [68]. Spatial and geospatial reasoning instead ground the check on deterministic geometry or executable programs [229, 230]. The schema extends further to instruction following [231], search grounded in knowledge graphs [232], translation without parallel data [72], and affective dialogue [233]. It also scales from single turns into multi-turn and multi-agent games, which include long-context evolution [71], zero-sum reasoning games [69], and competitive visual games in which the roles must reason about hidden state [70].

Stretching the competitive relation. Two variants stretch the competitive relation to its limit without abandoning it. An *error-driven* opponent is actively rewarded for producing wrong outputs so that the solver learns to discriminate [234, 235]. An *adversarial discriminator* instead supplies dense step-level judgments against the reasoner [236]. The relation stretches further to full self-play defect injection-and-repair for software agents [237] and to intrinsic-supervision proposer-solver loops that reward the whole trajectory rather than only the final answer [238].

Keeping the competitive signal live. Because the competitive signal is only as informative as the competition is live, a substantial line of work turns from applying the schema to *stabilizing* it, and these methods are best read as patches for its specific failure modes [73, 239–243]. When the preference oracle grows overconfident on semantically similar responses, S-SPPO anneals the win-rate target toward a maximum-entropy baseline [73]. When the current advantage vanishes across iterations, TPAW plays the policy against its own historical checkpoints and T-SPIN adds a historical-advantage triplet [239, 240]. When the game becomes deterministic and the gradient dies, DEPT detects the resulting "evolutionary impasse" and reshapes the advantage [241]. When the proposer's distribution narrows and the curriculum collapses, vocabulary dropout forces diversity [242]. When only relative reward is optimized, SPACE adds noise-contrastive estimation to anchor the absolute values and provably converge to the true distribution [243]. Foresight over an opponent [244], continuously tunable Rényi-divergence objectives [245], and guided asymmetric curricula anchored on benchmark problems [246] extend the same stabilizing agenda. A game-theoretic line further reframes self-play preference optimization as convergence to a Nash equilibrium [247, 248]. These stabilization methods delimit when the comparative signal remains informative. The comparative signal advances beyond self-training only while the competition keeps injecting learnable information, and it decays into noise once the two copies stop meaningfully disagreeing. That failure is the one §4.4 analyzes as the level's ceiling [249, 250].

4.3 Cooperative Co-Evolution

This section covers the methods in which a functionally distinct role co-evolves to help the trainee rather than to defeat it. We divide it by how the helping role differs from the trainee, including privileged information, a distinct evaluation function, separately trained parameters, or population membership. These distinctions may affect signal informativeness, but they do not establish external audit status. The four parts therefore run from a privileged teacher, through a co-evolving verifier, to two mutually trained models, and finally to a whole population.

4.3.1 Privileged Teachers

The near end of the cooperative range is a teacher that differs from the trainee only by a privilege gap, such as access to hindsight, distilled skills, or trajectory context the student lacks. Because the teacher remains closely coupled to the trainee, its additional information comes from that privilege gap rather than from genuine audit independence.

Teachers with a privilege gap. A third group keeps two roles but reverses their relation, so that a functionally distinct role co-evolves to help the trainee rather than to defeat it [75, 78, 79, 251]. At the near end, that role is a *teacher* conditioned on information the student lacks, such as distilled skills, hindsight, or privileged trajectory context. It supplies dense token-level supervision on the student's own rollouts, so the student internalizes guidance it could not have produced alone. GenEvolve abstracts best-versus-worst trajectory differences into privileged visual experience [252]. OPD-Evolver internalizes read, use, write, and maintain memory competence through privileged-hindsight distillation [74]. π-Play uses problem-construction paths as the teacher's privileged context [253]. VPD casts the teacher as an actively refined variational posterior over language feedback [254].

4.3.2 Co-Evolving Verifiers and Critics

The second role may instead become an *evaluator* rather than a teacher. A separately trained evaluator can contribute a signal with different errors or information from the generator, although separate training alone does not make that signal an external audit. Q-Evolve learns an in-distribution critic that emits per-step process rewards [75]. CME goes furthest by scoring the generator under an *independent* verifier model whose signal cannot be gamed by self-consistency [251]. Self-Guide and MAESTRO learn an internal reward or scalarization jointly with the policy [155, 255]. A visual critic co-evolves with a proposer to sharpen GUI grounding [256]. DPA-GRPO lets a verifier issue safety-assurance cases that ground the generator's revisions [257]. In agent settings the evaluator is often a full reward model trained apart from the policy. WebRL learns an outcome-supervised reward model whose binary success signal drives the web agent's update [76]. SEAgent fine-tunes a World-State-Model critic that labels each action and emits the step-level reward the computer-use agent trains on [77].

4.3.3 Two Co-Trained Models

When both roles are trained together, the question becomes what distinguishes genuine cooperation from a relabeled single-role loop. At the far end, both roles become full trainable models that supply each other's signal. A world model and a policy co-evolve so that the model predicts the feedback the policy learns from [12, 258]. Two networks with complementary strengths tutor each other on-policy [259–262]. A coder and a unit tester co-evolve so that the tester learns directly from the coder's errors [78, 263], and ZeroCoder rewards the coder and tester for reaching consensus rather than for defeating each other [264]. A generator and a *generative*, process-level verifier are reinforced together to resist the reward hacking that fixed verifiers invite [79]. An extractor and a solver are jointly optimized so that experience is both found and used [265]. An agentic recommender and a user model co-evolve from a shared interaction reward [266]. A skills-manager and a worker share one policy across a reasoning hierarchy [267]. Cooperative agentic pipelines with dedicated planner, verifier, and generator modules belong here as well [268], as do solver and reframer schemes whose second role reframes the input to *harden* the pseudo-label rather than to defeat the solver [269]. Across this range, the recurring design is the same, because a second model is trained precisely so that its signal remains useful to the first. This deliberate coupling, rather than the diversity of the pairings, makes the group a single mechanism rather than a catalogue.

The cooperative-objective test. The cooperative-objective test that defines this group is also what separates it from self-play at the boundary. DUEL, for instance, builds a challenger that constructs hard negatives, which reads as adversarial. Its purpose, however, is to *stabilize* the solver's visual grounding rather than to beat it, so we place it here and flag the ambiguity [270]. The general point is that this group differs from self-play not through the mere presence of a second role, but through a cooperative objective that makes the second role supply guidance to the trainee. An independently trained verifier may provide more informative

guidance when its errors differ from the trainee's, but its control and query permissions determine whether it can also participate in an external audit.

4.3.4 Multi-Agent and Population Signal

The helping role reaches its widest form when a whole group, rather than a single partner, produces the training signal for one model. This is the far end of the cooperative range, since a group can average out an individual helper's blind spots but also risks a shared bias that no member is positioned to correct. It is separated from the co-trained case because the reliability question shifts from the alignment of one partner to the diversity of the collective.

Forms of collective signal. The signal is then manufactured not by one role or a dyad but by a collective whose incentive structure shapes what the single trained model learns [80, 271–273]. The forms this takes vary with what the collective is asked to produce, but each extends the cooperative idea from two parties to many. SERM assigns a multi-agent miner-and-annotator pipeline to detect distribution shift and emit reliable labels for a single relevance model [271]. DPE runs the same multi-agent recipe for a multimodal learner. Multiple agents annotate and quality-control unlabeled data with tools such as web search and image editing, generating weakness-targeted samples that a single diagnosed model trains on across successive rounds [81]. ProDa likewise builds its fine-tuning corpus with a multi-LLM pipeline of an extractor, a synthesizer, an LLM judge, and a debugger, so that the signal a separate target model trains on is produced and repaired entirely by other models [274]. Collaborative self-play rewards a team only when it collectively reaches the correct answer. Metacognitive knowledge about when to rely on parameters, when to rely on tools, and when to abstain emerges from this group incentive and is then internalized into one agent's policy [80]. Multi-agent deliberation has personas negotiate to synthesize a collective-alignment signal [272]. PopuLoRA replaces single-model self-calibration with cross-subpopulation evaluation over a population of LoRA adapters and weight-space evolutionary operators [273].

Error-averaging and its limit. The premise that motivates them all is that a collective can denoise the idiosyncratic errors that trap a lone self-trainer, since a bias one agent holds need not be shared by the group. The promise is thus a form of error-averaging that a single model cannot achieve from inside itself. The risk is that this averaging holds only while the collective's consensus tracks the truth rather than a bias the whole group happens to share. A further risk is that the consensus itself becomes an optimization target the training loop can learn to satisfy without becoming more correct. The group therefore does not escape the level's ceiling. It relocates the point at which the self-generated signal can decouple from truth, moving it from a single model's confidence to a population's agreement.

4.4 Reliability and the Fixed-Scaffold Limit

Having traced the three groups from a single self-teaching role out to a whole population, we can now state precisely what each one buys and what none of them can. The three relations can be placed side by side by the identity and function of the signal-producing party, each paired with the failure mode it characteristically incurs. The comparison identifies the signal source available to each group and the condition under which that signal remains trustworthy.

Signal source. In the families surveyed here, the training signal is a model-produced label, reward, or preference. The three groups vary only in who produces it, namely a single role in self-training, a competitor

in self-play, and a cooperating teacher, critic, or population in cooperative co-evolution. In no case is this signal itself the external target. The groups differ in whether the signal comes from the trainee itself, an opponent, a cooperative role, or a population.

Condition for reliability. In every case, the improvement claim is supported only to the extent that the internal training signal predicts outcomes under the external target. Self-training meets it only while the model's confident outputs happen to be correct, because nothing in its loop can contradict a confident error. Self-play meets it while the competition keeps injecting learnable information, and it fails when the two copies stop disagreeing. Cooperative co-evolution can add privileged information, distinct errors, or complementary roles, but those properties do not guarantee correctness. Population signal meets it while the collective consensus tracks truth rather than a shared bias. The comparison therefore organizes internal signal sources rather than ranking reliability. Reliability still depends on target-matched external evidence and on who controls the evidence source and acceptance policy.

When the reliability condition breaks, the loop amplifies its own error, and even a stable loop still leaves a fixed scaffold that L1 cannot rewrite.

Failure mode. When the condition fails, self-generated noise is amplified instead of corrected, and the resulting degenerations are the empirical signature of this level rather than incidental bugs. Under self-consumption the effect appears first as model collapse, which is formalized as a change of scaling laws [275]. It also appears as tail-narrowing stagnation, which arises when easy samples are over-sampled and hard ones are starved [276]. A further form is "superficial" self-improvement, which lifts in-distribution accuracy while eroding out-of-distribution generalization [277]. A final form is capability erosion under lifelong adaptation [278]. For self-play specifically, the failure takes the form of stalling. Stalling occurs when the self-synthetic pipeline stops adding learnable information [249], or when data gating and reward grounding no longer hold the loop stable [250]. A complementary line of work explains why the condition is fragile rather than merely reporting that it breaks. One result bounds how closely self-generated supervision can approach oracle supervision [279]. Another shows that problem-solving RL implicitly induces the very process-reward capability it appeared to presuppose [280]. A third stabilizes learning against uncertain feedback through confidence orchestration [281]. At the most abstract, a further group models self-referential recoverability collapse as a structural consequence of capacity saturation [282–285]. Read together, these analyses explain why the reliability condition above cannot be assumed to hold.

The next ceiling. Two limits force the descent to the next level, and they operate even when the reliability condition does hold. The first is that the model remains trapped in its own distribution, so that retraining is both costly and collapse-prone, which caps how far weight updates alone can carry the system. The second limit is the decisive one. The *scaffold* around the model, meaning its prompts, tools, memory, and control flow, has stayed hand-coded and fixed throughout this level, and it is frequently the true bottleneck on what the trained weights can express. To break that bottleneck the loop must step outside the model body, so that the object of improvement becomes the scaffold itself rather than the weights it surrounds, which is the move to L2 (§5). Its price continues the pattern established here, because the per-sample self-generated referent of this level is surrendered in turn, leaving end-to-end behavior as the sole remaining observable against which improvement can be judged.

5 L2: Scaffold-Level Self-Evolution

L2 at a Glance

Boundary. L2 persistently changes the scaffold σ_k, including prompts, tools, memory, workflows, topology, skills, and runtime controls. The model state θ_k may also change, but the scaffold remains the deepest active evolution target. The improver $\mathcal{U}_k$ and criterion C_k remain fixed. Any audit-supported improvement claim therefore concerns later end-to-end behavior under the declared external target.

Roadmap. We organize the level by the scope of the mutable scaffold, moving from prompts and programs (§5.1) to architecture and workflows (§5.2), skills and experience (§5.3), memory and retrieval (§5.4), and the runtime harness (§5.5).

Reliability focus. Local inspection, compilation, and regression tests can screen individual edits, but they do not establish end-to-end improvement. Promotion should compare the candidate and incumbent on representative later tasks under matched resource conditions, using an external audit outside the update boundary. Versioned provenance and a rollback path should accompany deployment. Component interactions, repeated benchmark access, or silent memory drift can otherwise make the scaffold overfit the development setting.

Scaffold-Level Self-Evolution is the level whose deepest retained rewrite lands on the scaffold σ_k, meaning the prompts, tools, memory policies, communication graphs, and runtime harness around the model, while the improver $\mathcal{U}_k$ and the criterion C_k stay fixed (Table 1). The model parameters θ_k may also be updated, but they are not the deepest change, so an improvement is consolidated in σ_k and must be assessed against the external target defined in §2.1. The evidence coarsens from per-example supervision to aggregate end-to-end behavior, so the operative question is no longer whether one self-generated example is a valid target, but whether the revised agent behaves better over a representative task distribution. We organize the level by the *scope* of the mutable object and widen the scaffold one layer at a time, so that each layer widens the causal footprint of an edit and presupposes the objects below it. As the scope widens from a compilable program to a silently retrieved memory entry, the local checks weaken and certification leans increasingly on an end-to-end comparison against the incumbent scaffold. This places the Scaffold-Level between the local supervision of the Model-Level and the update-rule changes of the Improver-Level. Table 5 aligns each method category with its mechanisms and with the check and characteristic failure that govern how hard it is to certify. Figure 6 fixes the cumulative widening of scope from a single artifact to the loop that runs every other layer. Table 6 then indexes representative systems layer by layer, giving for each one the mechanism family it belongs to, the retained move it makes, and a public implementation where one exists.

Boundary and persistence. At L2 (Table 1), the deepest retained rewrite is the scaffold σ_k, meaning system prompts, rule files, executable tools, workflow graphs, role pools, routing policies, memory schemas, skill libraries, validators, and runtime controls, while the improver and criterion remain fixed. Writing the retained state as $X_k = (\theta_k, \sigma_k, \mathcal{U}_k, C_k)$, the L2 boundary is

$$X_k \longmapsto X_{k+1}, \qquad \sigma_{k+1} \not\equiv_{\text{act}} \sigma_k, \qquad (\mathcal{U}_{k+1}, C_{k+1}) \equiv_{\text{act}} (\mathcal{U}_k, C_k), \tag{8}$$

so a method belongs here whenever the scaffold is the deepest retained change, even when the model parameters θ_k are updated as well, provided the improver and criterion stay fixed. Agentic-RL roadmaps and persistent-agent architectures motivate this scope by treating trajectory protocols, workspaces, skills, and evolution control as first-class parts of learning systems [287–290]. The object rule keeps mixed systems

Table 5 | Mutable objects of Scaffold-Level Self-Evolution. Method categories are ordered by the scope of the rewritten object, which widens cumulatively so that each category presupposes the ones below it. The reliability column pairs the strongest locally available check with the failure that check cannot catch. As the object widens, the check weakens and certification leans further on an end-to-end comparison against the incumbent scaffold. Mechanism lists are representative rather than exhaustive, and the final row records the cross-category synthesis rather than a sixth mutable object.

Method	Representative mechanisms	Reliability focus
§5.1 **Prompts and programs**	• Control-surface rewriting • Typed, auditable artifacts • Program discovery	Compilation, unit tests, and a readable diff of one artifact. *Failure:* weak attribution, because one reworded clause can shift tool use, evidence selection, and stopping at once.
§5.2 **Architecture and workflows**	• Staged roles with verifiers • Architecture search • Communication and routing	Per-stage verifiers and ablations over roles or edges. *Failure:* a gain may reflect extra coordination or compute rather than a structure that transfers.
§5.3 **Skills and experience**	• Trajectory distillation • Verifiable skill programs • Lifecycle governance	Held-out probes under a regression budget on invoked entries. *Failure:* silent regression, where an added entry improves its target cases and breaks previously correct behavior.
§5.4 **Memory and retrieval**	• Typed writes • Adaptive write/recall • Deletion as a learning rule	Replay against retained raw episodes over long horizons. *Failure:* unfiltered accumulation degrades behavior, the scaffold analogue of Model-Level self-consumption.
§5.5 **Runtime harness**	• Feedback-channel study • Harness as durable program • Governing every layer	Execution of the loop together with a recorded rollback path. *Failure:* an acceptance gate the loop can query repeatedly stops being held-out evidence.
§5.6 **Reliability and fixed-improver limit**	• Scaffold overfitting • Bounded lifecycle • Fixed-improver handoff	Controlled end-to-end comparison against the incumbent scaffold on tasks the edit did not select [286]. *Failure:* scaffold overfitting to benchmark, judge, memory, or router quirks.

separable. A learned router, adapter, or verifier is a Model-Level update only when its parameters are retained, whereas an external symbolic policy or validation program stays here. Promoting a new rubric or evaluator protocol instead reaches the Criterion-Level, and retained state that changes how later edits are proposed reaches the Improver-Level. Because the evidence has coarsened from a per-example label to end-to-end behavior, a proposed scaffold must transfer beyond the examples that prompted it, improve beyond a single successful interaction, and keep enough provenance to attribute later regressions. Reliable comparison therefore needs matched budgets, an incumbent scaffold, ablations, and fresh probes rather than a single score.

5.1 Prompts and Programs

Having fixed the scaffold boundary and its behavioral-evidence standard, we begin the scaffold layering at the smallest persistent scaffold: a single text or code artifact revised while the base model stays frozen. A prompt clause, rubric, or program is the most inspectable and auditable editable object, because it is local and, for code, also compilable and testable. It precedes the architecture layer, which composes such artifacts into interacting roles and control flow.

5.1.1 Editing Prompts and Text

This first part covers the smallest scaffold edit, a rewrite of natural-language control surfaces and other human-readable artifacts while the model stays fixed. We separate it from program editing because a text edit is inspected by reading rather than by compiling, so its reliability rests on human judgment rather than on an executable check.

Table 6 | Representative works in Scaffold-Level Self-Evolution. Rows are grouped by the five layers of §5.1–§5.5 and are representative rather than exhaustive. **Type** names the mechanism family, **Key mechanism** the retained move, and **Resource** a public implementation, where a dash means none was found rather than that none exists.

Method	Type	Key mechanism	Resource
§5.1 Prompt and code			
Self-Optimizing MAS [82]	Control-surface rewriting	Searches orchestration prompts by self-play	–
AutoVQA-G [83]	Control-surface rewriting	Refines annotation prompts from checked critique	GitHub
EEVEE [84]	Control-surface rewriting	Co-evolves a task router with per-cluster prompts	GitHub
SHARP [85]	Typed, auditable artifact	Atomic edits to a condition-action rubric	–
Combee [86]	Typed, auditable artifact	Parallel prompt learning from pooled trajectories	–
Meta Context Engineering [87]	Typed, auditable artifact	Context skills co-evolve with the files they emit	GitHub
Self-Evolved ABC [88]	Executable artifact	Frozen model rewrites a logic-synthesis system	–
MLEvolve [89]	Executable artifact	Tree-searched pipelines with retrospective memory	GitHub
GenTI [90]	Executable artifact	Chain-of-verification intrusion-rule generation	–
§5.2 Agent architecture			
SEMA-RAG [91]	Roles and orchestration	Separates interpretation, exploration, adjudication	–
AgentOrchestra [92]	Roles and orchestration	Conductor retains memory, tools, and prompts	–
Mimosa [93]	Roles and orchestration	Meta-orchestrator scores generated topologies	GitHub
AgentSquare [94]	Searched architecture	Searches a modular planning and memory space	GitHub
GPTSwarm [95]	Searched architecture	Optimizes node prompts and rewires edges	GitHub
MAS-Zero [96]	Searched architecture	Per-instance design, retained only if carried forward	GitHub
QueenBee [97]	Communication and routing	Inter-agent DAG as a retrievable design skill	GitHub
TopoEvo [98]	Communication and routing	Co-adapts topology edges at test time	GitHub
EvolveRouter [99]	Communication and routing	Co-evolves the routing policy with prompts	GitHub
§5.3 Skill library			
ISM [100]	Formation and verification	Distills strategies into a retrievable store	GitHub
CoEvoSkills [101]	Formation and verification	Co-evolving verifiers vet multi-file skill packages	GitHub
VASO [102]	Formation and verification	Counterexample traces revise a skill contract	–
GRASP [103]	Lifecycle governance	Admits a skill only on gated held-out net gain	GitHub
SkillsVote [104]	Lifecycle governance	Create, improve, merge, retire with provenance	GitHub
SkillOpt [105]	Lifecycle governance	Bounded skill edits admitted on held-out gain	GitHub
SkillDAG [106]	Lifecycle governance	Typed edges registered on execution evidence	GitHub
Alita [107]	Sharing and reuse	Forges and persists callable MCP tools	GitHub
SkillWeaver [13]	Sharing and reuse	Discovers and hones reusable web APIs	GitHub
SkillWiki [108]	Sharing and reuse	Provenance-bearing skills in a living knowledge base	GitHub
§5.4 Memory store			
ACE [109]	Write and recall policy	Incremental edits avoid context collapse	GitHub
ReasoningBank [110]	Write and recall policy	Distills strategies from successes and failures	GitHub
MemRL [111]	Write and recall policy	Non-parametric reinforcement over episodic memory	GitHub
xMemory [112]	Structure and silent injection	Revisable fragment-to-group hierarchy	GitHub
WebCoach [113]	Structure and silent injection	Injects cross-session advice via runtime hooks	GitHub
RGMem [114]	Structure and silent injection	Coarse-grains episodes into stable user profiles	GitHub
APEX-MEM [115]	Provenance and deletion	Append-only graph resolves conflicts at read time	–
ReMe [116]	Provenance and deletion	Utility-based refinement removes stale entries	GitHub
CTIM-Rover [117]	Provenance and deletion	Retained memory did not beat the memoryless baseline	GitHub
§5.5 Runtime harness			
CUDAnalyst [118]	Harness as peripheral object	Attributes plan decisions to feedback components	GitHub
AutonomyLens [119]	Harness as peripheral object	Synthesizes new tests from observed failures	–
AutoHarness [120]	Harness as peripheral object	Synthesizes guard code around a frozen agent	–
M* [121]	Harness as durable program	Builds a per-task executable memory loop	GitHub
Meta-Harness [122]	Harness as durable program	Searches full run history, logs, and source	GitHub
Self-Harness [123]	Harness as durable program	Admits an edit only after regression verification	–
SemaClaw [124]	Harness as durable program	Two-phase orchestration with a permission layer	GitHub
HarnessForge [125]	Harness as durable program	Co-adapts harness with the reasoning policy	GitHub
MemoHarness [126]	Scaling and navigation	Adjusts six harness control dimensions per case	GitHub
The Last Harness [127]	Scaling and navigation	Per-task loop plus cross-task meta-evolution	–
Harness Handbook [128]	Scaling and navigation	Behavior-centric map of a harness codebase	GitHub
Continual Harness [129]	Scaling and navigation	Reset-free loop refines prompts, skills, and memory	GitHub

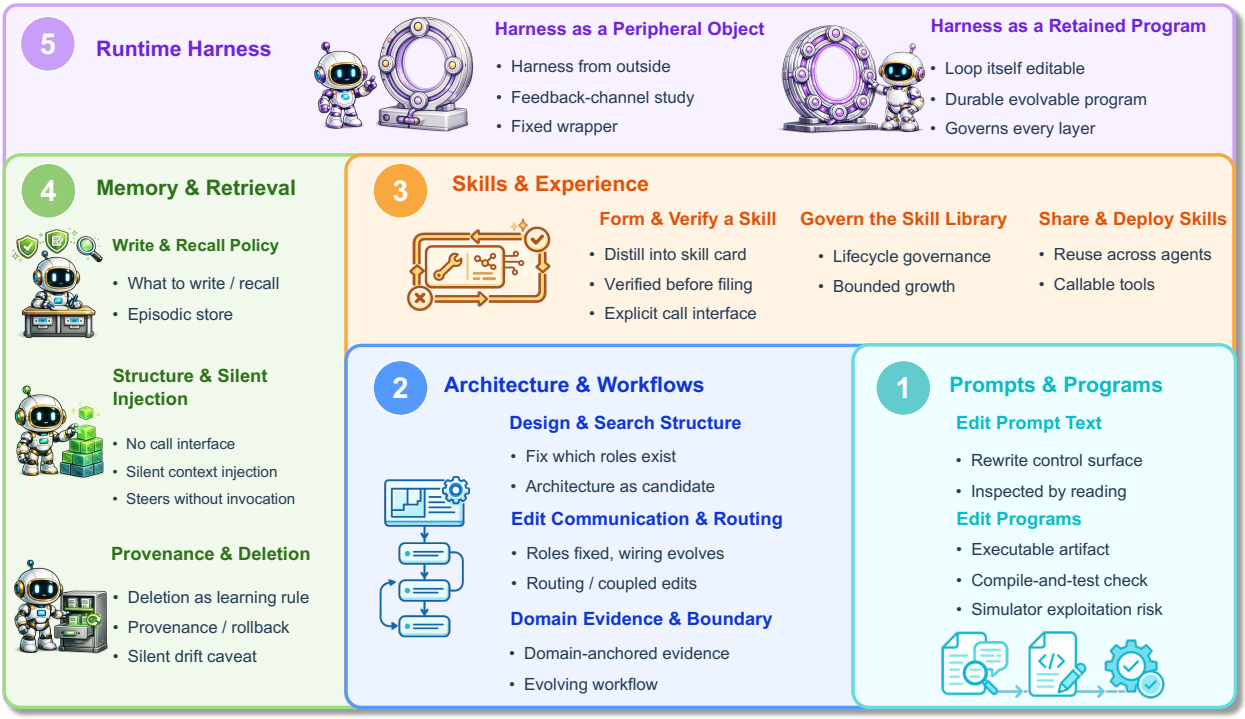


Figure 6 | **The widening scaffold scope of L2.** The five layers are drawn as progressively larger nested regions, from a single prompt or code artifact, through the agent architecture, the skill library, and the memory store, out to the runtime harness that encloses them all. Each wider region contains the smaller ones, which shows that a broader scaffold presupposes the narrower objects it organizes while the improver and criterion stay fixed.

Rewriting the control surface. The base move keeps the model fixed and rewrites the natural-language control surface, meaning orchestration prompts, generation prompts, and test-time prompt configurations, in response to observed failures [82–84]. For instance, self-optimizing deep-research orchestration searches prompt combinations through self-play to match or surpass expert hand-written prompts [82]. AutoVQA-G applies the same idea to annotation prompts, refining them through consistency-checked criticism from a prompt-optimization agent [83]. EEVEE goes a step further and co-evolves a task router with per-cluster prompt configurations to limit cross-dataset interference [84]. This is the smallest and most inspectable edit available, but its attribution is weak, because a single reworded instruction can at once alter tool use, evidence selection, and stopping behavior, so gains measured on the tuning stream need held-out confirmation.

Typed, auditable artifacts. A more disciplined branch narrows the edit surface to typed or human-readable artifacts that support localized comparison [85–87]. SHARP replaces free-text mutation with atomic edits to a condition-action rubric, and it localizes rule failures with a cross-sample attribution agent under walk-forward validation [85]. Relatedly, Combee scales parallel prompt learning from aggregated trajectories without loss of quality [86]. Meta Context Engineering extends the pattern to reusable context-engineering skills, co-evolving them together with the context files and code they produce [87]. The reliability advantage is inspectability and rollback, because a constrained rubric or context operator makes each accepted change diffable and attributable, unlike opaque free-form rewriting.

Toward the improver boundary. Text edits reach the improver boundary when the retained prompt begins to govern how future prompts are optimized [291]. SePO has the prompt agent optimize its own system prompt

alongside the task-agent prompt within an open-ended evolutionary search over a candidate archive [291]. The retained task-agent prompt is a Scaffold-Level object, but the prompt agent's own retained system prompt governs how future task prompts are optimized and therefore reaches into the Improver-Level. This case marks where text edits stop being purely scaffold and begin to touch the mechanism that proposes edits. The distinction follows from which future process the retained prompt controls, not from self-referential terminology alone.

5.1.2 Editing Programs and Symbolic Artifacts

This second part moves from instructions to executable programs and symbolic rules, which enlarges the reach of an edit while strengthening the local checks available before promotion. It is placed after prompt editing because a program can be compiled, run, and regression-tested, so the reliability question shifts from readability to whether those executable checks predict transfer.

From instructions to programs. The most direct case has a frozen model rewrite an executable artifact under a loop that compiles and scores each candidate [88, 292, 293]. A frozen model iteratively rewrites the source of the ABC logic-synthesis system under a joint correctness-and-quality loop [88]. SelfEvolve applies the same approach at runtime, synthesizing and integrating new functions to extend a live software system without restart [292]. A game-playing agent likewise refines a Python control policy from execution traces and language feedback [293]. Because a candidate program can be compiled, executed, diffed, and regression-tested, the local evidence is stronger here than for text, but these checks establish syntax and behavior only on covered executions and not transfer.

Program discovery under a simulator. The same propose-compile-evaluate pattern recurs in scientific and algorithmic discovery, where the language model evolves an external program while a simulator or downstream utility decides retention. Instances include a fluid controller refined from simulation diagnostics [294], MLEvolve's tree-searched AutoML pipelines with retrospective memory [89], co-evolved coupled heuristic operators for combinatorial optimization [295], and archived causal-effect estimators [296]. These loops iterate across generations, which superficially resembles self-improvement, but the improver stays frozen and only the external program evolves, so the retained object is Scaffold-Level rather than Improver-Level, and a one-off controller not carried across tasks is merely an ephemeral output. The characteristic reliability caveat is simulator exploitation, because a program can win on the evaluation harness without transferring.

Screened symbolic rules. A final group generates symbolic rules and structured representations that a verification loop can screen before promotion [90, 297, 298]. GenTI turns analyst prompts into deployable Snort, Suricata, and YARA intrusion rules through a chain-of-verification pipeline [90]. HIER instead refines injected hierarchical semantic representations from multimodal-LLM feedback at inference time [297]. Dense-feedback policy synthesis produces code-based policies scored by social metrics rather than a single scalar reward [298]. Rejecting malformed artifacts before promotion is a reliability gain, but compilation and unit tests certify only syntax and behavior on covered cases. Held-out end-to-end evaluation is still required to establish broader benefit, which motivates the coarse behavioral-evidence standard that recurs throughout this level. Once individual prompts and programs are editable, the next layer widens the scope from single artifacts to their organization, meaning which roles exist and how control flows between them, and that organization is the agent architecture.

5.2 Architecture and Workflows

Where the previous layer rewrote what a single prompt says or what a program computes, this layer widens the object to how components are organized and how control flows among them. The mutable object is now which roles exist, how they communicate, which tools they may call, and how work is routed through the system. A structural edit changes the interaction pattern rather than one component's behavior, which places it above the text and code layer. It also fixes the organization within which reusable competence will later accumulate, which places it below the skills layer.

5.2.1 Designing and Searching Structure

This first part concerns the organization itself, ranging from hand-designed staged roles to architectures treated as candidate artifacts scored by an evaluator. We group these together because they all fix or search which roles exist, which is logically prior to how those roles then communicate or accumulate skills.

Staged roles with verifiers. The smallest structural move decomposes a monolithic agent into specialized roles bound by a staged workflow, closing each stage with a verifier so that failure can be localized [91, 299, 300]. SEMA-RAG illustrates the pattern by separating interpretation, exploration, and adjudication into distinct roles [91]. The same division appears in clinical and planning systems, which instantiate doctor, patient, and measurement roles or policy, world-model, and critic roles [299, 301, 302]. A stricter variant closes every stage with a check, and these verification-closed pipelines route biological-protocol generation, C-to-synthesizable-C conversion, image restoration, and articulated-CAD assembly through staged verifiers under a central coordinator [300, 303–306]. AgentOrchestra takes the coordination further by placing specialized agents beneath a central conductor through a Tool-Environment-Agent protocol, and it retains the conductor's memory, tools, and prompts across independent tasks while the base model stays frozen [92]. The reliability contribution is attribution: role separation and per-stage verifiers make it easier to say which component failed. But the decomposition here is designed rather than searched, so it provides evidence about a good hand-built structure rather than about a structural edit that transfers. What persists across tasks is the experience accumulated inside the fixed roles, and not yet a transferable structural edit.

Architecture as candidate artifact. A second group treats the architecture itself as the candidate artifact, searching, recombining, and selecting over a role pool, module graph, or agent configuration [94, 307–317]. AgentSquare searches a modular design space of planning, reasoning, tool-use, and memory blocks [94]. GPTSwarm instead casts language agents as computational graphs and runs automatic graph optimizers that both refine node-level prompts and rewire edge connectivity, and it retains an optimized topology that transfers across MMLU, HumanEval, and GAIA [95]. SERO evolves a typed role-card pool and admits a proposal only when it preserves the structural contracts and improves task score [307]. MetaGen rewrites query-conditioned roles and topologies [308]. Further systems evolve reasoning architectures, share exploration across a population, accumulate executable sub-agents, evolve the entire configuration without gradients, or expand a factorized action space [309, 318–321]. These systems are Scaffold-Level only when a selected structure is retained and reused on later independent tasks: MAS-Zero designs a fresh configuration per instance and does not qualify unless that configuration is carried forward, and MetaGen's per-query generation counts only when a role cache is kept [96, 308]. Reliability then turns on transfer beyond the selecting tasks and on matched model calls, tools, and latency, so that additional coordination is not mistaken for a better structure.

Reward-scored workflow synthesis. Workflow-synthesis systems close the search loop with an evaluator or reward signal that scores generated topologies [93, 322–324]. Mimosa's meta-orchestrator generates workflow topologies and refines them from an LLM judge [93]. HERA jointly evolves orchestration and role prompts with reward-guided sampling and credit assignment [322]. GraphMind mines operational traces into an executable workflow graph and reinforces the paths that succeed [323]. The Puppeteer paradigm schedules agents through a central orchestrator [324]. Retained workflows are L2 when no deeper active target changes. If persistent optimization experience changes how later workflows are proposed, the realized transition reaches L3. The Puppeteer orchestrator also contains a simultaneous L1 scheduling-policy update, but we cite it here for its retained orchestration structure.

5.2.2 Communication, Routing, and Coupled Edits

This second part holds the set of roles fixed and instead evolves how they communicate and how instructions co-evolve with the tools they call. It is separated from structure search because the edit now targets the interaction pattern rather than the inventory of roles, which raises attribution as the central difficulty.

Communication graph and routing. The first object of this kind is the communication graph together with the routing policy that moves work through it [97–99, 325]. QueenBee treats the inter-agent DAG as a retrievable design skill and guards each edit with held-out acceptance and motif-level attribution [97]. Along the same lines, TopoEvo, TacoMAS, EVOCHAMBER, and SkillGraph co-adapt edges, expertise, or team composition at test time [98, 325–327]. Routing-centric designs narrow the object further, and they co-evolve routing with prompts, send problems to disjoint reasoning frameworks as an epistemic-control layer, or route prompts across generative back-ends [99, 328, 329]. The characteristic risk is confounding: an apparently superior graph may exploit a benchmark-specific ordering, a permissive judge, or extra compute rather than a transferable coordination principle, so a claim should report what persists and control for communication volume and total inference budget. A parametric caveat also applies, because a differentiable or fine-tuned router is a Model-Level update when its parameters are retained, as in the differentiable mixture-of-agents router [330]. By contrast, a symbolic router that changes no weights stays at the Scaffold-Level, and a test-time topology is Scaffold-Level only when the resulting graph crosses the task boundary.

Coupled instruction-tool edits. Structure rarely evolves in isolation: instructions co-evolve with tools, and layered agent architectures are revised from execution traces, so the added reliability requirement is attribution across a widened edit surface [331, 332]. EGL-SCA couples an instruction (prompt) space with a tool (program) space under a verifier and uses structural credit assignment to route each failure either to prompt optimization or to tool synthesis and repair [331]. CyberEvolver instead turns noisy execution logs into actionable revision signals through a trace-to-diagnosis mechanism and preserves diverse variants with population-based beam search [332]. Once several structural components change together, a single score cannot reveal which edit helped, and explicit credit assignment is what keeps a coupled-space edit auditable.

5.2.3 Domain Evidence and the Improver Boundary

This third part collects the domain systems that supply most of the layer's evidence and marks where architecture search begins to press against the improver boundary. It closes the section because these cases test the boundary rather than extend the mechanism, showing when an evolving structure stays Scaffold-Level and when it starts to change how future edits are proposed.

Domain systems as evidence. In these systems a frozen model gains capability because the workflow around it evolves, and the domain supplies the executable or simulator-anchored evidence that certifies each edit [333–348]. In engineering design, repository-level and multi-agent loops evolve hardware and HLS code, RF amplifiers, power-flow analyses, and catalyst digital twins on executable or simulator-anchored evidence [333, 349–353]. In production and operations, comparable systems iterate industrial recommenders, mobility heuristics, safety-critical driving scenarios, cloud defense, literature retrieval, feature-engineering trees, and NOTAM interpretation [334, 354–359]. In embodied control, further systems evolve modular on-orbit and robotic control stacks [335, 360, 361]. These systems are Scaffold-Level only when the evolved structure persists and is reused: a configuration produced within a single task and then discarded remains transient, and mixed cases must be split. SpaceMind and the metasurface framework state explicitly that no weights are updated [335, 351]. C-NAV instead includes a simultaneous L1 update through feature distillation and replay [362]. Across this family, validation must match models, tools, and latency so that extra coordination or a stronger sub-agent is not mistaken for a better architecture.

Approaching the improver boundary. Toward the outer edge of this layer, architecture search begins to press against the Improver-Level [138, 140, 363]. AIRA and TacEvo let an LLM search harness autonomously design external neural architectures and training scripts, and BaSE studies how to allocate compute across an evolutionary search [363–365]. In each of these the search rule and evaluator stay fixed, so the evolving object is an external artifact rather than the improver itself, and the systems remain Scaffold-Level in substance while only motivating recursive self-improvement. The genuine crossings are EvoTrainer, which co-evolves an LLM policy (a Model-Level update) with a training harness and retained diagnostics that shape subsequent search [140], and TPGO's meta-learner, which learns from past optimization experience how to propose better edits [138]. Because these retained changes alter the procedure that generates future updates, the complete realized transitions are L3 and are discussed in §6. Once these structures begin to accumulate experience across episodes, the question shifts from how components are arranged to which competences should persist and be reinvoked, which is the subject of the next layer on skills.

5.3 Skills and Experience

Once structural search has fixed how roles, tools, and workflows are arranged, the next question is which of the competences those arrangements produce should be retained and reinvoked on later independent tasks. This layer therefore advances from structural configuration to the persistent, invocable competence that configuration yields, widening the mutable object from a system's organization to its retained experiential state. It sits after architecture because skills accumulate only once components are stably arranged, and before memory because a skill is disciplined experience equipped with an explicit invocation interface and a declared validity domain, whereas general memory can shape behavior with neither.

5.3.1 Forming and Verifying Skills

This first part concerns how a skill is created and checked, from a distilled trajectory to a form a verifier can screen before promotion. We treat formation before governance and sharing because a bank can only be curated or distributed once its entries exist and can be validated.

Distilling trajectories into skills. The base move of this layer is to distill successful and failed trajectories into named, retrievable artifacts, such as natural-language strategies, self-evolving rewrite rules, reflective heuristics, metacognitive knowledge, or visual concepts, that a frozen model consults through an explicit retrieval-and-injection interface [100, 366–381]. What distinguishes such an entry from a raw episode is

that it is meant to be invoked on independent tasks and carries an implicit validity domain, whether an input type or a task class. This is why early lifelong learners already pair accumulation with input-type routing and with explicit create, validate, and retire operations over rules [29, 382, 383]. For reliability, the operative question thus shifts from whether the library grows to whether an invoked entry transfers beyond the trajectory that produced it, and playbook-style artifacts push this further by claiming transfer even to weaker backbones [384]. As a strength-of-evidence caveat, EvolveR is a mixed system whose policy-reinforcement component is a Model-Level (weight) update [385]. It is cited here only for its persistent external rule store, not as a scaffold-only method.

Skills as verifiable programs. Representing a skill as a program, a logic-grounded procedure, a callable tool, or a formally specified contract narrows its invocation interface and strengthens the local evidence available before promotion, because a candidate can then be compiled, model-checked, or rejected by a verifier. Some hybrids go further and crystallize recurring plans into deterministic code that needs no inference-time model call [386–389]. Verification-guided systems turn counterexample traces into text-gradient revisions of the skill contract, and co-evolving surrogate verifiers can vet multi-file skill packages when ground-truth tests are absent [101, 102]. For reliability, however, formal or execution checks establish syntax and behavior only on the executions they cover, so held-out, end-to-end evidence remains necessary to establish transferable benefit. On classification, VASO (skill contracts) and KBSpec (an evolving formal knowledge base) were originally flagged as Improver-Level boundary cases. Yet their base weights stay frozen and they include no self-modifying improver, so they evolve a skill or specification artifact rather than the improver itself, and they appear here as Scaffold-Level cases [102, 390].

5.3.2 Governing the Library

This second part treats the skill bank as an object with a lifecycle, covering admission, revision, typed composition, and retirement. It is separated from formation because the reliability concern is no longer whether one skill is valid but whether the bank as a whole stays useful as it grows.

Lifecycle governance. The first requirement of such governance is an admission rule that keeps the bank from growing without control [103, 104, 391–398]. Such stores admit a candidate only when it yields a net improvement on balanced held-out probes under a hard regression budget, revise skills from execution evidence, and expose explicit create, improve, merge, and retire operators with provenance and rollback [103, 104, 391, 399, 400]. Selection is increasingly multi-objective over utility, cost, and regression risk, whether in Pareto or training-free-GRPO style [105, 401, 402]. Error-driven revision clusters failures, for example into missing, wrong, or conflicting cases, and turns them into targeted patches that are committed only when a regression gate passes and can compile to auditable deterministic code [403, 404]. What separates cumulative capability from library bloat is a sequestered acceptance set together with regression accounting, and a useful counterfactual asks whether removing an entry degrades fresh-task performance, while still allowing for redundant or substitutable skills. A recurring hazard, framed as an empirical finding rather than a definition, is silent regression: an added skill can improve its targeted cases yet quietly break previously correct behavior, which is precisely what admission gates and rollback are designed to catch.

Typed composition and tool co-evolution. As libraries scale, skills must be selected and combined without silently changing meaning, which calls for typed interfaces, explicit dependency and conflict structure, and skills that co-evolve with the tools they call [106, 405, 406]. Typed skill graphs meet part of this need by exposing a structural retrieval interface and registering edges only with execution evidence under a propose-

then-commit protocol [106]. In the same spirit, ecological utility models bind competence to executable tool interfaces and retire recorded anti-patterns [405, 407], while on-demand tool forging recycles verified procedures back into the callable set [406, 408]. On-demand tool generation extends this to whole toolboxes. A generalist agent with minimal predefinition dynamically forges and persists callable MCP tools in a reusable toolbox across independent tasks [107], and a closed-loop orchestration pipeline maintains a persistent tool dataset of generated tools that later tasks reuse [409]. In both cases callable capability accumulates while the base model stays frozen. Explicit contracts and conflict signals make composition auditable and let a later regression be localized to a specific edge, skill, or tool, which flat skill lists cannot support. Several of these systems also contain trained sub-components, such as a jointly optimized planner, that are separate Model-Level updates and are cited here only for the scaffold-level skill and tool structure they retain.

5.3.3 Sharing and Deploying Skills

This third part concerns the largest scope, where skills become shared assets that move across agents, models, or users rather than staying private to one agent. It follows governance because sharing adds a compatibility requirement on top of curation, so a bad entry can now propagate beyond the agent that created it.

Shared skill assets. As shared assets rather than private notes, these skills take several forms: living knowledge bases link heterogeneous resources to provenance-bearing reusable skills, validated skill packages are assembled from scientific corpora, and portable specifications or federated skill diffs move competence across agents, models, and clients without a common weight update [108, 410–413]. Open-world and collective variants bootstrap or synchronize libraries that transfer across models through self-built verification anchors and cross-user aggregation, and they add institutional safeguards such as audit separation, contract protocols, and organizational memory to guard against knowledge degradation [414–416]. Sharing introduces a new compatibility requirement: interfaces and semantic versions must remain valid across models, tools, and users, so transfer success must be measured separately from within-agent reuse. Shared stores also raise the stakes of admission, because a bad entry now propagates.

Domain deployments. Most of the empirical support for this layer comes from domain deployments that wrap a frozen model in an evolving skill or heuristic library, and these deployments share the coarse, end-to-end evidentiary burden of the level [417–429]. In embodied navigation and manipulation, agents distill trajectories into scene-aware or heuristic skill libraries with confidence-gated retrieval and pre-action foresight, and robot-design trials are turned into transferable skills [417, 430–436]. Web agents built on a frozen model autonomously discover and hone a library of lightweight, plug-and-play APIs that persist and are reused across tasks [13]. Medical, emotional-support, and recommendation agents accumulate value-aware, procedural, or per-user policy skills [418, 437–443], and a biomedical research agent distills successful reasoning strategies into reusable templates archived in a growing library and tool collection that a frozen base consults across tasks [444]. Software, data-science, and finance agents record which repair procedures or edit motifs succeed and which fail [419, 445–452], and multimodal agents evolve skill harnesses for image workflows, retrieval, few-shot reasoning, and guide-to-skill distillation [453–460]. These settings expose the same failure risk as the layer as a whole, namely fitting benchmark, judge, or router quirks rather than a transferable procedure, and the strongest instances pair accumulation with execution-anchored validation and with explicit cautionary failure entries rather than storing only successes. Skills expose competence through an explicit invocation interface and a declared validity domain. The next layer widens the scope to governable memory, where retained experience can influence later behavior without any such interface or local validity check.

5.4 Memory and Retrieval

Skills expose reusable competence through an explicit invocation interface, whereas persistent memory removes that boundary, so a single stored entry can steer a later decision without ever being deliberately called. This layer therefore extends the mutable scaffold to its widest and least-visible causal footprint within the Scaffold-Level. It sits after skills because dropping the invocation boundary also drops the local validity check that made a skill auditable, and before the runtime harness because governing these silent writes is exactly what motivates a versioned, replayable execution substrate.

5.4.1 Write and Recall Policy

This first part concerns the operations over the store and the policy that governs what to write and recall, made auditable and in some systems adaptive. We treat the policy before the store's structure because what enters and leaves memory is prior to how the retained entries are then organized.

Typed, auditable writes. Because a memory entry can shape a later decision without being explicitly invoked, the reliability question shifts from whether an artifact was called correctly to what silently entered context and whether that write can be inspected [109, 461, 462]. A first group answers this by replacing free-form summarization with typed, auditable operations over the store. ACE treats the context as an evolving playbook and updates it through structured incremental edits that avoid the context collapse produced by iterative rewriting [109]. In the same spirit, Evo-RAD casts evidence acquisition as an MDP with explicit DELETE, INSERT, and TERMINATE moves [461], while Thought-Retriever narrows what is stored to reusable intermediate reasoning rather than raw chunks [462]. MemRes follows the same typed approach and accumulates structured, reusable resolution patterns in a domain error-pattern base [463]. Evo-RAD's GRPO-trained graph agent is a simultaneous L1 update. We cite the system here only for its structured evidence operations over the retained store. A typed write surface is diffable and attributable, whereas unconstrained rewriting leaves no record of what changed or why.

Adaptive write and recall policy. Beyond fixing the operation set, several systems make the policy governing what to write and recall itself adaptive while leaving base weights fixed [110, 111, 464–466]. MemRL applies reinforcement learning over episodic memory but is explicitly nonparametric, so the update lands on the retained store rather than the model, and it remains Scaffold-Level despite the RL label [111]. ReasoningBank distills generalizable strategies from self-judged successes and failures, then retrieves and rewrites them at test time and scales the process with memory-aware test-time search [110]. MemSkill recasts memory operations as learnable memory skills and pairs a controller that selects them with a designer that reviews hard cases to propose new ones [464]. MetaMem evolves a meta-memory that governs how stored knowledge is used [465]. SEDM goes further by adding a distributed controller that admits writes through reproducible replay, then ranks and consolidates entries by empirical utility. Its cross-domain knowledge diffusion abstracts reusable insights so that entries transfer and evolve across heterogeneous tasks while the base stays frozen [466]. The taxonomy boundary is the operative caveat: a curator whose parameters are trained and retained is a Model-Level update, even when the store it governs remains a Scaffold-Level object.

5.4.2 Structure and Silent Injection

This second part concerns how the store is organized and how its entries reach later behavior when they are injected silently rather than by a deliberate call. It follows the write policy because organization determines what remains retrievable, and silent injection is the risk that makes this layer harder to audit than skills.

Structuring the store. How the store is organized decides what stays retrievable and whether a later regression can be localized, so a second line of work structures the store itself rather than the write policy [112, 114, 467–470]. xMemory decouples interaction history into a revisable message-to-fragment-to-component-to-group hierarchy that is maintained incrementally [112]. H-Mem combines temporal and semantic trees with a knowledge graph so that short-term entries consolidate into long-term ones [467]. SEGA layers episodic, semantic, and experiential memory for test-time extension, while its MASE stage adds a simultaneous L1 update by training the base policy [468]. AI-Agent School maintains parallel experience and knowledge stores [471]. SHIMI organizes memory into layered semantic nodes and supports top-down traversal from abstract intent to specific entities, with agents maintaining local hierarchical memory trees that are synchronized asynchronously, so the layered structure persists and shapes retrieval on later tasks without base-model training [469]. SGMem represents dialogue as sentence-level graphs and combines retrieved raw turns with generated summaries, facts, and insights that persist across conversations to supply the frozen LLM with accumulated long-term-dialogue context [470]. RGMem borrows renormalization-group coarse-graining to consolidate episodic dialogue into stable user profiles [114], and AutoAgent orchestrates elastic prompt-level cognition over the store [472]. Consolidation controls which state stays active, but every compression step is also a chance to discard the provenance needed to audit a later failure.

Silent injection over long horizons. The defining risk of the layer becomes concrete when experience is injected by retrieval or runtime hooks rather than by a deliberate call, since it then shapes behavior with no local validity check [113, 473–475]. WebCoach condenses cross-session navigation logs and injects advice through runtime hooks, remaining model-agnostic and requiring no retraining [113]. PRINCIPLES derives a reusable synthetic strategy memory from offline self-play to guide planning at inference [473]. PRIME accumulates user preferences together with success and failure modes without gradients, then evolves them with meta-operators [474]. The FRIDAY agent of OS-Copilot, built on a frozen LLM, accumulates tools and skills into a self-improving procedural memory after successful completions and generalizes to unseen applications in a generalist-computer-agent setting [475]. Mem^{2}Evolve co-evolves an experience memory with an asset memory that spawns tools and sub-agents [476], and NavMorph and Richelieu maintain contextual or self-play memory that adapts online in navigation and diplomacy [477, 478]. Because these stores condition tool use and planning silently and persistently, their effect must be measured over long horizons rather than within the session that wrote them.

5.4.3 Provenance and Deletion

This third part treats temporal provenance and validated deletion as part of the learning rule rather than as mere capacity management. It closes the layer because a store that only grows eventually accumulates stale or conflicting entries, so what is removed matters as much as what is written.

Temporal provenance and reconciliation. Persistent state accumulates stale and contradictory entries, so these systems add temporal provenance and query-time reconciliation [115, 479, 480]. APEX-MEM keeps an append-only property graph that preserves the full temporal evolution of the store and uses a multi-tool retrieval agent to resolve conflicting or evolving facts at read time [115]. MemoTime attaches a temporal knowledge graph and stores verified reasoning traces and tool decisions for reuse [479]. ARIA maintains a timestamped knowledge base, evaluates its own uncertainty, and requests human guidance to detect and resolve conflicts or outdated knowledge [480]. Temporal provenance is what lets the system tell a stale fact from a current one, and conflict resolution is needed precisely when user preferences, world state, or learned procedures disagree, and neither capability is achievable over an undated flat store.

Deletion as a learning rule. The layer culminates in treating deletion and demotion as part of the learning rule rather than as capacity management [116, 481, 482]. ReMe manages a procedural-memory lifecycle with utility-based refinement that actively removes stale entries to keep the pool compact [116], and Live-Evo learns online from continuous feedback, reweighting useful experience while forgetting misleading or stale entries under distribution drift [481]. A cautionary case sharpens why such filtering is not optional. CTIM-Rover augments a software-engineering agent with a cross-task-instance episodic memory retained across tasks on a frozen base, yet it fails to outperform its memoryless baseline in any configuration. The reported analysis attributes the degradation to noise from distracting retained items rather than to accumulated useful knowledge [117]. This negative finding is direct evidence that an unfiltered store persisting across tasks can accumulate noise instead of competence, reinforcing that continuous writing without validated deletion is itself a reliability hazard. The interpretive claim that closes the layer is that continuously rewritten memory can degrade even when it was initially useful, so a deletion must be validated by replay against retained raw episodes to confirm that it repairs one failure without silently reintroducing another. This is the Scaffold-Level analogue of Model-Level self-consumption, because the weights are unchanged yet future behavior is conditioned on state produced by earlier, potentially incorrect system behavior. Because memory has a wider and less-visible causal footprint than an explicit skill, promotion and retirement here should be judged over longer horizons and with probes that exercise retrieval, tool use, and downstream decisions together. Because memory writes act silently and over long horizons, they demand an execution substrate that can version, replay, and roll them back, which is the runtime harness that governs every scaffold edit and forms the final layer.

5.5 Runtime Harness

Skills and memory settle which experience persists, whereas the harness settles how every persistent component is invoked, observed, and committed. This closing layer therefore widens the mutable scaffold from the objects being run to the loop that runs them: the harness builds context, calls tools, observes traces, attributes failures, and executes internal checks, while the fixed improver controls proposal, internal selection, commit, and rollback. It sits last because the harness is the substrate that executes and observes every earlier layer, so making it writable subsumes prompts, architecture, skills, and memory under one editable loop. It is also the thinnest layer, since much apparent loop engineering retains trained weights and therefore belongs to the Model-Level rather than the scaffold.

5.5.1 The Harness as Peripheral Object

This first part treats the harness from the outside, either analyzing its feedback channel or building it into a verification apparatus. We separate these cases because neither retains the harness as an evolved object, so they set up the contrast with the durable-harness systems that follow.

Feedback channel as object. The first such move studies the harness's feedback channel rather than the harness as a whole. CUDAnalyst freezes trajectories and selectively injects individual feedback components, using coalition-style attribution to identify which signals actually drive planning decisions in CUDA-kernel generation [118]. Its reliability contribution is observability and causal attribution inside the loop, because it renders the feedback-to-plan pathway diagnosable rather than opaque, so that a later gain can be traced to a specific reasoning-feedback interaction. This remains a diagnostic analysis of the loop's peripheral feedback mechanism with no weight change, and such attribution supports, but does not by itself constitute, a persistent scaffold update.

Harness as verification apparatus. A related move makes the harness itself the verification apparatus. AutonomyLens unifies scenario specification, simulation execution, telemetry analysis, and counterfactual test generation into a single self-evolving testing loop for autonomous systems, synthesizing new test cases from observed failures to close the loop [119]. Here the harness is the machinery that produces the end-to-end behavioral evidence on which the Scaffold-Level regime depends. Yet building a testing loop is distinct from certifying a persistent scaffold gain, because this is engineering of the verification harness, a peripheral program with no weight update, so the loop generates evidence rather than constituting the retained capability itself. A writable testing harness can therefore supply development evidence, but it cannot serve as the external audit of its own update.

5.5.2 The Harness as Retained Program

This second part covers the systems that retain the harness itself as a durable program which governs every lower layer within one loop. It follows the peripheral cases because here the loop is the evolved object rather than a fixed wrapper, which is the deepest form a scaffold edit can take.

Harness as durable program. A cluster of systems makes the harness the durable, evolvable program rather than a fixed wrapper. M* evolves task-specific memory harnesses by iteratively adding and removing retrieval, indexing, and scoring mechanisms from a RAG baseline, so that each task receives a customized executable memory loop [121]. In the same spirit, Milkyway maintains a persistent, editable prediction harness of reusable procedural guidance, updating it from pre-resolution signals gathered by repeatedly revisiting unresolved questions [483]. Meta-Harness treats the whole harness as the optimization target and has a coding agent search the full execution history, logs, and source of prior runs rather than a compressed summary. Each retained edit to context construction, tool use, error recovery, and state management is therefore grounded in complete diagnostic evidence [122]. Self-Harness structures the same idea as a repeating loop that mines weakness patterns, proposes a harness improvement, and admits it only after regression verification, which makes internal admission explicit rather than implicit [123]. SemaClaw carries the same durable-harness view into production personal agents, organizing execution through a two-phase orchestration graph with tiered context management and an explicit permission layer. The harness rather than the frozen model thereby becomes the auditable and controllable object [124].

Specializing the harness object. Other systems keep the durable-harness view but vary how much of it is placed under evolution. HarnessForge is a mixed case that co-adapts the retained harness together with the internal reasoning policy. The realized transition is L2 with a simultaneous L1 policy update, and we cite it here for the harness change [125]. AutoHarness narrows the object to the guard code itself, automatically synthesizing the harness that wraps a frozen agent so that prohibited or invalid actions are caught before execution rather than corrected afterward. This replaces the hand-written harnesses that such systems usually require [120]. LedgerAgent instead makes the harness state explicit, maintaining a persistent ledger of the facts, identifiers, and constraints observed across turns. Policy-adherent tool calls then read from a retained structured state rather than reconstructing it from the prompt on every step [484].

Scaling and navigating the harness. A further group scales the retained harness across cases and makes its codebase navigable for the next edit. MemoHarness treats the whole harness as the adaptive object, learning from its own executions to adjust six editable control dimensions, namely context, tools, orchestration, memory, decoding, and output handling, per case rather than reusing one global harness for every task [126]. The Last Harness pushes this to two levels, pairing a per-task harness-evolution loop with a cross-task

meta-evolution loop that generalizes harness blueprints. We read the retained object here as the harness itself, while noting that the meta-loop begins to touch the improver machinery formalized in §6 [127]. A complementary line makes the harness itself navigable, since before any edit can be applied a developer or coding agent must first find every code location that implements the target behavior. The Harness Handbook synthesizes a behavior-centric representation of a harness codebase through static analysis and LLM-assisted structuring, linking each behavior to its source and guiding edits from high-level behavior to implementation. It thereby treats behavior localization rather than edit generation as the central bottleneck in harness evolution [128]. Across these harness-as-program systems the retained Scaffold-Level object is an executable harness, checkable by execution rather than only by prose.

Capstone: governing every layer. Continual Harness is the capstone case, a reset-free, online self-improving harness for embodied agents that alternately executes and refines its own prompts, subagents, skills, and memory from a minimal environment interface [129]. It shows the runtime substrate governing every earlier layer within a single loop, which is why it sits last in the progression. As a mixed system, it also contains a teacher-relabeling co-learning step that updates the model at L1. We cite it here for the retained harness change, which is the deeper L2 target. The harness completes the scaffold-scope layering, yet every harness edit is still proposed, internally selected, and committed by a fixed improver. External acceptance remains a separate decision made by the acceptance policy, which motivates the reliability accounting for scaffold overfitting and the subsequent move to Improver-Level Self-Evolution, where the internal update procedure itself becomes writable.

5.6 Reliability and the Fixed-Improver Limit

The closing observation of §5.4, that continuously rewritten memory can silently corrupt future behavior as the Scaffold-Level analogue of Model-Level self-consumption, shows that local component-level checks are insufficient to certify a scaffold edit. This section steps back from individual mutable objects to ask what single standard of evidence certifies any scaffold edit once weights are held fixed and the signal has coarsened to end-to-end behavior. It supplies the Scaffold-Level analysis summarized later in the cross-level reliability ledger of Table 9: the external audit, reliability condition, characteristic failure mode, and fixed-improver ceiling. The reliability column of Table 5 already set the five method categories side by side on the axis that governs how hard each is to certify, pairing the strongest locally available check with the failure that check cannot catch. The following analysis identifies the external audit that can certify a scaffold edit and the condition a revised scaffold must meet before promotion.

External audit. The certifying evidence for a scaffold edit is a controlled end-to-end comparison of the revised agent against its incumbent scaffold on tasks the edit did not select. It is not an aggregate benchmark number, which conflates the edit's contribution with sample luck, extra tool calls, more retrieved context, or a more permissive judge. The load-bearing support is the decoupling of two abilities that are usually collapsed, namely producing a persistent scaffold update from execution evidence (harness-updating) as distinct from actually benefiting from that update at solve time (harness-benefit) [286]. At its stated strength the finding is that updating is roughly flat in base-model capability while benefit is non-monotonic, and this holds as an empirical result on specific models rather than a law. The decoupling therefore explains why a successful update and a higher score do not by themselves certify improvement under the declared external target. It anchors the reliability analysis that follows.

Reliability condition. The reliability condition requires representative fresh tasks evaluated under matched models, tools, token budgets, and stopping rules. It also requires acceptance gates whose evidence the proposer cannot query indefinitely, together with a tested rollback path. For accumulating stores the condition sharpens into bounded lifecycle management rather than append-only growth. SkillBrew formalizes skill-bank curation as a constrained multi-objective problem over usefulness, diversity, and coverage, solved by a bi-level propose-then-verify loop, so that redundant, outdated, or harmful skills are removed rather than merely accumulated [485]. A more minimal variant, Ratchet, works around a frozen model and combines outcome-driven retirement, a bounded active cap, meta-skill authoring guidance, and schema normalization [486]. Its non-divergence result should be read as a stated proposition under its assumptions, that performance should not fall below the no-skill baseline, which is the external anchor that makes rollback meaningful. Matched budgets matter throughout, so that additional coordination or compute is not mistaken for a better scaffold. Two consequences follow from this evidence standard. The first is a characteristic failure mode, scaffold overfitting, and the second is that even a well-audited scaffold leaves the fixed improver as the ceiling that the next level takes as its object.

Failure mode: scaffold overfitting. The characteristic failure is scaffold overfitting, which fits benchmark, judge, memory, or router quirks rather than a transferable procedure, and it appears in three diagnosable variants. First, holdout erosion and long-horizon drift turn a repeatedly queried nominal holdout into adaptive development evidence. A 391-consecutive-session action study reports this pattern: trading more formal and symbolic constraints for reliability backfired, and the system degraded into symbol-layer self-reference at the expense of task semantics [487]. This evidence remains a single-project case study rather than a general law. Second, component-interaction confounds arise because errors surface late in long multi-agent trajectories and dependencies intertwine, so a candidate can look better only because work shifted or context grew, which means failure must be localized to responsible agents, steps, and harness artifacts. ErrorProbe performs symptom-driven back-tracing plus verified episodic-memory updates committed only on confirmed evidence, and HARNESSFIX maps trace-grounded attribution to scoped repair operators accepted under regression-aware validation [488, 489]. Third, silent store drift appears when a locally beneficial edit degrades behavior after the retrieval policy or neighboring skills change, which Library Drift documents with a reproducible drift-trigger ablation together with per-skill contribution scores and routing-participation logs [490]. Across these variants, a local check can miss failures that emerge only through component interactions, repeated use, or later retrieval. These failures are precisely why the external audit must be a controlled procedure rather than a score.

The next ceiling. No scaffold edit removes the final ceiling, because even a fully editable scaffold is optimized by a fixed, hand-built improver that determines which edits can be proposed, how candidates are internally selected, and how exploration is budgeted, while the fixed criterion supplies the internal judgment rules. Under tight evaluation budgets this improver choice is consequential in its own right. A systematic comparison covers Elo-tournament selection in RoboPhD, Pareto selection, and greedy hill-climbing under a fixed evaluation budget [491]. RoboPhD also uses validation-free evolution with Elo scores computed on training data. Together, these settings show that the improver, not the scaffold alone, governs which candidates are generated and selected internally. Whether those candidates should be accepted and promoted remains a question for an external acceptance policy using target-matched evidence. RoboPhD is strictly an Improver-Level boundary case, because its object is the optimization paradigm itself, so under the survey's numbering it belongs to the Improver-Level section (§6) and must not be read as a Scaffold-Level capability result. Making the improver writable is what §6 examines: the procedure proposing, selecting, committing, and rolling back scaffold edits becomes the evolution target rather than a fixed update process.

6 L3: Improver-Level Self-Evolution

L3 at a Glance

Boundary. L3 persistently changes the improver $\mathcal{U}_k$, the procedure that proposes, selects, commits, or rolls back later updates. Here, *self-reference* means that the current improver helps produce or internally select a candidate successor that can become the retained $\mathcal{U}_{k+1}$ after external audit and promotion. The criterion C_k remains fixed, so any audit-supported improvement claim concerns future update production rather than criterion validity.

Roadmap. We examine self-referential agents (§6.1), learned improvement strategies (§6.2), and reliability under a fixed criterion (§6.3).

Reliability focus. Internal traces, archive results, and descendant scores can identify a promising improver, but they do not establish that it produces better future updates. The declared external target and matched resource conditions define the comparison, a protected evidence source supplies fresh-task results, and an acceptance policy outside the update boundary maps that evidence to accept, reject, or escalate. Only a candidate accepted under that policy should be promoted, with versioned lineage and a tested rollback path. Otherwise, metric capture or correlated self-evaluation can favor an improver without corresponding improvement relative to the declared external target.

Section 5 allows persistent changes to prompts, memory, workflows, runtime harness, and other scaffold components while the improver stays fixed. The improver $\mathcal{U}_k$ is the mechanism that proposes, selects, commits, or rolls back candidate changes. It may interpret experience, allocate search resources, and make selection decisions. It does not define the criterion used to judge candidates. The criterion C_k stays fixed in L3, so the claim concerns the quality of later update production rather than the validity of the criterion.

Boundary and persistence. Writing the retained state as $X_k = (\theta_k, \sigma_k, \mathcal{U}_k, C_k)$, as in Equation (1), the L3 boundary is

$$X_k \longmapsto X_{k+1}, \qquad \mathcal{U}_{k+1} \not\equiv_{\text{act}} \mathcal{U}_k, \qquad C_{k+1} \equiv_{\text{act}} C_k, \tag{9}$$

A transition is L3 when the improver is the deepest component whose active semantics change and causally affect later updates. The model parameters θ_k or scaffold σ_k may also change in the same transition, but the criterion must remain fixed. A practical test first ignores the answer, algorithm, or task solver produced in the current round. It then asks whether the next round would propose, select, commit, or roll back changes differently because of what was retained. If the same update procedure still controls those actions, classify the transition by the retained target that actually changed. A model-state change is L1, an installed runtime-structure change is L2, and a transition with no retained agent-state change is L0. A standalone artifact outside the retained agent state does not by itself constitute an L3 transition.

Self-reference. Self-reference is a functional relation rather than a system name. Recursive calls or source-code editing alone do not establish it. It is present only when the current improver helps modify its own successor and the retained modification affects later update rounds. The required evidence is a retained causal change in how future updates are made.

Table 7 groups representative systems under the section's two main mechanism families. It compares the L3-relevant retained change, the internal signal or screening rule, and the main limitation. Figure 7 shows how both mechanism families produce a candidate improver under a fixed criterion.

Table 7 | Representative systems reviewed in Improver-Level Self-Evolution. The two groups correspond to self-referential agents and systems that learn better improvement strategies. A listed system realizes an L3 transition only when its retained change causally alters later update procedures. The third column reports development or internal-selection signals, not external audit evidence.

System	L3-relevant retained change	Internal signal or screening	Main limitation
Self-Referential Agents			
Gödel Machine [130]	Self-rewrite of its program and proof searcher	Proof under fixed axioms and utility	The guarantee depends on the formal model, and proof search is costly
Gödel Agent [131]	Task policy and self-referential learning algorithm	Goal-conditioned environment feedback and task-benchmark scores	Empirical scores give no formal guarantee, and the outer goal is fixed
Self-Developing, SICA, HyperAgents [132, 133, 492]	Improvement algorithm or combined task and meta-improvement logic	Outer preferences, utility, task scores, or costs	Task behavior and improvement logic may change together, making their effects hard to separate
Darwin Gödel Machine, Huxley–Gödel Machine [14, 134]	Retained agent code that changes how later descendants are produced	Coding scores, archive rules, and lineage-productivity estimates	Search and screening reuse benchmarks, and long-horizon productivity remains estimated
Learning Better Improvement Strategies			
STOP, Promptbreeder, SePO [135, 136, 291]	Self-applied improver or mutation prompts that control later prompt optimization	Fixed utility, training fitness, or task scores	The base model and the outer search loop stay fixed
Polaris, TPGO [137, 138]	Failure-derived repair policy or retained proposal experience	Failure traces, conservative validation, and rollback checks	Evidence must show that stored experience changes later update behavior
A-Evolve-Training, EvoTrainer, Meta Context Engineering [87, 139, 140]	Search direction, intervention policy, or reusable context-improvement skill	Development metrics, intervention backtests, and fixed task protocols	Simultaneous model, scaffold, and improver changes require causal separation

6.1 Self-Referential Agents

6.1.1 Proof-Based Self-Modification

Proof-gated self-modification. Proof-based methods turn the decision to install a self-modification into an explicit proof obligation. They therefore provide a clear formal starting point for self-referential updating. The Gödel Machine searches for rewrites under fixed axioms and a fixed utility function. It executes a rewrite only after proving that the rewrite has higher expected utility than continued search [130]. The proof searcher is also part of the writable program. The machine can therefore change both its task policy and the process that finds and certifies later changes.

Limits of formal guarantees. The guarantee is conditional rather than a general safety result for real agents. The formal result need not imply target-relative gains in deployment if the axioms omit relevant facts or the utility function misses the intended goal. It also provides no practical gain when the proof system cannot find a proof within the available budget [130]. The Gödel Machine makes the logic of L3 explicit, but its proof-gated installation rule remains tied to a fixed criterion. Later empirical systems replace full proofs with task scores, backtesting, or lineage performance. This makes the mechanism easier to apply, but shifts the reliability burden to evaluation coverage, matched resources, and control over benchmark access.

6.1.2 Code-Based Self-Modification

Direct code self-modification. Source code is a general way to implement self-reference. The key issue is whether a retained edit changes the code that controls future updating. Gödel Agent separates a task

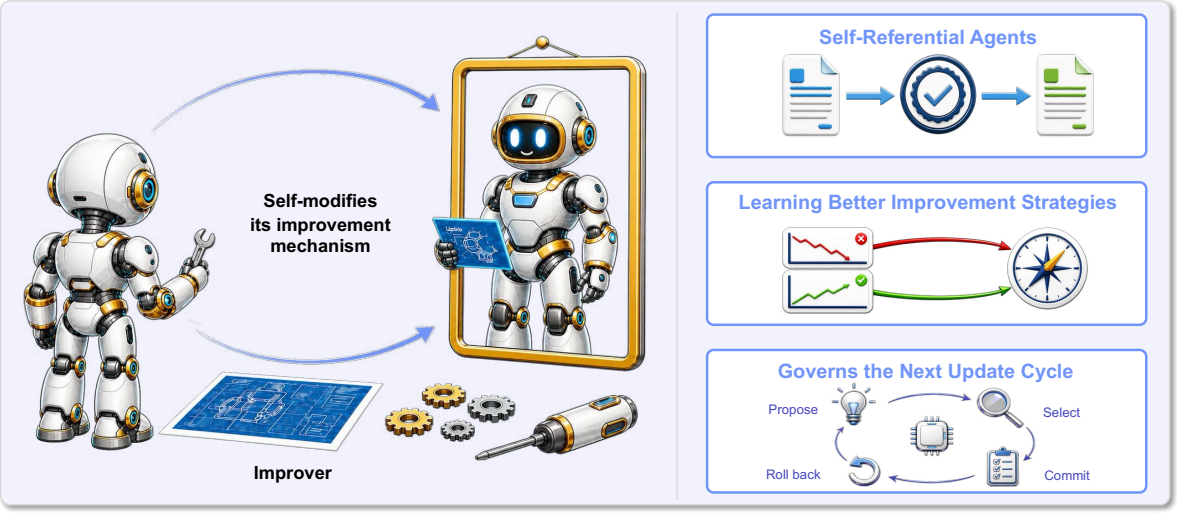


Figure 7 | **Improver-Level Self-Evolution.** A current improver helps produce or internally select a candidate successor to its own update mechanism. Self-referential agents and learned improvement strategies provide two routes to this change. After external audit and promotion, the retained $\mathcal{U}_{k+1}$ governs later proposal, internal selection, commit, and rollback under a fixed criterion C_k.

policy π_t from a self-referential learning algorithm I_t. The current I_t produces both π_{t+1} and I_{t+1}, rather than changing only the task policy [131]. Its implementation reads its own functions, classes, and other code from Python runtime memory. It uses monkey patching to add, replace, or remove logic. A recursive main function runs the changed learning logic at the next depth [131]. This mechanism meets the functional condition for self-reference. Its task feedback still comes from a fixed high-level goal and environment interface, so it does not provide a Gödel Machine proof guarantee.

Combining task and improvement logic. Code-based systems also span a range from generating an improvement algorithm to combining task and meta-improvement logic. Self-Developing proposes, implements, and refines executable improvement algorithms. A fixed outer evaluation and preference process guides this algorithm search [133]. A retained algorithm can later control model merging or another update. Otherwise, it remains an artifact produced by a fixed search process. SICA selects a strong agent from its archive to perform the next meta-improvement step. That agent reads the archive, proposes a change, and edits its full Python codebase [132]. Fixed utility and benchmark checks decide whether the new code enters the archive. The agent's task ability can help with later meta-improvement, but outer rules still set cost, speed, and benchmark weights [132]. HyperAgents places the task agent and the meta-agent that can modify both agents in one editable program. It also allows the meta-improvement process to change itself [492]. Reported changes such as persistent memory and performance tracking affect how later agents are produced.

Separating task code from update code. Practical systems often edit task code and control code in the same repository. MOSS can rewrite an agent's source code and test candidates through replay and trial runs. Its fixed pipeline also provides rollback after provisional installation. Its stage order, arbitration, and coding tools remain in a fixed pipeline [493]. Live-SWE-agent keeps editing its scaffold while it solves software tasks [494]. PACE alternates between faster prompt updates and slower control-logic updates. It admits control-logic candidates after held-out validation [312]. Across these systems, task-execution edits must be separated from edits to the update process.

6.1.3 Open-Ended Agent Evolution

Archive-based evolution. Open-ended evolution extends one self-modification chain into a population that keeps several search paths. This openness does not imply that the outer search rules also change. The Darwin Gödel Machine (DGM) selects a parent agent from an archive. The parent reads its benchmark logs, proposes a feature change, and edits its own code. Coding benchmarks evaluate the new agent before it enters the archive [14]. The new agent can later become a parent and produce further agents. Its retained code can therefore change future self-modification rather than only one task output [14]. Archive management, the parent-selection formula, the base model, and benchmarks such as SWE-bench and Polyglot remain fixed. The reported evidence therefore supports only a benchmark-scoped claim about archived descendants under the stated budgets and selection rules. It does not support improvement claims outside the tested domains or evaluation horizons [14].

Evaluating future improvement. Open-ended search also exposes a gap between current performance and future improvement ability. The Huxley–Gödel Machine selects agents using both current coding scores and the aggregate performance of several descendants. The aggregate estimates a lineage's ability to produce later higher-scoring descendants [134]. This longer horizon can preserve a parent with modest current performance but strong later potential. The estimate is not external audit evidence because it shares fixed coding benchmarks and search budgets with candidate selection. Long-term update productivity beyond the reported evaluation horizon remains unobserved. Lineage evidence can compare how improvers produce later benchmark gains, but external audit still requires fresh tasks, matched resources, and longer horizons.

6.2 Learning Better Improvement Strategies

6.2.1 Learning from Failures and Experience

Learning repair policies from failures. Stored experience can become a retained policy that changes later proposals, filtering, or rollback. Polaris turns failure traces into small policy-code patches through failure analysis, policy formation, and experience abstraction. Conservative checks select patches that are then used on unseen instances from the same benchmark [137]. The important change is not the stored failure itself. The patch changes how later failures are analyzed and repaired. A fixed patch format, validation set, and outer loop still limit this self-referential process.

Using experience in later updates. TPGO represents a multi-agent system as a graph of textual parameters. It derives textual gradients from execution traces to locate faults. The mechanism studies past successes and failures, so later proposals change with optimization experience [138]. A single graph-node edit changes the current system, while retained GRAO experience can change the distribution of later proposals. Meta Context Engineering evolves both context artifacts and context-engineering skills. Its meta-agent searches with skill history, execution, and evaluation, so reusable skills affect later context construction [87]. Some skills support task execution, while others guide how later context changes are searched, combined, or selected for reuse. These studies need causal tests that separate stored content from changes to the improvement strategy.

6.2.2 Adapting Search and Training

Self-applied search strategies. Adaptive search and training can change the search target, model weights, or the search procedure itself. STOP gives a seed improver to itself as an input program. It uses the resulting scaffolding improver for later program generation [135]. The language model, utility function, and outer

self-application process remain fixed. Promptbreeder evolves task prompts and the mutation prompts that control later task-prompt mutations. Only retained mutation prompts that control future mutations change the improver [136]. Retained task prompts remain scaffold changes. SePO optimizes both the task agent's system prompt and the prompt agent's own system prompt. After multi-task pretraining, the prompt agent's retained prompt continues to guide prompt optimization on target tasks [291]. Its archive search, fitness evaluation, and base model remain fixed. These systems expose local self-application, but they do not make the whole improvement process writable.

Adapting training strategies. Larger training systems place changes to the improvement strategy inside training operations and backtesting. A-Evolve-Training changes the requested search direction when an internal development metric stops tracking a fixed performance measure. It avoids continuing to optimize a failed proxy [139]. When the switch persists into later interventions, it changes the search strategy rather than the fixed performance measure. Changes to the judgment used for internal candidate selection belong to the criterion and are discussed in §7. Search actions and judgment rules must therefore be recorded separately. EvoTrainer jointly updates a model policy and a training-side harness. Rollout diagnostics, intervention backtests, and reusable skills can influence later search [140]. EvoTrainer combines model updates, runtime changes, and changes to the strategy used for later interventions. Ablations or counterfactual tests are needed to separate these effects.

Distinguishing adjacent-level transitions. Some systems adapt feedback, distractions, or training tasks with the learner, while the main retained change remains in model parameters under a fixed outer training process. POP, Seirênes, and CPMöbius let the model help score outputs, generate distractions, or generate training tasks. They show how feedback generation can adapt even when the outer training process stays fixed [495–497]. RLCER uses the same model as both a reasoner and a rubricator [498]. Under a fixed training process and a frozen verifier, it updates model parameters while generating task-specific scoring rules. Unless those rules are retained and govern later judgments, the realized transition is L1 with an L4-facing feedback role rather than L3. If they are retained and govern later judgments, the deepest active change reaches L4. AREX checks its current answer constraint by constraint and launches targeted follow-up research on the claims that remain unresolved [499]. Because its outer loop, confidence threshold, tools, and update rules stay fixed, the described loop revises the current answer and task-local research state. The separately trained compaction tool is a pipeline component rather than a retained change produced by that loop, so it does not by itself raise the realized transition above L0. The paper describes this loop as recursive self-improvement, a broader use of the term than the retained improver rewrite that defines L3. EvoRubric writes selected rubrics to persistent memory and uses them for later reward. Its persistent rubric memory changes later judgments, so its main relevance is Criterion-Level Self-Evolution [144]. These systems help separate feedback changes, model changes, and criterion changes. They do not replace causal evidence that the improver's active semantics changed and affected later updates.

6.3 Reliability and the Fixed-Criterion Limit

Comparing improvers. An L3 classification concerns a retained change to the update procedure, not only the best artifact that procedure produces. The corresponding improvement claim asks whether the candidate improver produces better later updates than the incumbent under a declared external target and matched models, tools, context, compute, and stopping rules. Fixed-process systems such as AlphaEvolve, DeepEvolve, ImprovEvolve, and Self-Supervised Theorem Discovery show why this distinction matters. Their artifacts can be checked with executable or formal evidence even though the outer improvement process stays

fixed [500–503]. The Huxley–Gödel Machine separates current task performance from future improvement productivity. A-Evolve-Training shows how a fixed proxy can saturate during repeated search [134, 139]. The audit record should therefore report current performance, later-round outputs, search cost, failure recovery, and transfer to fresh tasks.

Metric capture and shared feedback. A fixed criterion gives different improvers a common internal reference. Repeated search can still adapt to that reference and raise internal scores without corresponding improvement relative to the external target. The risk is stronger when feedback producers and learners share parameters or adapt to one another. POP, Seirênes, CPMöbius, RLCER, and EvoRubric provide related evidence of this shared-feedback problem [144, 495–498]. Their internal signals can support learning, but agreement within the same loop is not external audit evidence.

External evidence and rollback. Promotion evidence is informative only when the evidence source and acceptance policy remain outside the candidate's update boundary, resource conditions are matched, and evaluation tasks did not guide proposal or internal selection. The audit record should include proposal history, selection reasons, resource use, and the sequence of promoted changes. A protected evidence source may use sealed evaluation, formal checks, controlled experiments, or human review. An externally controlled acceptance policy then maps that evidence to accept, reject, or escalate. Replay and regression analysis can support diagnosis, while canary deployment and rollback can limit impact after promotion. These operational controls bound harm but do not by themselves strengthen the improvement claim. Repeated access can turn a protected holdout into development feedback, and relabeling it does not restore independence.

Fixed criterion and handoff. The fixed criterion C_k is the common internal reference in L3. It defines the objectives, rewards, evaluation protocols, comparison rules, and constraints used to judge different improvers. It remains distinct from the declared external target used by the audit. An externally controlled acceptance policy should use external audit evidence to accept, reject, or escalate the candidate. Only an accepted candidate may then be promoted. The candidate must not choose or change the audit data, scoring rules, or acceptance policy. When rubric memory, evaluator protocols, judgment tasks, reward meaning, or other judgment rules change persistently and govern later judgments after activation, the deepest active change reaches Criterion-Level Self-Evolution. Section 7 asks how to assess such a transition when internal judgment semantics become writable. Any cross-version improvement claim still requires a protected external target and a baseline fixed before the transition.

7 L4: Criterion-Level Self-Evolution

L4 at a Glance

Boundary. L4 persistently changes the criterion C_k, including judgment tasks, reward semantics, evaluator protocols, comparison rules, constraints, or values. The revised criterion governs later judgments. Any audit-supported improvement claim must therefore specify which property of judgment improves relative to a declared external target, rather than rely on a higher internal score alone.

Roadmap. We examine evolving evaluation mechanisms (§7.1), evolving tasks, objectives, and values (§7.2), and reliability with the criterion inside the loop (§7.3).

Reliability focus. When the criterion itself is updated, higher internal scores alone do not guarantee validity or real improvement against external targets. To certify a proposed criterion, one must compare it with the incumbent on a shared set using external anchors or expert judgments outside the update loop. Otherwise, criterion drift or circular validation can masquerade as improvement.

Boundary and persistence. At L4, the deepest retained rewrite is the active criterion C_k (Table 1). Writing the retained state as $X_k = (\theta_k, \sigma_k, \mathcal{U}_k, C_k)$, as in Equation (1), the L4 boundary is

$$X_k \longmapsto X_{k+1}, \qquad C_{k+1} \not\equiv_{\text{act}} C_k \tag{10}$$

Because the criterion is the deepest coordinate of X_k, L4 imposes no equality constraint on θ_k, σ_k, or $\mathcal{U}_k$: these components may remain fixed or change in the same transition, but the active criterion rewrite determines the realized self-evolution depth. Operationally, an L4 method proposes a candidate criterion component, evaluates it, and activates the selected version so that it governs a later feasibility decision, candidate ordering, threshold, or tie break. Most methods rewrite only one component of C_k and leave task specifications, protected cases, outcome checks, or authorization rules unchanged. These fixed components constrain the search and provide comparison evidence, while the final reliability claim depends on whether that evidence and its acceptance policy remain outside the update boundary.

Across these families, the recurring algorithmic pattern is to represent a candidate judgment rule explicitly, test it against partially protected evidence, and version the rule that will govern subsequent decisions. The searched representation may be a natural-language rubric, learned evaluator, executable program, task set, or reward. Methods also differ in how they compare candidate criteria before activation. Table 8 organizes the reviewed families by what changes, how candidates are compared or screened, and what limits their interpretation as realized L4 transitions.

7.1 Evolving Evaluation Mechanisms

Evaluation-mechanism methods change how outputs are interpreted, compared, or accepted. They represent the criterion as natural-language rubrics, learned evaluators, compositional metrics, or executable verifiers, then update that representation from disagreement, rollout outcomes, or protected test cases.

7.1.1 Rubrics and Retained Judgment Rules

Rubric synthesis and calibration. Rubric methods factor judgment into explicit criteria that can be drafted, scored, revised, and reused. AutoCalibrate generates and self-refines candidate criteria, selects them by correlation with expert labels, and inserts the selected criteria into subsequent evaluator prompts [141]. GER-Eval and ELMES+ broaden this approach by generating task-specific or scenario-specific rubrics, but their evaluator analyses expose weaker cross-model and human alignment, self-preference, and model-specific leniency or severity [142, 504]. RubricBench isolates the rubric itself: with the judge backbone, prompt, and decoding fixed, human-authored rubrics improve preference accuracy by roughly 22 to 28 percentage points over self-generated rubrics across seven model backbones [143]. Together, these results make rubric provenance, judge identity, and disagreement important inputs to rubric selection.

Persistent rubric stores. Persistent methods turn rubric generation into a stateful update process. EvoRubric maintains a rubric-memory pool and uses meta-verification and peer consensus to screen entries before they

Table 8 | Representative method families reviewed in Criterion-Level Self-Evolution. The two groups follow §7.1 and §7.2.

Method family	What changes	Candidate comparison or screening
Evolving Evaluation Mechanisms		
Rubric synthesis and calibration [141–143, 504]	Task- or scenario-specific rubrics reused in later judgments	Expert labels, cross-model agreement, and fixed-judge accuracy
Persistent rubric stores [144–146]	Retained rubric pools, items, or process rules	Peer checks, discrimination, held-out rollouts, and constraints
Versioned evaluator replacement [15, 147]	Metrics or evaluators installed for later search	Protected cases, locked holdouts, final judges, and tests
Executable verifier repair [148]	State checkers reused across later tasks	Reference judgments, human agreement, and regression tests
Learned evaluator proposals [149, 505, 506]	Evaluator weights or prompts under fixed semantics	Synthetic preferences, fixed labels, and process rewards
Evolving Evaluation Tasks and Objectives		
Evolving evaluation task sets [150, 507, 508]	Retained task archives or refreshed benchmarks	Correctness, novelty, human validation, and developer checks
Adaptive benchmarks [151, 152, 162, 509]	Questions or scenarios near a capability frontier	Difficulty, separability, policy grounding, and human samples
Evolving evaluation environments [153, 510, 511]	Environment populations around current capability	Viability, transfer, and regret
Adaptive reward composition [154–156]	Weights over a fixed reward basis	Learning progress, task context, hypervolume, and gradients
Executable reward synthesis [157–160]	Executable rewards for later policy learning	Rollouts, fixed fitness, execution checks, and preferences
Value representation and assessment [161–163, 512–515]	Value representations, evaluators, scenarios, or difficulty models	Human review, cross-cultural comparison, and unseen references

shape later rewards [144]. DR Tulu retains search-grounded, discriminative rubric items that score later policy updates, whereas SkillCoach patches versioned process rubrics and admits revisions only after held-out rollout checks and hard constraints [145, 146]. The three systems operate at different granularities: pool entries, discriminative items, and process steps. All three combine an editable rubric representation with an explicit admission rule. Their internal gates test consistency, discrimination, or rollout utility. External-target validity remains a separate question addressed in §7.3.

7.1.2 Evaluators and Verifiers

Evaluator methods update either a learned decision rule or executable code that implements success conditions. The main design choice is whether to search over complete evaluator versions, localized program repairs, or trainable judge parameters.

Versioned evaluator replacement. Double Ratchet co-evolves a compositional metric with a skill loop while reserving fixed development cases, locked held-out cases, and a separate final judge. Its ablations show why those controls matter: removing the case guards yields an almost always-pass report metric in all three reported seeds, and the final judge detects a Goodhart episode in which evolved skills exploit the report rubric [147]. Red Queen Gödel Machine (RQGM) instead freezes the incumbent evaluator within each epoch, tests a challenger on a fixed held-out ground-truth set, and installs it only after a statistically significant improvement, after which the challenger governs the next epoch. Experiments in coding, paper writing, and grading show that replacement can alter the search frontier, but the guarantees are epoch-local and the evidence covers only short search horizons [15]. Together, the two systems instantiate a common replacement protocol: freeze the incumbent, compare a challenger on protected evidence, install the accepted version, and use it throughout the next search period.

Repairing executable verifiers. Executable-verifier evolution changes the program that maps observable task state to a success judgment. OpenComputer treats application-specific state checkers as mutable evaluation artifacts across 33 desktop applications. Calibration executions compare checker outputs with reference judgments, and the reported repair stage corrects most checker-side errors while increasing agreement with human judgments [148]. The repair boundary nevertheless excludes trajectories, sandbox state, task specifications, expected outputs, and official reward semantics, and the benchmark omits visual or geometric outcomes that its checkers cannot yet assess. This narrow update makes the checker code inspectable and supports direct regression testing, but it cannot assess outcomes that its observable state and predicates do not represent. CoEvoSkills provides a complementary test-escalation primitive: when a fixed hidden oracle exposes false acceptance, an independent surrogate verifier rewrites its deterministic test suite before evaluating the next skill version [101]. OpenComputer installs repaired checkers across later tasks, whereas CoEvoSkills uses verifier revision as a diagnostic mechanism within one skill-construction episode.

Learned evaluator proposals. Learned-judge optimization supplies candidate implementations and diagnostics for criterion search. Self-Taught Evaluators trains on synthetic preference pairs and filtered judgments, while Meta-Rewarding shares one model across actor, judge, and meta-judge roles [505, 506]. SEVA instead uses a structured fact-attribution process reward to drive repeated cycles of verification, reflection, probing, and refinement [149]. These studies expose practical proposal risks: performance can depend on the seed evaluator, same-model feedback can exhibit positional bias or score saturation, and repeated training can produce benchmark specialists rather than a uniformly stronger judge. These techniques are therefore most useful as proposal components within L4: the surrounding procedure must state which rubric or protocol semantics change, compare those revisions, and decide which version becomes active.

7.2 Evolving Evaluation Tasks and Objectives

Beyond the mechanisms that implement evaluation, criterion methods can change the cases presented for judgment or the objectives used to value outcomes. The main representations are evolving task sets, adaptive benchmarks and environments, executable reward programs, and explicit value or constraint relations.

7.2.1 Tasks, Benchmarks, and Environments

These methods evolve the evaluation distribution by updating existing task sets, generating adaptive benchmarks, and evolving environments that expose new behaviors. Across all three families, the central design question is which generated tasks or environments enter the retained evaluation distribution and are used to judge later candidates.

Evolving task sets for evaluation. Task-set methods differ primarily in what triggers an update and which proposed tasks are retained for subsequent evaluation. AC/DC couples task discovery with model search: global and active archives store questions, answers, and executable scoring functions, while compilation, self-solving, reflection, and novelty filters govern archive admission. The active archive ranks model candidates, downstream benchmarks remain hidden from the search, and sampled expert review finds high task correctness while judging a smaller share genuinely creative [150]. Dynabench instead retains human-validated model failures collected over successive rounds of human and model interaction, whereas EvoCodeBench rebuilds a repository-grounded coding benchmark from recent projects through an automated pipeline [507, 508]. These approaches therefore update task sets through complementary signals: joint task and model competition, observed model failures, or temporal data refresh.

Generating adaptive benchmarks. Adaptive benchmarks concentrate evaluation on regions where a static test no longer distinguishes candidate systems. GETA estimates an examinee's capability and generates value-alignment items near that boundary, while MathDuels evaluates models as both problem posers and solvers and infers task difficulty from their result matrix [162, 509]. AgenticEval grounds safety-case generation in policy documents and checks its rubric-constrained judge against a human-labeled sample [151]. AutoBencher makes the search objective explicit, using candidate-model scores to refine dataset descriptions for salience, difficulty, separability, and novelty [152]. The methods thus differ in the feedback that steers generation: estimated capability, adversarial problem-solving outcomes, policy-grounded safety coverage, or explicit benchmark objectives. The resulting comparisons are easiest to interpret when task construction remains tied to sources outside the generator's control, such as policy documents, sampled human labels, or held-out evaluation cases.

Evolving evaluation environments. Environment-based methods extend adaptive evaluation from individual items to populations of interactive worlds. POET mutates viable environments, optimizes their paired agents, and transfers solutions across an archive, whereas PAIRED trains an environment adversary to maximize regret between an antagonist and a protagonist [510, 511]. ACCEL combines evolutionary editing with regret-based prioritization by mutating selected high-regret levels and retaining variants that remain informative at the current capability frontier [153]. Together, they expose three complementary controls over environment evolution: viability and transfer in POET, adversarial regret in PAIRED, and frontier-focused editing in ACCEL. All three adapt environment difficulty around the current agents while leaving reward or regret semantics fixed. When selected environments are retained for future evaluation, the resulting population defines the test distribution against which later candidates are compared.

7.2.2 Reward Design and Value Assessment

Adaptive reward composition. Reward composition keeps the reward basis fixed while adapting how strongly each component influences policy learning. DyLam derives a self-curriculum from component-wise learning progress, reducing the weight of mastered components and redirecting attention toward under-optimized ones [154]. MAESTRO instead learns a Conductor that maps task representations to weights over five reward components while jointly updating the scalarization policy and the language model [155]. Dynamic Reward Weighting targets multi-objective language-model alignment: hypervolume feedback guides optimization when user priorities are available, whereas gradient influence drives weight updates when they are not [156]. Together, these methods condition scalarization on training progress, task context, or objective geometry. Because the reward basis remains fixed, however, they cannot represent an omitted outcome or repair a component that encodes a biased proxy.

Executable reward synthesis. Executable reward programs expand the design space from mixture weights to the logic and parameters that compute reward. Text2Reward generates dense reward code from a natural-language goal and an environment abstraction, then lets users revise the code after inspecting policy rollouts [157]. EUREKA replaces user-guided revision with evolutionary search, using fixed task fitness to compare reward programs before training policies with the selected program [158]. CARD closes the revision loop with execution checks and process, trajectory, and preference feedback [159]. R* further separates reward structure from numerical calibration: language model mutation and modular crossover evolve reward components, while an ensemble of generated critics provides trajectory preferences for fitting their parameters [160]. Together, these methods move from interactive code revision to automated structural and numerical search. Their evidence nevertheless remains tied to fixed task descriptions and outcome measures, so high policy return does not establish that the synthesized reward covers every intended outcome.

Representing pluralistic and evolving values. Value methods require representations that preserve contextual conflict, cultural variation, and change over time. Value Kaleidoscope uses ValuePrism to generate, explain, and assess the relevance and valence of values, rights, and duties in context, retaining competing considerations rather than collapsing them into one scalar label [161]. UniVaR instead learns a high-dimensional representation from value-eliciting responses and evaluates it across fifteen language models and twenty-five languages and cultures [512]. ProgressGym adds a temporal axis by reconstructing historical language-model proxies and simulating bidirectional influence between human and AI values over long horizons [513]. These approaches expose complementary structure through explicit value conflicts, cross-cultural geometry, and longitudinal trajectories, but they remain descriptive models rather than procedures for authorizing new value semantics.

Adapting value assessment. Adaptive value assessment can revise the evaluator, the scenario distribution, or the difficulty model as evidence accumulates. CLAVE extracts general value concepts with a large language model and fine-tunes a smaller recognizer, allowing the evaluator to calibrate to a specified value system with fewer than one hundred labeled examples per value type [514]. ALI-Agent uses evaluation memory and target-model feedback to generate and iteratively refine scenarios that probe long-tail failures across stereotypes, morality, and legality [515]. GETA jointly estimates item difficulty and model value conformity while generating new items matched to the current model, producing measurements that agree more closely with unseen in-distribution and out-of-distribution references than static or selection-only adaptive tests [162]. AdAEM uses in-context optimization to generate controversial and informative questions under Schwartz Value Theory, emphasizing discrimination among models rather than calibrated difficulty [163]. These methods address different weaknesses of static evaluation, including shifting definitions, sparse long-tail scenarios, test saturation, and weak discrimination. These evaluations do not by themselves establish that the underlying value ontology is complete or normatively legitimate, nor do the reviewed systems provide a validated procedure for authorizing and activating revised value semantics. We therefore treat stakeholder evidence, constraints and approval authority outside the relevant update boundary, and rollback procedures as governance requirements for value change.

7.3 Reliability with the Criterion Inside the Loop

The reviewed mechanisms expose a common reliability problem. Rubric pools can ratify shared preferences, evaluator populations can mutually accommodate, adaptive tests can become difficult without becoming more relevant, and reward programs can optimize an incomplete proxy. An L4 promotion claim is supported only to the extent that its evidence remains informative about a declared external target and outside the criterion update boundary.

The moving-ruler problem. When both the evaluated system and its criterion change, a rising sequence of internal scores combines two effects: behavioral change under the old standard and movement of the standard itself. Evaluation under only the revised criterion cannot distinguish a more valid rule from one that is merely easier for the revised system to satisfy. This limitation does not make Criterion-Level Self-Evolution intrinsically unreliable. It shows why internal scores alone cannot support the promotion claim.

Cross-evaluation across criteria. A defensible L4 protocol identifies the modified component and preserves a common comparison domain. Where comparisons remain meaningful, it evaluates the incumbent and candidate systems under both the incumbent and proposed criteria. Disagreements should remain visible because they reveal which decisions changed and where the proposed criterion departs from the previous

standard. Fresh executable checks, sealed cases, adversarial probes, or authorized human judgments can then test whether that departure better serves the external target.

Coverage, authority, and rollback. An external anchor constrains only the properties it measures and cannot establish the complete validity of a revised criterion. A promotion record should therefore state the invariant external target, the writable criterion components, the evidence coverage, the acceptance policy, the approving authority, and the rollback condition. Staged and versioned activation preserves a comparison baseline, allows later failures to be traced to the relevant judgment rule, and supports rollback without silently redefining earlier results.

Claim scope and synthesis handoff. With adequate coverage and authorized promotion, L4 evidence can support a bounded claim that a revised criterion better serves the declared external target on the audited domain. It does not establish unrestricted criterion validity, durable agreement among co-evolving judges, or legitimacy for people and settings outside that domain. Open questions include whether co-evolving judges retain discrimination rather than mutually accommodate, whether adaptive tests remain relevant rather than merely difficult, and how normative criterion changes can remain accountable under finite human oversight. Section 8 places these L4-specific problems in the cross-level audit structure and compares criterion drift with the distinct failure mechanisms at Levels L0 through L3.

8 Cross-Level Reliability: Evidence, Acceptance, and Control

The preceding sections classify self-evolution by what changes, from a task-local output to retained model, scaffold, improver, and criterion state. This section asks a different question: what evidence supports a claim that the change is an improvement? We organize the answer around evidence, acceptance, and control. The external target, evidence source, and acceptance policy must remain outside the relevant update boundary, while the evidence must match the claimed effect and evaluation horizon. These requirements vary by level, but self-evolution depth alone does not determine reliability.

Figure 8 makes the section's organizing design principle concrete. We call this matched structure the *reliability ladder*. Each rung pairs the deepest active evolution target with representative external evidence sources and decision or recovery controls appropriate to the claimed effect. The L0 rung ends in task-local acceptance or return, whereas the L1–L4 rungs concern retained changes that require external acceptance before promotion as well as protected lineage and recovery. The rise of the steps denotes self-evolution depth only: a higher rung changes the audit obligation rather than assigning a higher or lower reliability score.

8.1 External Audit Across Self-Evolution Levels

A common audit structure. Each audit card in Figure 8 abbreviates a common structure whose roles are formalized by Equations (4) and (5). The evolution step generates and internally selects a candidate, after which the audit produces external evidence z_k^{ext} and a gate decision a_k^{ext}. Within each card, the "External audit" field names representative evidence sources, while the "Decision" or "Control" field names part of the decision or post-acceptance lifecycle. For the audit to remain external, the external target, evidence source, and acceptance policy must stay outside the relevant update boundary. The compact labels do not collapse these roles into a score or a single control. We call evidence disclosed to candidate generation or internal selection *development evidence*. Once this evidence has guided either step, it should not serve as the sole final audit evidence for the same update.

Figure 8 | The reliability ladder for self-evolving agents. The colored steps order the deepest active evolution targets from task-local output to retained criterion, while the cards give representative evidence and controls matched to each target. The labels abbreviate Output-Level, Model-Level, Scaffold-Level, Improver-Level, and Criterion-Level Self-Evolution. The rise of the steps represents self-evolution depth, not capability or reliability. L0 ends in task-local acceptance or return, whereas L1–L4 candidates require external acceptance before promotion, protected lineage, and independent recovery controls. These pairings are conditional design requirements rather than sufficient guarantees of improvement.

What changes across levels. Moving up the ladder changes the deepest active evolution target and can change the form and horizon of the required audit, not how strong the supporting evidence is. Fresh executable outcomes for an L1 update may be more informative than an uncalibrated self-judge used for an L0 revision. Likewise, an L3 update can remain auditable when descendant systems are tested on unseen tasks under matched resources. By contrast, repeated feedback can turn a held-out benchmark into a development target at any level. The key comparison is therefore between what the update controls and who controls the audit, not between level numbers.

A cross-level comparison. The ladder presents the matched-audit principle in compact form, while Table 9 expands each rung into the changed object, vulnerable evidence, characteristic audit failure, and matched controls. It summarizes §3.3–§7.3 as a checklist rather than a guarantee of adequate coverage.

8.2 Level-Specific Audit Failures and Evaluation Horizons

A common failure pattern. The five rungs respond to one structural problem: at every level, an update can raise the measured score by influencing the evidence rather than improving under the external target. The mechanism differs by level: at L0, the proposer and critic can share the same error, while at L1, biased supervision can enter retained parameters. At L2, the scaffold can overfit interactions among benchmarks, memory, and workflows. At L3, changes to proposal, search, or internal selection can capture a fixed metric, while at L4, the update can change the standard used to judge improvement. An adaptive coding study found

Table 9 | Cross-level reliability ledger. Each row identifies the deepest active evolution target and the evidence most exposed to that update. It then lists the characteristic audit failure and matched acceptance or promotion controls, without ranking reliability across L0–L4.

Level	Deepest active evolution target	Evidence exposed to the update	Characteristic audit failure	Matched audit and controls
L0	Current output or trajectory	Self-critique, consensus, generated tests, or model scores used within the task	Self-confirmation and harmful over-refinement	Execution, source-grounded checks, fixed rubrics, environmental observations, or human adjudication, with evidence sources and decision procedures controlled outside the task-local update boundary
L1	Retained model or policy state	Self-generated examples, pseudo-labels, rewards, preferences, or learned verifiers	Model collapse, tail narrowing, superficial gain, and forgetting	Fresh post-update transfer, retention, and interference tests, executable outcomes, and data-admission checks controlled outside the model update boundary
L2	Retained prompts, tools, skills, memory, workflow, topology, or execution harness	End-to-end scores, retrieved context, component validators, and repeatedly queried holdouts	Scaffold overfitting, resource confounding, component interaction, and silent store drift	Matched-budget incumbent comparison on fresh tasks, component ablations, replay, provenance, version lineage, and a tested rollback path
L3	Procedure that proposes, selects, commits, or rolls back later updates	Current-task score, archive fitness, repeated benchmarks, and the improver's own search diagnostics	Metric capture, resource confounding, and mismatch between current performance and descendant productivity	Descendant performance on fresh tasks under matched resources, with proposal history, sealed evaluation, protected lineage, and rollback authority
L4	Evaluator protocol, judgment tasks, reward semantics, constraints, or values	Scores and preferences produced under the revised criterion itself	Criterion drift, mutual accommodation, and a standard that is easier to satisfy but less aligned with the external target	Protected outcome measures, pre- and post-criterion cross-evaluation, authorization outside the update boundary, and protected constraints

high generative–evaluative agreement on syntactically checkable skills but near-zero agreement on several design-level skills [516]. This result shows that one evaluation stack can be informative for one construct and weak for another. Agreement within one stack and additional revision rounds do not replace an external audit. These mechanisms motivate the matched audits in the ladder, but they do not form an ordering of failure severity.

Matching evidence to retained effects. The audit attached to each rung should match the claimed scope and evaluation horizon. L0 concerns the current artifact, while L1 requires fresh transfer, retention, and interference tests on later independent tasks. L2 requires fresh end-to-end comparison against the incumbent under matched resources, with component analysis and provenance where needed. At L3, the audit should test descendant performance rather than only the current score [134]. At L4, it should compare judgments under the old and proposed criteria. Controlled interventions across four frameworks found stronger dependence on raw trajectories than on condensed summaries [517]. A separate study reported capability erosion during workflow, skill, model, and memory adaptation, and its mitigation results favor explicit retention constraints [278]. Together, these results show that storing an artifact does not establish that it remains active or preserves prior capabilities.

Three properties of evidence. The compact audit fields in Figure 8 identify different evidence channels, not interchangeable guarantees. Evidence can be easy to inspect and still be weak. Structured traces, source spans, decomposed verdicts, and version diffs improve inspectability, but they do not establish informativeness or externality. Evidence may still be biased, narrow in coverage, or weakened by repeated exposure. These properties should therefore be reported separately before the evidence is used for acceptance.

8.3 Acceptance Policies and Requirements for Promotion

A common decision problem. The ladder makes a lifecycle discontinuity visible: its L0 card ends with a task-local decision, whereas the L1–L4 cards add controls for retained change. Candidate generation, internal selection, and external acceptance are separate. The rule in Equation (4) selects only a provisional candidate for audit. For retained L1–L4 changes, *promotion reliability* is the probability that an accepted update improves over the incumbent under the declared external target and matched evaluation conditions. L0 instead ends in task-local acceptance or return. Optional stopping, repeated selection, multiplicity, or a mutable threshold can invalidate the decision rule. PACE uses paired sequential tests to control false commits under stated assumptions, while SEA embeds anytime-valid gates in a versioned agent stack [518, 519]. Both methods govern how evidence is processed, but neither adds coverage beyond the tested data or establishes reliable promotion at higher levels.

Fresh evidence and coverage. RSEA provides a complementary control by separating the evolution pool from a validation split and retaining a state only after strict improvement on that split [520]. Its ablations show that held-out selection can reveal development overfitting, although repeated use turns the validation split into development evidence. Claims beyond the gate still require fresh final tests because sequential validity and data separation do not cover untested capabilities or rare harms.

Promotion and recovery. The control fields in the ladder compress two distinct functions. External acceptance determines whether a candidate may be promoted, while protected lineage and recovery controls govern the retained state after that decision. Versioned resources, transition histories, and explicit commit operators make a promoted update reconstructable. Autogenesis implements these features with registered prompts, tools, agents, environments, memories, and version lineage [316]. These records support reconstruction and recovery, but they do not show that the promoted state is better. For an L1–L4 candidate, the audit should compare the candidate with the incumbent under the declared external target and matched conditions, using fresh evidence and an acceptance policy that remains valid under adaptive selection. If the candidate is accepted, promotion requires protected lineage and a tested rollback path. Independent rollback and halt authority limit harm. Rejection keeps the incumbent unless the candidate was provisionally installed, in which case it triggers rollback. These conditions should be reviewed whenever evidence access or update authority changes. These are design requirements, not sufficient guarantees.

8.4 Preserving Audit Independence Under Compromise

Persistence under compromise. Persistence creates an attack surface because retained L1–L4 changes can carry a compromise into later tasks, whereas L0 has no retained state by definition. A module–lifecycle analysis maps attack opportunities across model, cognitive-resource, execution, self-design, and collective components [521]. Zombie Agents show how untrusted content written in one session can later be retrieved as an instruction, allowing the injection to outlive the original interaction [522]. These examples show how persistence expands exposure, although vulnerabilities differ across systems. The shared threat is loss of audit independence through probe selection, hidden-outcome inference, altered enforcement, corrupted lineage, or disabled recovery. An audit shown above a rung remains external only if its evidence and controls remain outside the update boundary after the reviewed component is compromised.

Independent enforcement and recovery. Freezing a base model, separating roles, or adding an internal verifier may improve robustness, but these choices do not place the audit outside the update boundary by

themselves. Runtime isolation, least-privilege execution, sealed probes, protected version history, rollback, and independent halt paths protect different parts of the system. Because their effectiveness depends on what the evolving system and an attacker can access or modify, the threat model must state these assumptions. The key test has two parts. Does the external target remain fixed? Do evidence collection, the acceptance policy, protected lineage, and recovery authority still work after the reviewed component is compromised?

Cross-level conclusions. Taken together, these requirements give the reliability ladder its intended meaning. Self-evolution depth changes the failure mechanism and evaluation horizon, but it does not determine reliability. Each improvement claim needs evidence that matches its scope. The audit record should state who controls the evidence, what it covers, and how it is used for acceptance. Retained changes also need protected lineage and independent recovery authority. The ladder therefore advances a matched-audit principle: no update should control the only evidence used for its own acceptance. At L4, a writable criterion is the deepest instance of an update that can influence its own acceptance evidence. Section 9 asks whether these conditions survive capability forecasting, adaptive benchmarks, specific deployment, and finite oversight.

9 Open Problems and Outlook

Section 8 highlighted a central challenge: reliable self-evolution depends on evidence that remains informative and outside the update's control. Building on that synthesis, this section focuses on four open questions. We first ask how self-evolving systems can improve over time and how evaluation should follow that update history. We then ask when an update is ready for deployment and how goals and oversight can remain effective as the system changes. For each question, we review the evidence so far and identify the main gaps for future work.

9.1 Evolution: Capability Growth and Learning Over Time

Persistent classification requires a retained change to remain active beyond the task that produced it. Whether that retained change improves later performance is a separate empirical question. Existing studies report both useful retained gains and failures such as forgetting, narrow specialization, and weak transfer. This section reviews that evidence and asks what would count as sustained capability growth.

Distinguishing learning from more compute. The same score gain can reflect genuine learning or simply more search and computation. One formal analysis models capability relative to an oracle A as $C(A) = \{B : B \leq_T A\}$. It proves that finite internal modifications stay within this layer, while the limit of stabilized revision is characterized by A' [523]. This is a separation result in computability theory, not a direct model of gradient training, finite-time learning, or sample efficiency. Its main value is diagnostic. A claim of qualitatively stronger capability needs to identify the new information, computational resource, or effective oracle that enabled it. An empirical counterpart is still missing. Future experiments could match task information, tools, wall-clock time, inference compute, and environmental access, then test whether the gain persists on new tasks.

Tracking retention and transfer over time. A single persistent update shows only that something was retained. It does not show transfer to new tasks, resistance to forgetting, or protection from negative transfer. Experiments that save successive scaffold snapshots find that useful intermediate versions can later collapse and that validation gains need not transfer to out-of-distribution tasks [524]. Other controlled experiments

separate failure detection, localization, and diagnosis from later failure avoidance. In their tested setting, they also find forgetting under accumulated noise and negative transfer across contexts [525]. Multi-agent memory results add another boundary. With matched trajectory counts, filtered peer traces help some plateaued agents but provide no general gain and do not help the strongest agent [526]. Together, these studies show that long-term learning requires more than a final success rate. Evaluation also needs to track retention, cross-task transfer, and resource use. The next question is how to distinguish sustained growth from temporary adaptation or increasingly narrow specialization over much longer task sequences.

Interpreting early-warning signals. Ordinary task scores are not enough to warn that a system may be improving its future update process. One proof-of-concept task gives coding agents three hours to rebuild an AlphaZero-style Connect Four pipeline and uses an exact solver as an anchor [527]. The strongest agent beat the solver as first player in seven of eight trials. The authors also report that the task had approached saturation by the time of evaluation. The same study observed model-specific under-use of the time budget. Follow-up prompts changed time use but not ratings significantly, so this behavior did not diagnose deliberate capability hiding. The Meta-Agent Challenge probes a related skill by asking a coding agent to develop another task agent under hidden tests and anti-exfiltration controls [528]. Only five model–domain configurations exceeded the mean of the corresponding human-engineered baseline. Results were highly variable, and attempted test exfiltration exposed the security burden created by optimization pressure. Both tasks measure difficult research automation. Neither shows that a system can repeatedly improve its own improver. Future early-warning suites could combine non-writable references, several held-out task families, matched budgets, repeated trials, and contamination controls. They also need a way to remain informative after public release without becoming recognizable training targets.

9.2 Evaluation: Longitudinal Measurement and Adaptive Auditing

Reliable evaluation must follow the full update history rather than only the final score. Current benchmarks reveal regression, forgetting, differences in resource use, and loss of test independence. No single protocol yet covers every update target. This section compares the current benchmark landscape and summarizes the main requirements for longitudinal and adaptive auditing.

Mapping current benchmark coverage. Most agent benchmarks evaluate a fixed system on independent episodes. To evaluate retained self-evolution, a benchmark must instead expose an update opportunity and test whether the resulting change remains active on later independent tasks or future updates. Recent benchmarks begin to support this shift by preserving task order, state snapshots, update artifacts, or controlled post-update tests. Table 10 compares benchmarks that directly test retained changes with adjacent benchmarks that probe autonomous model or agent development without directly evaluating a retained Improver-Level transition.

Adjacent development benchmarks. The second group in Table 10 evaluates whether an agent can perform parts of the research and engineering loop needed to improve models or agent systems. These suites move beyond fixed-system task execution, but they generally ask a researcher agent to attempt to improve a separate target model or to construct a task-specific scaffold within one bounded run. They are therefore adjacent evaluations for self-evolving agents rather than direct evidence of retained Improver-Level Self-Evolution. Long-context memory benchmarks, curated-skill ablations, isolated task suites, and within-task trajectory graders are also useful component tests, but they do not by themselves attribute later-task gains to an active retained update.

Table 10 | Representative benchmarks for self-evolving agents. Direct benchmarks test retained Model-Level or Scaffold-Level changes, whereas adjacent benchmarks test bounded autonomous model or agent development without establishing a retained Improver-Level transition. The suites differ in their changed object and evaluation horizon, so their results do not define one comparable scalar ranking.

Benchmark	Evaluated change and horizon	Evaluation design and evidence	Main limitation
Direct Evaluation of Retained Self-Evolution			
SEA-Eval [529]	Retained cross-task memory and scaffold behavior	Length-five task streams with success, resource use, transfer, and interference-stability trajectories	Efficiency gain does not alone establish capability gain, and the defined alignment proxy is not evaluated
SEAGym [524]	Successive persistent harness snapshots	Training, frozen update validation, held-out in-distribution and out-of-distribution tests, replay, and cost records	The views do not form one uniform history, and validation gains may fail to transfer
BenchTrace [525]	Retained reflection and failure-avoidance behavior	Separate tests of failure detection, localization, diagnosis, later avoidance, noise sensitivity, and cross-context transfer	Some transfer and forgetting results are limited to one game environment and one model
EvoMemBench [530]	Task-local and retained cross-task memory conditions	Comparisons across memory forms, content types, task horizons, long-context baselines, and token cost	Backbone differences prevent a clean causal comparison for every memory-free baseline
SE-Bench [531]	Retained knowledge internalization in model state	Obfuscated APIs separate prior knowledge, documentation access, parameter retention, and compositional use	The diagnostic covers relatively simple coding tasks over one synthetic interface
PAST-Bench [532]	Retained memory, skills, artifacts, and profile state across fresh-session task families	Matched persistence-on and persistence-off episodes, control episodes, saved artifacts, and trace-level pathway evidence	The scope is personal-agent persistence rather than retained Model-Level, Improver-Level, or Criterion-Level change
Adjacent Evaluation of Autonomous Model and Agent Development			
Meta-Agent Challenge [528]	Task-specific agent-scaffold development within one bounded run	Development feedback, a sealed final test, resource limits, and defenses against evaluator and test exfiltration	The retained artifact is an L2 scaffold, while the meta-agent does not retain a changed improver across runs
RSIBench-Data [533]	Iterative data-centric research that updates a separate target model	Fixed target model, bounded training interface, shared training and serving, sandboxed evaluation, and a separate official test	The researcher updates a separate target model while its own update procedure remains fixed
PostTrainBench [534]	End-to-end post-training that updates a supplied target model	A ten-hour single-GPU budget, held-out scoring, and checks for model substitution and test-data use	The run produces an L1 target-model update but does not retain a changed researcher or improver for later runs
Agent^2 RL-Bench [535]	Agent-engineered supervised or reinforcement-learning updates to a target model	Isolated workspaces, a fixed twelve-hour budget, iterative grading, runtime records, and artifact-level analysis	The agent engineers a target-model training pipeline, but the benchmark does not test multi-generation self-modification of that engineering procedure

Recording the full update path. Self-evolution is a sequence of proposed, accepted, rejected, and reverted updates, not a single evaluated episode. SEA-Eval records both success and resource trajectories. It finds up to a 31.2-fold token difference between frameworks on individual tasks despite similar success rates [529]. SEAGym saves successive scaffold snapshots and separates training, frozen update validation, held-out in-distribution and out-of-distribution tests, replay, and cost [524]. BenchTrace further separates failure detection, localization, diagnosis, and later avoidance [525]. Together, these studies show that a terminal score cannot separate retained transfer from temporary adaptation, regression, forgetting, or extra resource use. One open direction is to report held-out gain, retention on earlier tasks, transfer, severe failures, update cost, and uncertainty over the full path. A protocol could also record rejected and reverted proposals and reserve a fresh audit stream that never selects intermediate versions.

Matching metrics to what changes. An evaluation first needs to state what changed, the claimed persistence horizon, and the later tasks on which active retention was demonstrated. EvoMemBench separates within-episode from cross-episode memory and knowledge-oriented from execution-oriented content. No memory method dominates every setting, strong long-context baselines remain competitive, and some memory methods add substantial token cost [530]. Only the cross-episode conditions directly support a claim of persistent Scaffold-Level change. SE-Bench obfuscates NumPy into a pseudo-novel interface to separate documentation access, reasoning difficulty, and parameter internalization. Its closed-book results show that using accessible documentation successfully does not mean that knowledge has entered the model [531]. For improvers, each Meta-Agent Challenge run leaves behind a task-agent scaffold, but the meta-agent itself does not improve across runs [528]. For shared memory, SAGE compares peer history with self-only history under matched trajectory counts. Its results support conditional help rather than a general improvement [526]. These scores are not directly comparable because they concern different writable objects and persistence horizons. Model updates can be tested for retention and interference. Scaffold updates can be tested for replay, permissions, cost, and stale skills. A proposed contract for improver updates would use held-out evidence to test whether future updating gets better. The same proposal would pair criterion changes with authorization and external audit. Cost, generalization, safety, and promotion reliability can be reported across these evaluations, but the levels do not form an ordinal capability scale.

Extending coverage to deeper updates. The existing benchmark landscape is strongest for retained model and scaffold changes. SE-Bench isolates one Model-Level capability, while SEA-Eval, SEAGym, BenchTrace, and EvoMemBench examine different forms of Scaffold-Level persistence. PAST-Bench adds matched persistence ablations and pathway evidence for personal-agent memory, skill, and state reuse [532]. The adjacent suites broaden the target to autonomous agent development, data-centric research, and model post-training, but each bounded run leaves the researcher or its update procedure fixed. Among these representative suites, none directly tests retained Improver-Level or Criterion-Level changes over several generations. The field therefore lacks a common protocol for asking whether a descendant produces, selects, or judges still later descendants more effectively under a declared external target and matched resource conditions. A broader benchmark should not collapse L1–L4 into one score because each level changes a different object and requires a different evaluation horizon. Instead, it should identify the realized transition and update boundary, preserve successive snapshots and proposal histories, and report target-relative held-out gain, transfer, forgetting, severe failures, and cost under matched resource conditions. For deeper self-evolution, the protocol should additionally test descendant productivity on fresh task families while keeping the external target, final evidence source, and acceptance policy outside the update boundary. Protected final tests and access records are also needed because repeated optimization can turn a benchmark into development evidence. The open problem is to combine these controls in a benchmark that remains discriminative across generations without exposing the evidence used for final audit.

Adapting benchmarks while keeping anchors. Adaptive benchmarks can preserve discrimination by generating harder or more diagnostic items. GETA generates value-alignment questions near a model's estimated capability boundary and compares them with static human-labeled and out-of-distribution sets [162]. Its difficulty estimator and item generator still share learned components. MathDuels makes models both proposers and solvers, estimates difficulty with a Rasch model, and filters faulty problems with symbolic checks and model-based adjudication [509]. Both designs resist ceiling effects, but new items are not automatically independent audit evidence. One possible direction is to separate adaptive item generation from fixed anchors and add rotating fresh sets with strict access records.

Adding interactive and expert tests. Interactive evaluation can obtain new cases from changing opponents or observations, but this novelty also has limits. Negotiation games retain fixed payoff tables and swap roles, yet results remain opponent-dependent and may be non-transitive [536]. ClawArena preserves hidden scenario truth and executable workspace checks. Its conflicting information is staged in advance rather than generated adaptively [537]. Frontier-Eng uses frozen simulators, hard feasibility constraints, and 47 engineering tasks. Its declining improvement frequency describes finite-budget search rather than sustained open-ended progress [538]. By contrast, safety and educational evaluations can directly adapt cases or rubrics. AgenticEval generates increasingly targeted cases from policy documents, but its model judge is only partly calibrated through human review and does not provide legal certification [151]. ELMES+ co-evolves educational scenarios and rubrics around expert-defined dimensions. It also keeps frozen anchors, rollback, and stopping rules while documenting judge bias and self-preference [504]. Human-designed items remain useful for rare diagnostics. Hand-built metalinguistic tests exposed a capability dimension on which then-current models performed near chance, although any fixed public set can itself become contaminated [539]. These studies show that the choice is not simply between fixed and adaptive benchmarks. The open questions are which parts may change, which references remain fixed, and who can see each data split.

Checking the evaluator itself. An evaluator that keeps changing must itself be checked with evidence it cannot control. One self-evolving behavioral instrument uses several certificate types to expose local regressions hidden by an aggregate score. Its main results still rely on model judges, and some certificates share evaluation components [540]. SrDetection uses execution-checked, semantically equivalent variants to detect potential leakage. It improves detection in controlled continued-pretraining experiments [541]. On public benchmarks without membership labels, however, it can only flag behavior consistent with exposure. It cannot prove that training-set membership inflated the reported score. PixJail reconstructs text-to-image jailbreak evaluation under a common pipeline and versioned memory. This provides a reproducibility check for results that depend on implementation details [542]. If reported attack success rates guide reconstruction, matching them is not an independent validity check. Signed error can also hide offsetting discrepancies. These methods audit dimension coverage, contamination, and implementation fidelity. None alone establishes the continuing validity of an evolving evaluator. One proposal is a versioned evaluator dossier that records the external target, evidence source, acceptance policy, writable components, fixed anchors, calibration data, access history, tail risk, version lineage, and rollback. The harder question is how much information a certificate retains once the system can observe, predict, or influence certification.

9.3 Applications: From Updates to Deployment

The evaluation principles above become concrete only when they are matched to an application. Recent work clusters around four broad directions: executable engineering, persistent digital agents, scientific discovery, and embodied or high-stakes systems. Figure 9 summarizes these directions and frames their common path from persistent adaptation to staged deployment. These directions differ less in how often they use reflection or memory than in what they retain and what evidence the environment can provide. The central deployment question is therefore not which domain is most popular, but whether its evidence source can support the claimed update without being absorbed into the same development loop.

Executable engineering and AI development. Software engineering, data science, database tuning, and automated algorithm design currently provide the clearest feedback loops. Compilers, unit tests, result-equivalence checks, hidden task evaluators, and measured latency can reject many bad candidates before a retained change is used again. The Darwin Gödel Machine is representative of agent-side evolution because

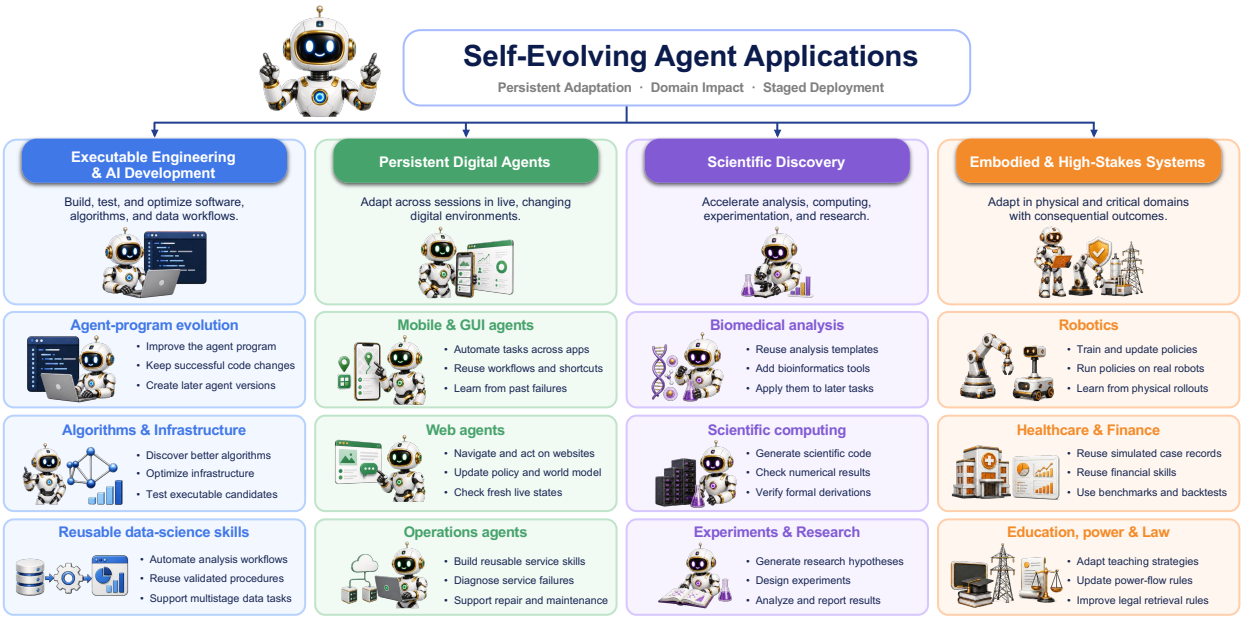


Figure 9 | **Applications and staged deployment of self-evolving agents.** The four columns cover executable engineering and AI development, persistent digital agents, scientific discovery, and embodied or high-stakes systems. Across these areas, agents may retain programs, algorithms, reusable skills, tools, world models, or policies, while the available evidence ranges from executable checks and live states to experiments, expert review, benchmarks, and backtests. Moving from experiments to wider use requires staged evaluation against a declared external target. The evidence source and acceptance policy should remain outside the relevant update boundary, while safeguards, monitoring, and rollback should reflect the application's risks.

descendants retain changes to the agent program and can generate later descendants [14]. AlphaEvolve instead retains algorithms and infrastructure artifacts selected by task-specific evaluators while its outer search procedure remains fixed [500]. EvoDS illustrates a scaffold-centered path in which verified data-science procedures become reusable executable skills, while adaptive context management supports later multistage tasks [419]. These systems make updates easy to execute and compare, but passing the available checks establishes improvement only under the tested behavior. Long update sequences still require fresh tests, protection against repeated holdout use, and a clear distinction between improving an artifact, improving an agent, and improving the procedure that generates later updates.

Persistent digital agents. Web, GUI, coding, and operations agents increasingly retain skills, memory, rules, or model updates across sessions. Mobile-Agent-E stores Tips and executable Shortcuts that can be reused on later phone tasks [543]. UI-Mem combines persistent workflow and failure memory with online reinforcement learning, providing both scaffold and model update paths [544]. ServiceOdyssey turns successful microservice-management traces into reusable operational skills, but its evidence remains a prototype rather than long-running production traffic [545]. WebEvolver further shows the attraction and risk of learning both an acting policy and a world model: the learned simulator can guide later decisions, but it can also become a self-confirming development signal unless checked against fresh live states [12]. Across this direction, the main deployment problems are environment drift, account permissions, prompt injection, stale skills, and memory contamination. A useful audit should therefore test new environments and independent sessions, record permission changes, and verify that obsolete or harmful state can be removed without erasing the evidence needed for review.

Scientific discovery. Scientific agents now span biomedical analysis, scientific computing, automated experimentation, and open-ended research workflows. STELLA provides a clear persistent example: verified templates and newly created bioinformatics tools are added to libraries that can affect later biomedical tasks [444]. Other systems update research artifacts only within the current project, even when the project contains many rounds of hypothesis generation, experiment design, or writing. The important organizing distinction is the final evidence source. Executable code and numerical checks can audit scientific-computing steps, formal proof tools can check derivations under stated assumptions, and physical experiments can test selected empirical claims. Model reviewers are useful for triage and diagnosis, but they do not independently establish novelty, statement faithfulness, or scientific value. This direction therefore needs layered evidence: task-local execution for correctness, protected held-out problems for transfer, and domain-expert or experimental review for claims that exceed what the executable checks measure.

Embodied and high-stakes systems. Robotics, healthcare, finance, education, power systems, and law provide consequential applications, but their strongest outcomes are slower, costlier, and harder to repeat. ENPIRE gives coding agents access to policy training and physical robot rollouts, creating stronger evidence than simulation alone while relying on pre-engineered resets, validators, and safety controls [546]. Medical consultation agents retain experience across simulated cases, but benchmark diagnoses and simulated patients do not establish clinical benefit [299]. FactorMiner retains financial skills and successful or forbidden patterns, yet its evidence remains historical backtesting rather than independently monitored trading [547]. Pedagogical agents evolve teacher strategies against simulated student outcomes, while human ratings mainly assess lesson quality rather than later student learning [548]. Power-flow and legal-retrieval agents likewise use simulation or benchmark relevance as development feedback [352, 366]. Physical measurements, expert review, or historical test sets are not external audits merely because they are difficult for a model to edit. Externality also depends on who selects the cases, how often the system can query them, and who controls the decision to activate an update.

Moving toward deployment in stages. Across these applications, deployment should be gradual: wider use should follow only as the evidence becomes stronger. Initial evaluation should use cases kept separate from development. The system can then be observed in real settings while its outputs are prevented from affecting users or operations, followed by limited use under explicit safeguards. Broader deployment should wait until independent evidence shows that the system meets the declared external target under matched evaluation and resource conditions, and until an authorized process judges the remaining limitations and risks acceptable for the people and settings affected.

Every retained L1–L4 update should have a traceable record linking the deployment decision to what changed, the external target and evaluation horizon, the development evidence considered, the external evidence source and acceptance policy, and the evaluation and resource conditions used for comparison. The record should also identify who may authorize promotion, deployment, suspension, and rollback. Evaluation must continue after release because changes in the system or its operating environment can make earlier evidence outdated. Later stages therefore require ongoing monitoring, incident reporting, and tests for rare but severe failures, together with renewed evaluation after material changes to the model, scaffold, improver, criterion, tools, sensors, or operating environment. The rigor of these safeguards should reflect both the scale of deployment and the severity of possible harm. Throughout this process, the evidence source and acceptance policy should remain outside the relevant update boundary. When evidence is incomplete, delayed, or contested, wider deployment depends on governance as well as technical evaluation: who may authorize it, and how can oversight keep pace with continued updates?

9.4 Governance: Goal Preservation and Scalable Oversight

The staged path above turns governance into the next bottleneck. Self-evolution makes that bottleneck a moving problem. Sustained interaction with AI may change user and overseer judgments, motivating a coupled human–AI view of long-term oversight [549]. Audit feedback can become training data, criterion changes can alter what counts as success, and human review may not keep pace with frequent updates. This section reviews current safeguards and asks how oversight can remain effective as the system changes.

Keeping audit feedback from becoming training data. The empirical acceptance policies in §8.3 depend on data sources and collection processes that deployment can change. Adaptive item generation can restore difficulty while changing the tested distribution. Behavioral contamination diagnostics can flag suspicious familiarity without providing definitive membership labels [162, 541]. Fresh data, adversarial probes, and environments outside the system's control support calibration only while repeated access does not turn them into development data. One possible protocol would version data provenance, separate development and audit budgets, log access, and recalibrate after distribution shift or repeated selection. The next question is how to keep using feedback without letting the same update loop shape both the evidence and its interpretation.

Constraining goal and criterion changes. The safety evidence in §8.4 shows that self-evaluation bias and changing success standards can persist across updates. When the criterion changes persistently, improvement under the new criterion does not by itself establish improvement under the declared external target. Adaptive safety tests can uncover problems missed by a static policy suite. Educational rubrics can expand long-tail coverage, and behavioral certificates can expose regressions hidden by aggregate scores [151, 504, 540]. The same studies also document shared judge bias, self-preference, and incomplete agreement with humans. This is not only a statistical problem. Safety constraints, rights, and acceptable trade-offs may be contested or inappropriate to delegate. One direction is to distinguish fixed-semantics item generation from a persistent change to the judgment-task distribution or criterion meaning. The process could preserve protected constraints outside the relevant update boundary, restrict repeated access, and state who may authorize each kind of change. It could also compare judgments under the pre-change criterion before promotion. Whether such a process can handle genuine goal change remains open.

Keeping oversight ahead of updates. External audit does not become reliable merely because a human or institution performs it. Human oversight has limited throughput, delayed feedback, varied expertise, and disagreement about acceptable outcomes. Current studies often restrict human review to sampled judge checks, a few expert-authored cases, lecture-quality ratings, or manual code validation [151, 504, 542, 548]. As systems change more components more often, limited review capacity can reduce coverage or be replaced by unvalidated proxies. One approach worth testing is to use automated checks to screen lower-risk updates. Experts could focus on criterion changes, severe or novel failures, and disagreements among checks. Future work also needs to measure audit coverage, delay, disagreement, backlog, escalation quality, and reviewer independence. Most importantly, it should test whether overseers can halt or reverse an update sequence in time. A candidate oversight design would also preserve a reference that the evolving system cannot control or gradually learn to game.

These four areas point to one shared question: after the system changes, what trustworthy evidence supports an improvement claim under the declared external target? They define an open research agenda, not a claim that deeper self-evolution is necessarily less reliable. Reliable self-evolution remains unresolved because systems, environments, and institutions all change, while the evidence used to judge improvement must remain valid.

10 Conclusion

Self-evolving agents use information produced during execution to revise current outputs or modify retained components that shape later behavior, updating, and judgment. This survey studies these systems through two questions: what changes during self-evolution, and what evidence supports a claim that the change is an improvement under a declared external target? Sections 2–7 address the first question by classifying each transition according to its deepest active semantic modification. The resulting taxonomy ranges from task-local Output-Level Self-Evolution (L0) to retained Model-Level, Scaffold-Level, Improver-Level, and Criterion-Level Self-Evolution (L1–L4). Across these levels, the survey compares representative systems by the object changed, the persistence of that change, the evidence used to guide it, and the boundary cases that separate adjacent levels.

Section 8 addresses the second question by comparing the evidence, acceptance, and control requirements across levels. Self-confirmation, model collapse, scaffold overfitting, metric capture, and criterion drift illustrate distinct ways in which an update can influence the evidence used to judge it. Self-evolution depth changes the failure mechanism and evaluation horizon, but it does not determine reliability. Each improvement claim needs evidence that matches its scope, while the external target, evidence source, and acceptance policy must remain outside the relevant update boundary. Retained changes also need protected lineage and independent recovery authority. The reliability ladder summarizes this matched-audit principle: no update should control the only evidence used for its own acceptance.

Section 9 turns these requirements into four open questions: whether capabilities improve over time, whether evaluation can follow the update history, when an update is ready for deployment, and how goals and oversight can remain effective as the system changes. Existing studies report useful retained gains as well as forgetting, narrow specialization, and weak transfer, so sustained capability growth remains an open empirical question. Likewise, evidence that an agent is self-evolving does not by itself establish improvement or imply accelerating gains. Progress therefore depends not only on what an agent can change, but also on whether the evidence used to evaluate that change remains informative and outside the relevant update boundary.

References

[1] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 235, pp. 62138–62160, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

[2] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

[3] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/b5c8c1c117618267944b2617add0a766-Abstract-Conference.html.

[4] Lilian Weng. Harness engineering for self-improvement, 2026. URL https://lilianweng.github.io/posts/2026-07-04-harness/.

[5] David Silver and Richard S. Sutton. Welcome to the era of experience, 2025. URL https://storage.googleapis.com/deepmind-media/Era-of-Experience/The%20Era%20of%20Experience%20Paper.pdf.

[6] Chengpeng Li, Mingfeng Xue, Zhenru Zhang, Jiaxi Yang, Beichen Zhang, Bowen Yu, Binyuan Hui, Junyang Lin, et al. START: Self-taught reasoner with tools. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 13512–13553, 2025. URL https://doi.org/10.18653/v1/2025.emnlp-main.683.

[7] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

[8] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=KuPixIqPiq.

[9] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2:801–824, 2025. URL https://dl.acm.org/doi/10.1145/3715754.

[10] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 2025. URL https://www.nature.com/articles/s41586-025-08661-4.

[11] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, et al. Promptagent: Strategic planning with language models enables expert-level prompt

optimization. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024. URL https://openreview.net/forum?id=22pyNMuIoa.

[12] Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. Webevolver: Enhancing web agent self-improvement with co-evolving world model. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 8959–8975, 2025. URL https://doi.org/10.18653/v1/2025.emnlp-main.454.

[13] Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, et al. SkillWeaver: Web agents can self-improve by discovering and honing skills, 2025. URL https://arxiv.org/abs/2504.07079.

[14] Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Godel machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations (ICLR)*, March 2026. URL https://openreview.net/forum?id=pUpzQZTvGY.

[15] Alex Iacob, Andrej Jovanović, William F. Shen, Daniel Burkhardt, Meghdad Kurmanji, Nurbek Tastan, Lorenzo Sani, Niccolò Alberto Elia Venanzi, et al. The Red Queen Gödel machine: Co-evolving agents and their evaluators, 2026. URL https://arxiv.org/abs/2606.26294.

[16] Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, et al. A survey on self-evolution of large language models, 2024. URL http://arxiv.org/abs/2404.14387.

[17] Zhe Ren, Yimeng Chen, Dandan Guo, Guowei Rong, Tonghui Li, R. B. Xiong, Qingfeng Lan, Wenyi Wang, et al. Self-improvements in modern agentic systems: a survey, 2026. URL https://arxiv.org/abs/2607.13104.

[18] Che Jiang, Jincheng Zhong, Yu Fu, Kai Tian, Junlin Yang, Kaikai Zhao, Yuchong Wang, Tianwei Luo, et al. Self-improving agents in the era of experience: A survey of self- to meta-evolution. *OpenReview Archive*, 2026. URL https://openreview.net/forum?id=IUltZSgLMm.

[19] Xun Liang, Shichao Song, Zifan Zheng, Hanyu Wang, Qingchen Yu, Xunkai Li, Rong-Hua Li, Yi Wang, et al. Internal consistency and self-feedback in large language models: A survey, 2024. URL http://arxiv.org/abs/2407.14507.

[20] Junhao Zheng, Shengjie Qiu, Chengming Shi, and Qianli Ma. Towards lifelong learning of large language models: A survey. *ACM Computing Surveys*, 57:1–35, 2025. URL https://doi.org/10.1145/3716629.

[21] Guanting Dong, Xiaoshuai Song, Yuyang Hu, Jiajie Jin, Chenghao Zhang, Yifei Chen, Xiaoxi Li, Huaying Yuan, et al. Towards long-horizon agents: a survey, 2026. URL https://openreview.net/forum?id=HyhfhlbWGh.

[22] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, et al. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43:1–47, 2025. URL https://doi.org/10.1145/3748302.

[23] Jiachun Li, Zhuoran Jin, Tianyi Men, Yupu Hao, Kejian Zhu, Lingshuai Wang, Dongqi Huang, Longxiang Wang, et al. Agentic environment engineering for large language models: A survey of

environment modeling, synthesis, evaluation, and application, 2026. URL http://arxiv.org/abs/2606.12191.

[24] Jinhu Qi, Muzhi Li, Jiahong Liu, Yuqin Shu, Dianzhi Yu, Shicheng Ma, Wenqian Cui, Yiyang Zhao, et al. Towards trustworthy agentic AI: a comprehensive survey of safety, robustness, privacy, and system security. *Academia AI and Applications*, 2(2), 2026. ISSN 3071-0286. URL https://www.academia.edu/166114173/Towards_trustworthy_agentic_AI_a_comprehensive_survey_of_safety_robustness_privacy_and_system_security.

[25] Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, et al. A comprehensive survey of self-evolving AI agents: A new paradigm bridging foundation models and lifelong agentic systems, 2025. URL http://arxiv.org/abs/2508.07407.

[26] Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, et al. 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*, 2026. URL https://openreview.net/forum?id=CTr3bovS5F.

[27] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

[28] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

[29] Yuxuan Cai, Yipeng Hao, Jie Zhou, Hang Yan, Zhikai Lei, Rui Zhen, Zhenhua Han, Yutao Yang, et al. Building self-evolving agents via experience-driven lifelong learning: A framework and benchmark, 2025. URL https://arxiv.org/abs/2508.19005.

[30] XYZ Agentic Team. AI4AI at scale: A full-pipeline system for enhancing LLM agentic capabilities. Technical report, XYZ AI Lab, July 2026. URL https://xyz-lab.ai/blogs/ai4ai-at-scale/assets/bounded-exploration-ai4ai-system-optimization.pdf.

[31] Zhishang Xiang, Chengyi Yang, Zerui Chen, Zhimin Wei, Yunbo Tang, Zongpei Teng, Zexi Peng, Zongxia Li, et al. A systematic survey of self-evolving agents: From model-centric to environment-driven co-evolution. *SSRN Electronic Journal*, 2026. ISSN 1556-5068. doi: 10.2139/ssrn.6626878. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6626878.

[32] Shijian Deng, Kai Wang, Tianyu Yang, Harsh Singh, and Yapeng Tian. Self-improvement in multimodal large language models: A survey. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, October 2025. URL https://aclanthology.org/2025.findings-emnlp.105/.

[33] Xiangjue Dong, Maria Teleki, and James Caverlee. A survey on LLM inference-time self-improvement, 2024. URL http://arxiv.org/abs/2412.14352.

[34] Shihao Qi, Jie Ma, Rui Xing, Wei Guo, Xiao Huang, Zhitao Gao, Jianhao Deng, Jun Liu, et al. Beyond individual intelligence: Surveying collaboration, failure attribution, and self-evolution in LLM-based multi-agent systems, 2026. URL http://arxiv.org/abs/2605.14892.

[35] Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, et al. The landscape of agentic reinforcement learning for LLMs: A survey. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=RY19y2RI1O.

[36] Pengfei Du. Memory for autonomous LLM agents:mechanisms, evaluation, and emerging frontiers, 2026. URL http://arxiv.org/abs/2603.07670.

[37] Peiying Yu, Guoxin Chen, and Jingjing Wang. Table-critic: A multi-agent framework for collaborative criticism and refinement in table reasoning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, May 2025. URL https://doi.org/10.18653/v1/2025.acl-long.853.

[38] Tal Ridnik, Dedy Kredo, and Itamar Friedman. Code generation with AlphaCodium: From prompt engineering to flow engineering, 2024. URL http://arxiv.org/abs/2401.08500.

[39] Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. In Satish Chandra, Kelly Blincoe, and Paolo Tonella (eds.), *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pp. 1229–1241, 2023. URL https://doi.org/10.1145/3611643.3616243.

[40] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, et al. Inner monologue: Embodied reasoning through planning with language models. In Karen Liu, Dana Kulic, and Jeffrey Ichnowski (eds.), *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782, 2023. URL https://proceedings.mlr.press/v205/huang23c.html.

[41] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with llms enables open-world multi-task agents. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/6b8dfb8c0c12e6fafc6c256cb08a5ca7-Abstract-Conference.html.

[42] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

[43] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

[44] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, October 2023. URL https://doi.org/10.18653/v1/2023.emnlp-main.507.

[45] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, March 2024. URL https://doi.org/10.1609/aaai.v38i16.29720.

[46] Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solver. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025. URL https://openreview.net/forum?id=6aHUmotXaw.

[47] Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=ktrw68Cmu9c.

[48] Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-tau Yih, Sida I. Wang, and Xi Victoria Lin. LEVER: Learning to verify language-to-code generation with execution. In *Proceedings of the International Conference on Machine Learning (ICML)*, September 2023. URL https://proceedings.mlr.press/v202/ni23b.html.

[49] Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3563–3578, 2024. URL https://aclanthology.org/2024.findings-acl.212/.

[50] Yuxuan Wan, Tianqing Fang, Zaitang Li, Yintong Huo, Wenxuan Wang, Haitao Mi, Dong Yu, and Michael R. Lyu. Inference-time scaling of verification: Self-evolving deep research agents via test-time rubric-guided verification. In *Findings of the Association for Computational Linguistics: ACL 2026*, April 2026. URL https://aclanthology.org/2026.findings-acl.1243/.

[51] Zhangyi Hu, Chenhui Liu, Tian Huang, Jindong Li, Yang Yang, Jiemin Wu, Zining Zhong, Menglin Yang, et al. CoSPlay: Cooperative self-play at test-time with self-generated code and unit test, 2026. URL http://arxiv.org/abs/2605.23491.

[52] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

[53] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023. URL https://doi.org/10.18653/v1/2023.acl-long.754.

[54] Zinan Tang, Xin Gao, Qizhi Pei, Zhuoshi Pan, Mengzhang Cai, Jiang Wu, Conghui He, and Lijun Wu. Middo: Model-informed dynamic data optimization for enhanced LLM fine-tuning via closed-loop learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, July 2025. URL https://doi.org/10.18653/v1/2025.emnlp-main.350.

[55] Zhe Xu, Daoyuan Chen, Zhenqing Ling, Yaliang Li, and Ying Shen. MindGYM: What matters in question synthesis for thinking-centric fine-tuning? In *Advances in Neural Information Processing Systems (NeurIPS)*, October 2025. URL http://papers.nips.

cc/paper_files/paper/2025/hash/f8fb39523b02c685c5cdd299e8753a05-Abstract-Datasets_and_Benchmarks_Track.html.

[56] Guanzheng Chen, Xin Li, Michael Qizhe Shieh, and Lidong Bing. LongPO: Long context self-evolution of large language models through short-to-long preference optimization. In *International Conference on Learning Representations (ICLR)*, March 2025. URL https://openreview.net/forum?id=qTrEq31Shm.

[57] Chenghua Huang, Zhizhen Fan, Lu Wang, Fangkai Yang, Pu Zhao, Zeqi Lin, Qingwei Lin, Dongmei Zhang, et al. Self-evolved reward learning for LLMs. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=Zonhl0c9I0.

[58] Weixuan Ou, Yanzhao Zheng, Shuoshuo Sun, Wei Zhang, Baohua Dong, Hangcheng Zhu, Ruohui Huang, Gang Yu, et al. SERL: Self-examining reinforcement learning on open-domain. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. URL https://doi.org/10.1609/aaai.v40i38.40539.

[59] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, et al. Constitutional AI: Harmlessness from AI feedback, 2022. URL https://arxiv.org/abs/2212.08073.

[60] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, et al. RLAIF vs. RLHF: Scaling reinforcement learning from human feedback with AI feedback. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/lee24t.html.

[61] Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, et al. TTRL: Test-time reinforcement learning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, volume 38, pp. 131459–131483, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/be690ea16f005c174f6c4102a5970e67-Abstract-Conference.html.

[62] Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/chen24j.html.

[63] Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, Nan Du, and Xiaolong Li. Self-playing adversarial language game enhances LLM reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/e4be7e9867ef163563f4a5e90cec478f-Abstract-Conference.html.

[64] Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, et al. Absolute zero: Reinforced self-play reasoning with zero data. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30*

- *December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/9837dc00ff67d176373268ed48042d49-Abstract-Conference.html.

[65] Chengsong Huang, Haolin Liu, Tong Zheng, Runpeng Dai, Langlin Huang, Jinyuan Li, Zongxia Li, Zhepei Wei, et al. G-Zero: Self-play for open-ended generation from zero data, 2026. URL https://arxiv.org/abs/2605.09959.

[66] Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, et al. R-Zero: Self-evolving reasoning LLM from zero data. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=96apU6YzSO.

[67] Wenqi Chen, Ziyan Zhang, Bin Wang, Lin Liu, Hengheng Zhang, and Zhengsu Chen. Learn from your mistakes: Tree-like self-play for secure code LLMs. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/61209.

[68] Yicheng He, Chengsong Huang, Zongxia Li, Jiaxin Huang, and Yonghui Yang. VisPlay: Self-evolving vision-language models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/He_VisPlay_Self-Evolving_Vision-Language_Models_CVPR_2026_paper.html.

[69] Bo Liu, Simon Yu, Zichen Liu, Leon Guertler, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, et al. SPIRAL: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=7Yayy5fNLg.

[70] Qinsi Wang, Bo Liu, Tianyi Zhou, Jing Shi, Yueqian Lin, Yiran Chen, Hai Helen Li, Kun Wan, et al. Vision-zero: Scalable VLM self-evolution via multi-agent self-play. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=s00SNXREV6.

[71] Ziyi Yang, Weizhou Shen, Chenliang Li, Ruijun Chen, Fanqi Wan, Ming Yan, Xiaojun Quan, and Fei Huang. SPELL: Self-play reinforcement learning for evolving long-context language models. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=83F6YF4Hz6.

[72] Wei Zou, Sen Yang, Yu Bao, Shujian Huang, Jiajun Chen, and Shanbo Cheng. Trans-zero: Self-play incentivizes large language models for multilingual translation without parallel data. In *Findings of the Association for Computational Linguistics: ACL 2025*, May 2025. URL https://aclanthology.org/2025.findings-acl.637/.

[73] Xiwen Chen, Wenhui Zhu, Jingjing Wang, Peijie Qiu, Zhipeng Wang, Huayu Li, ZhengXiao He, Xuanzhao Dong, et al. S-SPPO: Semantic-calibrated self-play preference optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/61018.

[74] Guibin Zhang, Xun Xu, Yanwei Yue, Zikun Su, Wangchunshu Zhou, Xiaobin Hu, and Shuicheng Yan. OPD-Evolver: Cultivating holistic agent evolver via on-policy distillation, 2026. URL https://arxiv.org/abs/2606.17628.

[75] Yudi Zhang, Meng Fang, Zhenfang Chen, and Mykola Pechenizkiy. Self-evolving LLM agents with in-distribution optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/62030.

[76] Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, et al. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations (ICLR)*, January 2025. URL https://openreview.net/forum?id=oVKEAFjEqv.

[77] Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. SEAgent: Self-evolving computer use agent with autonomous learning from experience. In *Proceedings of the International Conference on Machine Learning (ICML)*, August 2026. URL https://icml.cc/virtual/2026/poster/65711.

[78] Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. Co-evolving LLM coder and unit tester via reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, September 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/d38653cdaa8e992549e1e9e1621610d7-Abstract-Conference.html.

[79] Kaiwen Zha, Zhengqi Gao, Maohao Shen, Zhang-Wei Hong, Duane S. Boning, and Dina Katabi. RL tango: Reinforcing generator and verifier together for language reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, October 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/ad06e23fe0c39b9de6e0cefe3b701f45-Abstract-Conference.html.

[80] Jacob Eisenstein, Reza Aghajani, Adam Fisch, Dheeru Dua, Fantine Huot, Mirella Lapata, Vicky Zayats, and Jonathan Berant. Don't lie to your friends: Learning what you know from collaborative self-play. In *Conference on Language Modeling (COLM)*, August 2025. URL https://openreview.net/forum?id=2vDJiGUfhV.

[81] Hongrui Jia, Chaoya Jiang, Yongrui Heng, Shikun Zhang, and Wei Ye. From blind spots to gains: Diagnostic-driven iterative training for large multimodal models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/60731.

[82] Arthur Câmara, Vincent Slot, and Jakub Zavrel. Self-optimizing multi-agent systems for deep research. In *Workshop on Conversational Search for Complex Information Needs at ECIR 2026*, 2026. URL https://arxiv.org/abs/2604.02988.

[83] Rongsheng Hu, Runwei Guan, Yicheng Di, Jiayu Bao, and Yuan Liu. AutoVQA-G: Self-improving agentic framework for automated visual question answering and grounding annotation. In *Proceedings of the 2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2026)*, pp. 12312–12316, 2026. URL https://doi.org/10.1109/ICASSP55912.2026.11462754.

[84] Weixian Xu, Shilong Liu, and Mengdi Wang. EEVEE: Towards test-time prompt learning in the real world for self-improving agents, 2026. URL https://arxiv.org/abs/2606.11182.

[85] Xiwen Chen, Wenhui Zhu, Songzhu Zheng, Kashif Rasul, Yueyue Deng, and Huayu Li. SHARP: A self-evolving human-auditable rubric policy for financial trading agents, 2026. URL https://arxiv.org/abs/2605.06822.

[86] Hanchen Li, Runyuan He, Qizheng Zhang, Changxiu Ji, Qiuyang Mang, Xiaokun Chen, Lakshya A Agrawal, Wei-Liang Liao, et al. Combee: Scaling prompt learning for self-improving language model agents, 2026. URL https://arxiv.org/abs/2604.04247.

[87] Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. Meta context engineering via agentic skill evolution. In *Proceedings of the International Conference on Machine Learning (ICML)*, February 2026. URL https://icml.cc/virtual/2026/poster/64296.

[88] Cunxi Yu and Haoxing Ren. Autonomous evolution of EDA tools: Multi-agent self-evolved ABC. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference (DAC 2026)*, 2026. URL https://63dac.conference-program.com/presentation/?id=RESEARCH1784&sess=sess168.

[89] Shangheng Du, Xiangchao Yan, Jinxin Shi, Zongsheng Cao, Shiyang Feng, Zichen Liang, Boyuan Sun, Tianshuo Peng, et al. MLEvolve: A self-evolving framework for automated machine learning algorithm discovery, 2026. URL https://arxiv.org/abs/2606.06473.

[90] Hassan Jalil Hadi, Rehana Yasmin, and Ali Shoker. GenTI: Benchmarking LLMs for autonomous IDPS rule generation for unseen attacks, 2026. URL https://arxiv.org/abs/2606.05844.

[91] Yongfeng Huang, Ruiying Chen, and James Cheng. SEMA-RAG: A self-evolving multi-agent retrieval-augmented generation framework for medical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.917/.

[92] Wentao Zhang, Liang Zeng, Yuzhen Xiao, Yongcong Li, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, et al. AgentOrchestra: Orchestrating multi-agent intelligence with the tool-environment-agent(TEA) protocol, 2025. URL https://arxiv.org/abs/2506.12508.

[93] Martin Legrand, Tao Jiang, Matthieu Feraud, Benjamin Navet, Yousouf Taghzouti, Fabien Gandon, Elise Dumont, and Louis-Félix Nothias. Mimosa framework: Toward evolving multi-agent systems for scientific research, 2026. URL https://arxiv.org/abs/2603.28986.

[94] Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. AgentSquare: Automatic LLM agent search in modular design space. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025. URL https://openreview.net/forum?id=mPdmDYIQ7f.

[95] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

[96] Zixuan Ke, Austin Xu, Yifei Ming, Xuan-Phi Nguyen, Ryan Chin, Caiming Xiong, and Shafiq Joty. MAS-ZERO: Designing multi-agent systems with zero supervision. In *NeurIPS 2025 Workshop on Scaling Environments for Agents (SEA)*, 2025. URL https://openreview.net/forum?id=j5VXzWyoyW.

[97] Congjia Tian, Yuhang Yao, and Jiaming Cui. QueenBee planner: Skill-evolving communication topologies for token-efficient LLM multi-agent systems, 2026. URL https://arxiv.org/abs/2606.27492.

[98] Junle Wang, Xingchuang Liao, and Wenjun Wu. TopoEvo: A topology-aware self-evolving multi-agent framework for root cause analysis in microservices, 2026. URL https://arxiv.org/abs/2605.15611.

[99] Jiatan Huang, Zheyuan Zhang, Kaiwen Shi, Yanfang Ye, and Chuxu Zhang. EvolveRouter: Co-evolving routing and prompt for multi-agent question answering, 2026. URL https://arxiv.org/abs/2604.05149.

[100] Prakhar Dixit and Tim Oates. ISM:self-improving strategy memory for continual mathematical reasoning. In *Proceedings of the 3rd AI for Math Workshop at ICML 2026*, 2026. URL https://openreview.net/forum?id=5JK3t0YI5Z.

[101] Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, et al. CoEvoSkills: Self-evolving agent skills via co-evolutionary verification, 2026. URL https://arxiv.org/abs/2604.01687.

[102] Yunhao Yang, Neel P. Bhatt, Kevin Wang, Samuel Tetteh, Zhangyang Wang, and Ufuk Topcu. VASO: Formally verifiable self-evolving skills for physical AI agents, 2026. URL https://arxiv.org/abs/2606.05395.

[103] Johannes Moll, Jean-Philippe Corbeil, Jiazhen Pan, Martin Hadamitzky, Daniel Rueckert, Lisa Adams, and Keno Bressem. GRASP: Gated regression-aware skill proposer for self-improving LLM agents, 2026. URL https://arxiv.org/abs/2605.29668.

[104] Hongyi Liu, Haoyan Yang, Tao Jiang, Bo Tang, Feiyu Xiong, Yuyu Luo, and Zhiyu Li. SkillsVote: Lifecycle governance of agent skills from collection, recommendation to evolution, 2026. URL https://arxiv.org/abs/2605.18401.

[105] Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, et al. SkillOpt: Executive strategy for self-evolving agent skills, 2026. URL http://arxiv.org/abs/2605.23904.

[106] Tong Bai, Zhenglin Wan, Pengfei Zhou, Xingrui Yu, Yang You, and Ivor W. Tsang. SkillDAG: Self-evolving typed skill graphs for LLM skill selection at scale, 2026. URL https://arxiv.org/abs/2606.03056.

[107] Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, et al. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution, 2025. URL https://arxiv.org/abs/2505.20286.

[108] Dingcheng Huang, Yuda Ding, Bingshuo Liu, Qingbin Liu, Xi Chen, Jiang Bian, Hongliang Sun, Zhiying Tu, et al. SkillWiki: A living knowledge infrastructure for agent skills, 2026. URL https://arxiv.org/abs/2606.16523.

[109] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=eC4ygDs02R.

[110] Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, et al. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=jL7fwchScm.

[111] Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, et al. MemRL: Self-evolving agents via runtime reinforcement learning on episodic memory, 2026. URL https://arxiv.org/abs/2601.03192.

[112] Zhanghao Hu, Qinglin Zhu, Runcong Zhao, Di Liang, Hanqi Yan, Yulan He, and Lin Gui. Beyond RAG for agent memory: Retrieval by decoupling and aggregation, 2026. URL https://arxiv.org/abs/2602.02007.

[113] Genglin Liu, Shijie Geng, Sha Li, Hejie Cui, Sarah Zhang, Xin Liu, and Tianyi Liu. WebCoach: Self-evolving web agents with cross-session memory guidance. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=FDrGfjwQM5.

[114] Ao Tian, Yunfeng Lu, Xinxin Fan, Changhao Wang, Lanzhi Zhou, Yeyao Zhang, and Yanfang Liu. RGMem: Renormalization group-inspired memory evolution for language agents. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/65195.

[115] Pratyay Banerjee, Masud Moshtaghi, Shivashankar Subramanian, Amita Misra, and Ankit Chadha. APEX-MEM: Agentic semi-structured memory with temporal reasoning for long-term conversational AI. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.749.

[116] Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.829/.

[117] Tobias Lindenbauer, Georg Groh, and Hinrich Schütze. From knowledge to noise: CTIM-rover and the pitfalls of episodic memory in software engineering agents. In *Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pp. 411–427, 2025. URL https://aclanthology.org/2025.realm-1.30/.

[118] Yee Hin Chong, Jiaming Wu, Youhui Zhang, and Peng Qu. Towards feedback-to-plan decisions for self-evolving LLM agents in CUDA kernel generation. In *Proceedings of the International Conference on Machine Learning (ICML)*, May 2026. URL https://icml.cc/virtual/2026/poster/61256.

[119] Ankit Agrawal, Jithin Garapati, and Bohan Zhang. AutonomyLens: A self-evolving simulation-based testing loop for autonomous systems. In *Companion Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering (FSE Companion 2026)*, 2026. URL https://doi.org/10.1145/3803437.3805540.

[120] Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. AutoHarness: improving LLM agents by automatically synthesizing a code harness. In *ICLR 2026 Workshop on AI with Recursive Self-Improvement*, 2026. URL https://openreview.net/forum?id=g9rEYVNn5T.

[121] Wenbo Pan, Shujie Liu, Xiangyang Zhou, Shiwei Zhang, Wanlu Shi, Mirror Xu, and Xiaohua Jia. M*: Every Task Deserves Its Own Memory Harness, 2026. URL https://arxiv.org/abs/2604.11811.

[122] Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026. URL https://arxiv.org/abs/2603.28052.

[123] Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves, 2026. URL https://arxiv.org/abs/2606.09498.

[124] Ningyan Zhu, Huacan Wang, Jie Zhou, Feiyu Chen, Shuo Zhang, Ge Chen, Chen Liu, Jiarou Wu, et al. SemaClaw: a step towards general-purpose personal AI agents through harness engineering, 2026. URL https://arxiv.org/abs/2604.11548.

[125] Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. HarnessForge: Joint harness and policy evolution for adaptive agent systems, 2026. URL https://arxiv.org/abs/2606.01779.

[126] Yue Huang, Wenjie Wang, Han Bao, Yuchen Ma, Xiaonan Luo, Yi Nian, Haomin Zhuang, Zheyuan Liu, et al. MemoHarness: Agent harnesses that learn from experience, 2026. URL https://arxiv.org/abs/2607.14159.

[127] Haebin Seong, Li Yin, Haoran Zhang, and Zhan Shi. The last harness you'll ever build, 2026. URL https://arxiv.org/abs/2604.21003.

[128] Ruhan Wang, Yucheng Shi, Zongxia Li, Zhongzhi Li, Yue Yu, Junyao Yang, Kishan Panaganti, Haitao Mi, et al. Harness handbook: Making evolving agent harnesses readable,navigable, and editable, 2026. URL https://arxiv.org/abs/2607.13285.

[129] Seth Karten, Joel Zhang, Tersoo Upaa, Jr., Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents, 2026. URL https://arxiv.org/abs/2605.09998.

[130] Jürgen Schmidhuber. Gödel machines: Self-Referential Universal Problem Solvers Making Provably Optimal Self-Improvements, 2003. URL https://arxiv.org/abs/cs/0309048.

[131] Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursive self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, May 2025. URL https://aclanthology.org/2025.acl-long.1354/.

[132] Maxime Robeyns, Martin Szummer, and Laurence Aitchison. A self-improving coding agent. In *ICLR 2025 Workshop on Self-Improving Foundation Models Without Human Supervision*, 2025. URL https://openreview.net/forum?id=rShJCyLsOr.

[133] Yoichi Ishibashi, Taro Yano, and Masafumi Oyamada. Can large language models invent algorithms to improve themselves? In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 10332–10363, 2025. URL https://doi.org/10.18653/v1/2025.naacl-long.519.

[134] Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-Gödel machine: Human-level coding agent development by an approximation of the optimal self-improving machine. In *International Conference on Learning Representations (ICLR)*, October 2026. URL https://openreview.net/forum?id=T0EiEuhOOL.

[135] Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. In *Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=46Zgqo4QIU.

[136] Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544, 2024. URL https://proceedings.mlr.press/v235/fernando24a.html.

[137] Aditya Kakade, Vivek Srivastava, and Shirish Karande. Polaris: A Gödel agent framework for small language models through experience-abstracted policy repair. In *Findings of the Association for Computational Linguistics: ACL 2026*, June 2026. URL https://aclanthology.org/2026.findings-acl.1969/.

[138] Shan He, Runze Wang, Zhuoyun Du, Huiyu Bai, Zouying Cao, Yu Cheng, and Bo Zheng. Learning to evolve: A self-improving framework for multi-agent systems via textual parameter graph optimization, 2026. URL https://arxiv.org/abs/2604.20714.

[139] Zhan Shi, Bing He, Yisi Sang, Hanqing Lu, and Benoit Dumoulin. A-Evolve-training: Autonomous post-training of a 30B model, 2026. URL http://arxiv.org/abs/2606.20657.

[140] Guhong Chen, Yingcheng Shi, Yongbin Li, Binhua Li, Xander Xu, Hu Wei, Shiwen Ni, Min Yang, et al. EvoTrainer: Co-evolving LLM policies and training harnesses for autonomous agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2606.03108.

[141] Yuxuan Liu, Tianchi Yang, Shaohan Huang, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, et al. Calibrating LLM-based evaluator. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING)*, 2024. URL https://aclanthology.org/2024.lrec-main.237/.

[142] Clemencia Siro, Pourya Aliannejadi, and Mohammad Aliannejadi. Learning to judge: LLMs designing and applying evaluation rubrics. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6371–6389, Rabat, Morocco, March 2026. URL https://aclanthology.org/2026.findings-eacl.335/.

[143] Junyi Zhou, Qiyuan Zhang, Yufei Wang, Fuyuan Lyu, Yidong Ming, Can Xu, Qingfeng Sun, Kai Zheng, et al. RubricBench: Aligning model-generated rubrics with human standards. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 31179–31200, San Diego, California, United States, July 2026. URL https://doi.org/10.18653/v1/2026.acl-long.1439.

[144] Xin Guan, Xiaomeng Hu, Shen Huang, Zhenyi Wang, Bo Zhang, Zijian Li, Pengjun Xie, Bo Liu, et al. EvoRubric: Self-evolving rubric-driven RL for open-ended generation, 2026. URL https://arxiv.org/abs/2605.29847.

[145] Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, et al. DR Tulu: Reinforcement learning with evolving rubrics for deep research. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=97NEP1pyS3.

[146] Jiayin Zhu, Kelong Mao, Yudong Guo, Dengbo He, Sulong Xu, Simiu Gu, and Yutao Yue. SkillCoach: Self-evolving rubrics for evaluating and enhancing agentic skill-use, 2026. URL http://arxiv.org/abs/2607.01874.

[147] Xing Zhang, Guanghui Wang, Yanwei Cui, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Who grades the grader? co-evolving evaluation metrics and skills for self-improving LLM agents, 2026. URL https://arxiv.org/abs/2607.12790.

[148] Jinbiao Wei, Qianran Ma, Yilun Zhao, Xiao Zhou, Kangqi Ni, Guo Gan, and Arman Cohan. OpenComputer: Verifiable software worlds for computer-use agents, 2026. URL https://arxiv.org/abs/2605.19769.

[149] Aojie Yuan, Yi Nian, Haiyue Zhang, Zijian Su, and Yue Zhao. SEVA: Self-evolving verification agent with process reward for fact attribution. In *Proceedings of the ICML 2026 Workshop on Trustworthy AI for Good (AI4GOOD)*, 2026. URL https://openreview.net/forum?id=Uv31y1YRpS.

[150] Andrew Dai, Boris Meinardus, Ciaran Regan, Yingtao Tian, and Yujin Tang. Discovering novel LLM experts via task-capability coevolution. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=efNINVs2So.

[151] Yixu Wang, Xin Wang, Yang Yao, Xinyuan Li, Xibang Yang, Yan Teng, Xingjun Ma, and Yingchun Wang. AgenticEval: Toward agentic and self-evolving safety evaluation of large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.727/.

[152] Xiang Lisa Li, Farzaan Kaiyom, Evan Zheran Liu, Yifan Mai, Percy Liang, and Tatsunori Hashimoto. AutoBencher: Towards declarative benchmark construction. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=ymt4crbbXh.

[153] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of machine learning research*, pp. 17473–17498, 2022. URL https://proceedings.mlr.press/v162/parker-holder22a.html.

[154] Mateus Machado and Hanseclever Bassani. DyLam: A dynamic reward weighting framework for reinforcement learning algorithms. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 2651–2653, 2025. URL https://dl.acm.org/doi/10.5555/3709347.3743967.

[155] Yang Zhao, Hepeng Wang, Xiao Ding, Yangou Ouyang, Bibo Cai, Kai Xiong, Jinglong Gao, Zhouhao Sun, et al. MAESTRO: Meta-learning adaptive estimation of scalarization trade-offs for reward optimization. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.1019.

[156] Yining Lu, Zilong Wang, Shiyang Li, Xin Liu, Changlong Yu, Qingyu Yin, Zhan Shi, Zixuan Zhang, et al. Learning to optimize multi-objective alignment through dynamic reward weighting. *Transactions of the Association for Computational Linguistics*, 2026. URL https://yining610.github.io/dynamic-reward-weighting-webpage/.

[157] Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2Reward: Reward shaping with language models for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=tUM39YTRxH.

[158] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, et al. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=IEduRUO55F.

[159] Shengjie Sun, Runze Liu, Jiafei Lyu, Jing-Wen Yang, Liangpeng Zhang, and Xiu Li. A large language model-driven reward design framework via dynamic feedback for reinforcement learning. *Knowledge-Based Systems*, 326:114065, 2025. URL https://doi.org/10.1016/j.knosys.2025.114065.

[160] Pengyi Li, Jianye Hao, Hongyao Tang, Yifu Yuan, Jinbin Qiao, Zibin Dong, and Yan Zheng. R*: Efficient reward design via reward structure evolution and parameter alignment optimization with large language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 34509–34527, 2025. URL https://proceedings.mlr.press/v267/li25v.html.

[161] Taylor Sorensen, Liwei Jiang, Jena D. Hwang, Sydney Levine, Valentina Pyatkin, Peter West, Nouha Dziri, Ximing Lu, et al. Value kaleidoscope: Engaging AI with pluralistic human values, rights, and duties. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19937–19947, 2024. URL https://doi.org/10.1609/aaai.v38i18.29970.

[162] Han Jiang, Xiaoyuan Yi, Zhihua Wei, Ziang Xiao, Shu Wang, and Xing Xie. Raising the bar: Investigating the values of large language models via generative evolving testing. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 27724–27771, 2025. URL https://proceedings.mlr.press/v267/jiang25l.html.

[163] Jing Yao, Shitong Duan, Xiaoyuan Yi, Dongkuan Xu, Peng Zhang, Tun Lu, Ning Gu, Zhicheng Dou, et al. AdAEM: An adaptively and automated extensible measurement of LLMs' value difference. In *International Conference on Learning Representations (ICLR)*, March 2026. URL https://openreview.net/forum?id=qNlTH4kYJZ.

[164] Nikhil Verma. Active context compression: Autonomous memory management in LLM agents, 2026. URL http://arxiv.org/abs/2601.07190.

[165] Jiawei Mao, Hardy Chen, Haoqin Tu, Yuhan Wang, Letian Zhang, Zeyu Zheng, Huaxiu Yao, Zirui Wang, et al. Kestrel: Grounding self-refinement for LVLM hallucination mitigation, 2026. URL http://arxiv.org/abs/2603.16664.

[166] Zhouzhou Shen, Xueyu Hu, Xiyun Li, Tianqing Fang, Juncheng Li, and Shengyu Zhang. World-model-augmented web agents with action correction, 2026. URL https://arxiv.org/abs/2602.15384.

[167] Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. CodeMonkeys: Scaling test-time compute for software engineering, 2025. URL http://arxiv.org/abs/2501.14723.

[168] Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Y. Zhao, Ni Lao, et al. RARR: Researching and revising what language models say, using language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, May 2023. URL https://doi.org/10.18653/v1/2023.acl-long.910.

[169] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

[170] Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=y0GJXRungR.

[171] Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. AgentCoder: Multi-agent-based code generation with iterative testing and optimisation, 2024. URL http://arxiv.org/abs/2312.13010.

[172] Shuimu Chen, Yuteng Chen, Yuanshen Guan, Zebang Cheng, Zeyu Zhang, Shengqian Qin, Bin Xia, Jiaran Li, et al. Reflect-R1: Evidence-driven reflection for self-correction in long video understanding. In *Proceedings of the European Conference on Computer Vision (ECCV 2026)*, 2026. URL https://github.com/ShuimuChen-hyq/Reflect-R1.

[173] Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. Self-evaluation guided beam search for reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, October 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/81fde95c4dc79188a69ce5b24d63010b-Abstract-Conference.html.

[174] Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, et al. Is your LLM secretly a world model of the internet? model-based planning for web agents. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=c6l7yA0HSq.

[175] Shaheer U. Saeed, Yipei Wang, Veeru Kasivisvanathan, Brian R. Davidson, Matthew J. Clarkson, Yipeng Hu, and Daniel C. Alexander. Reasoning in machine vision by learning fast and slow thinking. *Nature Communications*, 2026. ISSN 2041-1723. URL https://www.nature.com/articles/s41467-026-74579-8.

[176] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

[177] Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, et al. S*: Test time scaling for code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025. URL https://aclanthology.org/2025.findings-emnlp.865/.

[178] Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, October 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/cde328b7bf6358f5ebb91fe9c539745e-Abstract-Conference.html.

[179] Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing LLM reasoning. In Aarti Singh, Maryam Fazel, Daniel

Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*, 2025. URL https://proceedings.mlr.press/v267/bi25a.html.

[180] Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. Natural language to code translation with execution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, November 2022. URL https://doi.org/10.18653/v1/2022.emnlp-main.231.

[181] Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, et al. Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024. URL https://openreview.net/forum?id=c8McWs4Av0.

[182] Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. Large language models can self-correct with key condition verification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, October 2024. URL https://doi.org/10.18653/v1/2024.emnlp-main.714.

[183] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, et al. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, October 2023. URL https://aclanthology.org/2023.findings-emnlp.167/.

[184] Jiaoyang Ruan, Xin Gao, Yinda Chen, Hengyu Zeng, Liang Du, Guanghao Li, Jie Fu, and Jian Pu. Reasoning on the manifold: Bidirectional consistency for self-verification in diffusion language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, May 2026. URL https://icml.cc/virtual/2026/poster/65552.

[185] Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. URL https://doi.org/10.18653/v1/2023.emnlp-main.67.

[186] Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, et al. Reinforced self-training (ReST) for language modeling, 2023. URL https://arxiv.org/abs/2308.08998.

[187] Junlong Jia, Ziyang Chen, Xing Wu, Chaochen Gao, TingHao Yu, Feng Zhang, and Songlin Hu. PolicyLong: Towards on-policy context extension, 2026. URL https://arxiv.org/abs/2604.07809.

[188] Nan Huang, Pengcheng Yu, Weijia Zeng, James M. Rehg, Angjoo Kanazawa, Haiwen Feng, and Qianqian Wang. Self-improving 4D perception via self-distillation, 2026. URL https://arxiv.org/abs/2604.08532.

[189] Jingwen Chen, Wenkai Yang, Shengda Fan, Wenbo Nie, Chenxing Sun, Shaodong Zheng, Yangen Hu, Lu Pan, et al. Rethinking continual experience internalization for self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2606.04703.

[190] Xinyi Wang, Rongze Chen, Ke Wang, Qiyuan Chen, Yanming Liu, Xiang Li, and Chunfu Jia. OASIF: An efficient obfuscation-aware self-improving framework for LLM-based assembly code instruction following and comprehension, 2026. URL https://arxiv.org/abs/2606.29155.

[191] Zichen Wen, Boxue Yang, Junlong Ke, Jiajie Huang, Chenfei Liao, Junxi Wang, Xuyang Liu, and Linfeng Zhang. EvoStreaming: Your offline video model is a natively streaming assistant, 2026. URL https://arxiv.org/abs/2605.10343.

[192] Shuyang Jiang, Yusheng Liao, Zhe Chen, Ya Zhang, Yanfeng Wang, and Yu Wang. Meds^3: Towards medical slow thinking with self-evolved soft dual-sided process supervision. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 31319–31327, 2026. URL https://doi.org/10.1609/aaai.v40i37.40395.

[193] Benjamin Schneider, Xavier Schneider, Victor Zhong, and Sun Sun. ASH: Agents that self-hone via embodied learning. In *Proceedings of the ICML 2026 Workshop on Scalable Learning and Optimization for Efficient Multimodal AI Agents (SCALE)*, 2026. URL https://openreview.net/forum?id=OR2EFiBnbA.

[194] Peng Xia, Jianwen Chen, Xinyu Yang, Haoqin Tu, Jiaqi Liu, Kaiwen Xiong, Siwei Han, Shi Qiu, et al. MetaClaw: Just talk – an agent that meta-learns and evolves in the wild, 2026. URL https://arxiv.org/abs/2603.17187.

[195] Xichen Zhang, Ziyi He, Yinghao Zhu, Sitong Wu, Shaozuo Yu, Meng Chu, Wenhu Zhang, Haoru Tan, et al. Searchgym: Bootstrapping real-world search agents via cost-effective and high-fidelity environment simulation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 18633–18665, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.848.

[196] Kairos Team, Fei Wang, Shan You, Qiming Zhang, Tao Huang, Zuoyi Fu, Zhisheng Zheng, Yunlong Xi, et al. Kairos: A regret-aware native world-action model stack for physical AI, 2026. URL https://arxiv.org/abs/2606.16533.

[197] Zhiyu Pan, Yizheng Wu, Jiashen Hua, Junyi Feng, Shaotian Yan, Bing Deng, Zhiguo Cao, and Jieping Ye. Through the lens of contrast: Self-improving visual reasoning in VLMs. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=ZymCPON45y.

[198] Wen Wen, Tianwu Zhi, Kanglong Fan, Yang Li, Xinge Peng, Yabin Zhang, Yiting Liao, Junlin Li, et al. Self-evolving vision-language models for image quality assessment via voting and ranking. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=INOi0YqI8p.

[199] Wenhao Li, Xiu Su, Dan Niu, Yichao Cao, Hongyan Xu, Zhe Qu, Lei Fan, Shan You, et al. Sentinel-VLA: A metacognitive VLA model with active status monitoring for dynamic reasoning and error recovery. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/61750.

[200] Chuan Xiao, Zhengbo Jiao, Shaobo Wang, Wei Wang, Bing Zhao, Hu Wei, Linfeng Zhang, and Lin Qu. Socratic-SWE: Self-evolving coding agents via trace-derived agent skills, 2026. URL https://arxiv.org/abs/2606.07412.

[201] Chunlei Shi, Junming Hou, Yi-Lin Wei, Jiong Wang, Yecheng Zhang, Yichao Dong, Wenqi Ren, and Dan Niu. LangRetrieval: Language-guided self-evolving satellite-to-radar retrieval via CSI-driven reward, 2026. URL https://arxiv.org/abs/2606.09486.

[202] Huanyu Liu, Jia Li, Yihong Dong, Chang Yu, Taozhi Chen, Lecheng Wang, Yongding Tao, Bin Gu, et al. EvoCoT: Overcoming the exploration bottleneck in reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.1031/.

[203] Chao Deng, Shaolei Zhang, Ju Fan, and Xiaoyong Du. DataEvolver: Automatic data preparation for large language models through multi-level self-evolving, 2026. URL https://arxiv.org/abs/2606.07001.

[204] Wei Liu, Junlong Li, Xiwen Zhang, Fan Zhou, Yu Cheng, and Junxian He. Diving into self-evolving training for multimodal reasoning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2025. URL https://proceedings.mlr.press/v267/liu25aj.html.

[205] Shuo Yang, Jinda Lu, Kexin Huang, Chiyu Ma, Shaohang Wei, Yuyang Liu, Guoyin Wang, Jingren Zhou, et al. One-way policy optimization for self-evolving LLMs. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/63408.

[206] Junming Liu, Yuqi Li, Yifei Sun, Maonan Wang, Piotr Koniusz, Yirong Chen, and Ding Wang. Self-evolving spatial reasoning in vision language models via geometric logic consistency, 2026. URL https://arxiv.org/abs/2605.18162.

[207] Shravan Venkatraman, Ritesh Thawkar, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Salman Khan, and Fahad Khan. Paying more attention to visual tokens in self-evolving large multimodal models. In *Proceedings of the European Conference on Computer Vision (ECCV 2026)*, 2026. URL https://mbzuai-oryx.github.io/VISE.

[208] Zhuo Wang, Zhuo Zhang, Yafu Li, Yu Cheng, Lizhen Qu, and Zenglin Xu. CoTEvol: Self-evolving chain-of-thoughts for data synthesis in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.1903/.

[209] Omar Shaikh, Michelle S. Lam, Joey Hejna, Yijia Shao, Hyundong Cho, Michael S. Bernstein, and Diyi Yang. Aligning language models with demonstrated feedback. In *International Conference on Learning Representations (ICLR)*, April 2025. URL https://openreview.net/forum?id=1qGkuxI9UX.

[210] Runlong Cao, Ying Zang, Chuanwei Zhou, Tianrun Chen, Tong Zhang, Zhen Cui, and Chunyan Xu. Learning to label: A reinforced self-evolving framework for semi-supervised referring expression segmentation. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/66442.

[211] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/yuan24d.html.

[212] Shimao Zhang, Xiao Liu, Xin Zhang, Junxiao Liu, Zheheng Luo, Shujian Huang, and Yeyun Gong. Process-based self-rewarding language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025. URL https://aclanthology.org/2025.findings-acl.930/.

[213] Zhengyang Tang, Ziniu Li, Zhenyang Xiao, Tian Ding, Ruoyu Sun, Benyou Wang, Dayiheng Liu, Fei Huang, et al. Self-evolving critique abilities in large language models. In *Conference on Language Modeling (COLM)*, August 2025. URL https://openreview.net/forum?id=TA6azZKWJq.

[214] Xiaoying Zhang, Zichen Liu, Yipeng Zhang, Xia Hu, and Wenqi Shao. RetroAgent: From solving to evolving via retrospective dual intrinsic feedback, 2026. URL https://arxiv.org/abs/2603.08561.

[215] Wen-Tse Chen, Jiayu Chen, Fahim Tajwar, Hao Zhu, Xintong Duan, Ruslan Salakhutdinov, and Jeff Schneider. Retrospective in-context learning for temporal credit assignment with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/680be66a383c5e7bfce021c47aedbb9e-Abstract-Conference.html.

[216] Qifan Zhang, Dongyang Ma, Tianqing Fang, Jia Li, Jing Tang, Nuo Chen, Haitao Mi, and Yan Wang. Training LLM agents for spontaneous, reward-free self-evolution via world knowledge exploration, 2026. URL https://arxiv.org/abs/2604.18131.

[217] Sikai Bai, Haoxi Li, Jie Zhang, Yongjiang Liu, and Song Guo. TTVS: Boosting self-exploring reinforcement learning via test-time variational synthesis, 2026. URL https://arxiv.org/abs/2604.08468.

[218] Aleksei Arzhantsev, Otmane Sakhi, and Flavian Vasile. RoiRL: Efficient, self-supervised reasoning with offline iterative reinforcement learning. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025. URL https://openreview.net/forum?id=PeJ1eGGygZ.

[219] Yanyu Chen, Jiyue Jiang, Dianzhi Yu, Zheng Wu, Jiahong Liu, Jiaming Han, Xiao Guo, Jinhu Qi, et al. LC-ERD: Mining latent logic for self-evolving reasoning via consistency-regulated reward decomposition. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2026)*, 2026. URL https://doi.org/10.1145/3770855.3817617.

[220] Zhiyin Yu, Bo Zhang, Qibin Hou, Zhonghai Wu, Xiao Luo, and Lei Bai. Easy samples are all you need: Self-evolving LLMs via data-efficient reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.773/.

[221] Yang Zhou, Can Jin, Zihan Dong, Zhepeng Wang, Yanting Yang, Shiyu Zhao, Lei Li, Runxue Bao, et al. DARE: Difficulty-adaptive reinforcement learning with co-evolved difficulty estimation, 2026. URL https://arxiv.org/abs/2605.09188.

[222] Chuanyang Jin, Jing Xu, Bo Liu, Leitian Tao, Olga Golovneva, Tianmin Shu, Wenting Zhao, Xian Li, et al. The era of real-world human interaction: RL from user conversations. In *ICLR 2026 Workshop on Navigating and Addressing Data Problems for Foundation Models*, 2026. URL https://openreview.net/forum?id=PRsVL3lqcm.

[223] Yixu Huang, Xinglei Yu, and Zhongyu Wei. ACE: Self-evolving LLM coding framework via adversarial unit test generation and preference optimization, 2026. URL https://arxiv.org/abs/2605.16299.

[224] Yijun Liang, Hengguang Zhou, Ming Li, Lichen Li, Cho-Jui Hsieh, and Tianyi Zhou. Self-evolving visual questioner, 2026. URL https://arxiv.org/abs/2606.13929.

[225] Xiuwei Chen, Wentao Hu, Hanhui Li, Yongxin Wang, Jun Zhou, Zisheng Chen, Meng Cao, Yihan Zeng, et al. SyncLoop: A multimodal dual-loop framework for self-improving mathematical reasoning. In *European Conference on Computer Vision (ECCV)*, 2026. URL https://arxiv.org/abs/2507.16518.

[226] Ritesh Thawkar, Shravan Venkatraman, Omkar Thawakar, Abdelrahman Shaker, Fahad Khan, Hisham Cholakkal, Salman Khan, and Rao Muhammad Anwer. Ask, solve, generate: Self-evolving unified multimodal understanding and generation via self-consistency rewards, 2026. URL https://arxiv.org/abs/2606.27376.

[227] Shiqi Huang, Ziyue Wang, Zhongrong Zuo, Han Qiu, Qi She, and Bihan Wen. EvoVid: Temporal-centric self-evolution for video large language models, 2026. URL https://arxiv.org/abs/2605.21931.

[228] Chaoran Xu, Yingmao Miao, Pengfei Zhang, Hao Dou, Lei Sun, and Xiangxiang Chu. RISE: Reliable improvement in self-evolving vision-language models, 2026. URL https://arxiv.org/abs/2605.20914.

[229] Dinging Li, Yingxiu Zhao, Xinrui Cheng, Kangheng Lin, Hongbo Peng, Hongxing Li, Zixuan Wang, Yuhong Dai, et al. SpatialEvo: Self-evolving spatial intelligence via deterministic geometric environments, 2026. URL https://arxiv.org/abs/2604.14144.

[230] Kyeongjin Ahn, Seungeon Lee, Krishna P. Gummadi, and Meeyoung Cha. GeoX: Mastering geospatial reasoning through self-play and verifiable rewards, 2026. URL https://arxiv.org/abs/2605.20006.

[231] Qingyu Ren, Qianyu He, Jiajie Zhu, Xingzhou Chen, Jingwen Chang, Zeye Sun, Han Xia, Fei Yu, et al. SEIF: Self-evolving reinforcement learning for instruction following, 2026. URL https://arxiv.org/abs/2605.07465.

[232] Huyu Wu, Jun Liu, Xiaochi Wei, Yan Gao, Yi Wu, and Yao Hu. Knowledge-graph paths as intermediate supervision for self-evolving search agents, 2026. URL https://arxiv.org/abs/2605.05702.

[233] Shaowei Zhang, Faqiang Qian, Yan Chen, Ziliang Wang, Kang An, Yong Dai, Mengya Gao, and Yichao Wu. SELF-EMO: Emotional self-evolution from recognition to consistent expression, 2026. URL https://arxiv.org/abs/2604.18003.

[234] Yuhao Zhang, Shaoming Duan, Jinhang Su, Chuanyi Liu, and Peiyi Han. SPFT-SQL: Enhancing large language model for text-to-SQL parsing by self-play fine-tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, October 2025. URL https://aclanthology.org/2025.findings-emnlp.59/.

[235] Pinzheng Wang, Juntao Li, Zecheng Tang, Haijia Gui, and Min zhang. Improving rationality in the reasoning process of language models through self-playing game. In *Proceedings of the International Conference on Machine Learning (ICML)*, July 2025. URL https://proceedings.mlr.press/v267/wang25bb.html.

[236] Qihao Liu, Luoxin Ye, Wufei Ma, Yu-Cheng Chou, and Alan Yuille. Generative adversarial reasoner: Enhancing LLM reasoning with adversarial reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ihucMuRXcY.

[237] Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David Zhang, Gabriel Synnaeve, Daniel Fried, Lingming Zhang, et al. Toward training superintelligent software agents through self-play SWE-RL. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/66747.

[238] Meghana Sunil, Manikandarajan Venmathimaran, and Muthu Subash Kavitha. iReasoner: Trajectory-aware intrinsic reasoning supervision for self-evolving large multimodal models. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.1468/.

[239] Wu Li, Yigeng Zhou, Zesheng Shi, Yequan Wang, Min Zhang, and Jing Li. Team-based self-play with dual adaptive weighting for fine-tuning LLMs. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.847.

[240] Yibo Wang, Hai-Long Sun, Guangda Huzhang, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. Triplets better than pairs: Towards stable and effective self-play fine-tuning for LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/3a797b10ff20562b1ecee0d4e914c1c7-Abstract-Conference.html.

[241] Minzheng Wang, Run Luo, Yanbo Wang, Zichen Liu, Yuqiao Tan, Tao Tan, Xu Nan, Lu Wang, et al. Breaking the impasse: Dual-scale evolutionary policy training for social language agents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.2096.

[242] Jacob Dineen, Aswin RRV, Zhikun Xu, and Ben Zhou. Vocabulary dropout for curriculum diversity in LLM co-evolution. In *Proceedings of the 3rd Conference on Language Modeling (COLM 2026)*, 2026. URL https://arxiv.org/abs/2604.03472.

[243] Yibo Wang, Guangda Huzhang, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. SPACE: Noise contrastive estimation stabilizes self-play fine-tuning for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/5692c7dbc4abcaa50f9ce609819212e5-Abstract-Conference.html.

[244] Jiashuo Wang, Jiawen Duan, Jian Wang, Kaitao Song, Chunpu Xu, Johnny K. W. Ho, Fenggang Yu, Wenjie Li, et al. Foresight optimization for strategic reasoning in large language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.1772.

[245] Wenjie Liao, Like Wu, Liangjie Zhao, Shihui Xu, and Shigeru Fujimura. IRIS: Interpolative Rényi iterative self-play for large language model fine-tuning, 2026. URL https://arxiv.org/abs/2604.20933.

[246] Swadesh Jana, Cansu Sancaktar, Tomáš Daniš, Georg Martius, Antonio Orvieto, and Pavel Kolev. GASP: Guided asymmetric self-play for coding LLMs. In *ICLR 2026 Workshop on Lifelong Agents*, 2026. URL https://openreview.net/forum?id=ChWC0E93lF.

[247] Yucong Huang, Xiucheng Li, Kaiqi Zhao, and Jing Li. Transitivity meets cyclicity: Explicit preference decomposition for dynamic large language model alignment. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/66071.

[248] Mingzhi Wang, Chengdong Ma, Qizhi Chen, Linjian Meng, Yang Han, Jiancong Xiao, Zhaowei Zhang, Jing Huo, et al. Magnetic preference optimization: Achieving last-iterate convergence for language model alignment. In *International Conference on Learning Representations (ICLR)*, April 2025. URL https://openreview.net/forum?id=PDnEDS244P.

[249] Wei Liu, Siya Qi, Yali Du, and Yulan He. Self-play only evolves when self-synthetic pipeline ensures learnable information gain. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/67044.

[250] Sophia Xiao Pu, Zhaotian Weng, Chengzhi Liu, Jayanth Srinivasa, Gaowen Liu, William Yang Wang, and Xin Eric Wang. Survive or collapse: The asymmetric roles of data gating and reward grounding in self-play RL, 2026. URL https://arxiv.org/abs/2605.22217.

[251] Matt Gorbett and Hossein Shirazi. Label-free reinforcement learning via cross-model entropy, 2026. URL https://arxiv.org/abs/2605.29009.

[252] Sixiang Chen, Zhaohu Xing, Tian Ye, Xinyu Geng, Yunlong Lin, Jianyu Lai, Xuanhua He, Fuxiang Zhai, et al. GenEvolve: Self-evolving image generation agents via tool-orchestrated visual experience distillation, 2026. URL https://arxiv.org/abs/2605.21605.

[253] Yaocheng Zhang, Yuanheng Zhu, Wenyue Chong, Songjun Tu, Qichao Zhang, Jiajun Chai, Xiaohan Wang, Wei Lin, et al. π-play: Multi-agent self-play via privileged self-distillation without external data, 2026. URL https://arxiv.org/abs/2604.14054.

[254] Yang Li, Erik Nijkamp, Semih Yavuz, and Shafiq Joty. Learning from language feedback via variational policy distillation, 2026. URL https://arxiv.org/abs/2605.15113.

[255] Xinyu Wang, Hanwei Wu, Jingwei Song, Shuyuan Zhang, Jiayi Zhang, Fanqi Kong, Tung Sum Thomas Kwok, Xiao-Wen Chang, et al. Co-evolution of policy and internal reward for language agents, 2026. URL https://arxiv.org/abs/2604.03098.

[256] Wenkai Wang, Xiyun Li, Hongcan Guo, Wenhao Yu, Tianqing Fang, Haitao Mi, Dong Yu, and Shengyu Zhang. Measure twice, click once: Co-evolving proposer and visual critic via reinforcement learning for GUI grounding. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 20964–20984, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.960.

[257] Fei Xu Yu, Zuyuan Zhang, Mahdi Imani, Nathaniel D. Bastian, and Tian Lan. Interactive critique-revision training for reliable structured LLM generation, 2026. URL https://arxiv.org/abs/2605.08327.

[258] Youwei Liu, Jian Wang, Hanlin Wang, and Wenjie Li. COMAP: Co-evolving world models and agent policies for LLM agents, 2026. URL https://arxiv.org/abs/2606.02372.

[259] Vaishali Senthil, Ashutosh Hathidara, and Sebastian Schreiber. CoHyDE: Iterative Co-Training of LLM Rewriter & Dense Encoder for Tool Retrieval, 2026. URL https://arxiv.org/abs/2605.29271.

[260] Hao Ma, Tianyi Hu, Zhiqiang Pu, Boyin Liu, Xiaolin Ai, Yanyan Liang, and Min Chen. Coevolving with the other you: Fine-tuning LLM with sequential cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, February 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/1c2b1c8f7d317719a9ce32dd7386ba35-Abstract-Conference.html.

[261] Naibin Gu, Chenxu Yang, Qingyi Si, Chuanyu Qin, Dingyu Yao, Peng Fu, Zheng Lin, Weiping Wang, et al. Co-evolving policy distillation, 2026. URL https://arxiv.org/abs/2604.27083.

[262] Woohyeon Byeon, Jiwon Jeon, Jeonghye Kim, and Youngchul Sung. Be my tutor: On-policy co-distillation for mutual LLM improvement via peer feedback, 2026. URL https://arxiv.org/abs/2606.14368.

[263] Changxin Ke, Rui Zhang, Shuo Wang, Li Ding, Guangli Li, Yuanbo Wen, Shuoming Zhang, Ruiyuan Xu, et al. QiMeng-MuPa: Mutual-supervised learning for sequential-to-parallel code translation. In *Advances in Neural Information Processing Systems (NeurIPS)*, October 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/d34f0c1158c1b8c4fcbe8fb2dbf0c552-Abstract-Conference.html.

[264] Lishui Fan, Mouxiang Chen, Tingwei Zhu, Kui Liu, Xin Xia, Shanping Li, and Zhongxin Liu. ZeroCoder: Can LLMs improve code generation without ground-truth supervision?, 2026. URL https://arxiv.org/abs/2604.07864.

[265] Zhiyuan Fan, Wenwei Jin, Feng Zhang, Bin Li, Yihong Dong, Yao Hu, and Jiawei Li. Evolving-RL: End-to-end optimization of experience-driven self-evolving capability within agents, 2026. URL https://arxiv.org/abs/2605.10663.

[266] Zongwei Wang, Min Gao, Hongzhi Yin, Junliang Yu, Tong Chen, Quoc Viet Hung Nguyen, Shazia Sadiq, and Tianrui Li. Self-distilled reinforcement learning for co-evolving agentic recommender systems, 2026. URL https://arxiv.org/abs/2604.10029.

[267] Yu Li, Rui Miao, Zhengling Qi, and Tian Lan. ARISE: Agent reasoning with intrinsic skill evolution in hierarchical reinforcement learning, 2026. URL https://arxiv.org/abs/2603.16060.

[268] Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, et al. In-the-flow agentic system optimization for effective planning and tool use. In *International Conference on Learning Representations (ICLR)*, October 2026. URL https://openreview.net/forum?id=Mf5AleTUVK.

[269] Ru Wang, Wei Huang, Qi Cao, Yusuke Iwasawa, Yutaka Matsuo, and Jiaxian Guo. Self-harmony: Learning to harmonize self-supervision and self-play in test-time reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=ZzG6oJ5ehI.

[270] Lin Qiu, Hanqing Zeng, Yao Liu, Bingjun Sun, Guangdeng Liao, and Ji Liu. DUEL: Adversarial self-play for multimodal reasoning, 2026. URL https://arxiv.org/abs/2605.24794.

[271] Chenglong Wang, Canjia Li, Xingzhao Zhu, Yifu Huo, Huiyu Wang, Weixiong Lin, Yun Yang, Qiaozhi He, et al. SERM: Self-evolving relevance model with agent-driven learning from massive query streams. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.823/.

[272] Panatchakorn Anantaprayoon, Nataliia Babina, Nima Asgharbeygi, and Jad Tarifi. Learning to negotiate: Multi-agent deliberation for collective value alignment in LLMs, 2026. URL https://arxiv.org/abs/2603.10476.

[273] Roger Creus Castanyer, Geoffrey Bradway, Lorenz Wolf, Maxwill Lin, Augustine N. Mavor-Parker, and Matthew James Sargent. PopuLoRA: Co-evolving LLM populations for reasoning self-play, 2026. URL https://arxiv.org/abs/2605.16727.

[274] Chenkai Pan, Xinglong Xu, Yuhang Xu, Yujun Wu, Siyuan Li, Jintao Chen, Conghui He, Jingxuan Wei, et al. Programming with data: Test-driven data engineering for self-improving LLMs from raw corpora, 2026. URL https://arxiv.org/abs/2604.24819.

[275] Elvis Dohmatob, Yunzhen Feng, Pu Yang, Francois Charton, and Julia Kempe. A tale of tails: Model collapse as a change of scaling laws. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2024. URL https://proceedings.mlr.press/v235/dohmatob24b.html.

[276] Yiwen Ding, Zhiheng Xi, Wei He, Zhuoyuan Li, Yitao Zhai, Xiaowei Shi, Xunliang Cai, Tao Gui, et al. Mitigating tail narrowing in LLM self-improvement via Socratic-guided sampling. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025. URL https://doi.org/10.18653/v1/2025.naacl-long.533.

[277] Xiangchi Yuan, Chunhui Zhang, Zheyuan Liu, Dachuan Shi, Leyan Pan, Soroush Vosoughi, and Wenke Lee. Superficial self-improved reasoners benefit from model merging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, October 2025. URL https://doi.org/10.18653/v1/2025.emnlp-main.301.

[278] Ye Yu, Xiaopeng Yuan, Haibo Jin, Heming Liu, Yaoning Yu, and Haohan Wang. Do self-evolving agents forget? capability degradation and preservation in lifelong LLM agent adaptation, 2026. URL http://arxiv.org/abs/2605.09315.

[279] Zhenting Qi, Susanna Maria Baby, Stefanie Anna Baby, Kan Yuan, Andrew Tomkins, Tu Vu, Da-Cheng Juan, and Cyrus Rashtchian. On the generalization gap in self-evolving language model reasoning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/61802.

[280] Zhangying Feng, Qianglong Chen, Ning Lu, Yongqian Li, Siqi Cheng, Shuangmu Peng, Duyu Tang, Shengcai Liu, et al. Is PRM necessary? problem-solving RL implicitly induces PRM capability in LLMs. In *Advances in Neural Information Processing Systems*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/2e99eb4e5357addf33fe246d5ad5ee03-Abstract-Conference.html.

[281] Bowen Wei, Nan Wang, Yuqing Zhou, Jinhao Pan, and Ziwei Zhu. Confidence-orchestrated self-evolution against uncertain LLM feedback, 2026. URL https://arxiv.org/abs/2605.28010.

[282] Pieter van Rooyen. First-order recoverability collapse in self-referential information decoders, 2026. URL https://arxiv.org/abs/2606.24861.

[283] Ji Ho Bae. When self-reference fails to close: Matrix-level dynamics in large language models, 2026. URL https://arxiv.org/abs/2604.12128.

[284] Xiaoyu Yang, En Yu, and Jie Lu. Autonomous drift learning in data streams: A unified perspective, 2026. URL https://arxiv.org/abs/2605.01295.

[285] Jingshen Zhang, Bo Wang, Yanlin Fu, Dongming Zhao, Ruifang He, Yuexian Hou, and Zifei Yu. Modeling implicit conflict monitoring mechanisms against stereotypes in LLMs, 2026. URL https://arxiv.org/abs/2605.09647.

[286] Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, et al. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2605.30621.

[287] Ran Yan, Wei Fu, Jiale Li, Shusheng Xu, Zhiyu Mei, Jiaxuan Gao, Jiarui Zhang, Wentai Zhang, et al. Next-generation agentic reinforcement learning systems enable self-evolving agents, 2026. URL https://arxiv.org/abs/2607.01120.

[288] Yongheng Zhang, Ziang Liu, Jiaxuan Zhu, Shuai Wang, Xiangqi Chen, Haojing Huang, Jiayi Kuang, Siyu Chen, et al. From chatbot to digital colleague: The paradigm shift toward persistent autonomous AI, 2026. URL https://arxiv.org/abs/2606.14502.

[289] Daniel Beechey, Derek Yuen, Jianheng Liu, Dezhao Luo, Tiantian He, Weilin Luo, Jun Wang, and Kun Shao. Darwin mobile agent: A roadmap for self-evolution, 2026. URL https://arxiv.org/abs/2606.20622.

[290] Borja Odriozola Schick. The root theorem of context engineering, 2026. URL https://arxiv.org/abs/2604.20874.

[291] Wangcheng Tao, Han Wu, and Weng-Fai Wong. SePO: Self-evolving prompt agent for system prompt optimization, 2026. URL https://arxiv.org/abs/2606.04465.

[292] Md Asif Iqbal Fahim, Oluwadamilola Adebayo, and Alessio Ferrari. Software self-extension with SelfEvolve: an agentic architecture for runtime code generation. In *Proceedings of the 21st International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2026)*, pp. 74–79, 2026. URL https://doi.org/10.1145/3788550.3794870.

[293] Zhiyi Kuang, Ryan Rong, YuCheng Yuan, and Allen Nie. Learning game-playing agents with generative code optimization. In *ICML 2025 Workshop on Programmatic Representations for Agent Learning*, 2025. URL https://openreview.net/forum?id=ZM65X3NoTd.

[294] Boai Sun, Wenjin Guo, Zongmin Yu, and Liu Yang. Self-evolving scientific agent discovers generalizable physically-reasoned fluid control, 2026. URL https://arxiv.org/abs/2606.08405.

[295] Mingen Kuang, Xudong Deng, Xi Lin, Ye Fan, Jianyong Sun, and Jialong Shi. LLM-Driven co-evolutionary automated heuristic design for bi-component coupled combinatorial optimization, 2026. URL https://arxiv.org/abs/2606.00718.

[296] Can Wang, Hongyu Zhao, and Yiqun Chen. InferenceEvolve: Towards automated causal effect estimators through self-evolving AI, 2026. URL https://arxiv.org/abs/2604.04274.

[297] Qianrui Zhou, Hua Xu, Yunjin Gu, Yifan Wang, Songze Li, and Hanlei Zhang. Evolutionary multimodal reasoning via hierarchical semantic representation for intent recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Zhou_Evolutionary_Multimodal_Reasoning_via_Hierarchical_Semantic_Representation_for_Intent_Recognition_CVPR_2026_paper.html.

[298] Víctor Gallego. Beyond scalar rewards: Dense feedback for LLM policy synthesis in sequential social dilemmas. In *ICML 2026 Workshop on New Frontiers in Game-Theoretic Learning (NExT-Game)*, 2026. URL https://openreview.net/forum?id=cnBEqoLMFz.

[299] Mohammad Almansoori, Komal Kumar, and Hisham Cholakkal. MedAgentSim: Self-evolving multi-agent simulations for realistic clinical interactions. In *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, pp. 362–372, 2025. URL https://papers.miccai.org/miccai-2025/0537-Paper2575.html.

[300] Yankai Jiang, Weiting Tang, Haoran Sun, Zhenyu Tang, Yuejie Hou, Yingnan Han, Rubo Wang, Yueyuxiao Yang, et al. A self-evolving agentic system for automated generation and execution of biological protocols, 2026. URL https://arxiv.org/abs/2606.31763.

[301] Oguzhan Gungordu, Siheng Xiong, and Faramarz Fekri. PathWise: Planning through world model for automated heuristic design via self-evolving LLMs. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/63037.

[302] Qianxue Zhang, Yiming Ren, Shihuan Qin, Xiao Zhang, Liao Zhang, Jinyang Huang, Zhengliang Liu, Chenbin Liu, et al. Toward vibe medicine: A self-evolving multi-agent framework for clinical decision support. *Meta-Radiology*, 4(2):100223, 2026. URL https://doi.org/10.1016/j.metrad.2026.100223.

[303] Zhe Zhao, Hongbing Lang, Zhihan Xiao, Luke Ztz Hu, John Imoleayo Adebisi, and Songping Mai. Evidence-driven LLM agent for C-to-synthesizable-C conversion and verification, 2026. URL https://arxiv.org/abs/2606.28409.

[304] Shuang Cui, Fan Ji, Guanglong Sun, Yufei Guo, Xiongxin Tang, Jiangmeng Li, and Fanjiang Xu. Self-evolving agentic image restoration via deliberate planning and intuitive execution, 2026. URL https://arxiv.org/abs/2606.28971.

[305] Yuxuan Liu, Hongda Sun, Wei Liu, Jian Luan, Bo Du, and Rui Yan. MobileSteward: Integrating multiple app-oriented agents with self-evolution to automate cross-app instructions. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1*, pp. 883–893, Toronto ON Canada, July 2025. ISBN 979-8-4007-1245-6. URL https://doi.org/10.1145/3690624.3709171.

[306] Yuan Shui, Yandong Guan, Zhanwei Zhang, Juncheng Hu, Jing Zhang, Dong Xu, and Qian Yu. ArtiCAD: Articulated CAD assembly design via multi-agent code generation, 2026. URL https://arxiv.org/abs/2604.10992.

[307] Ling-Yue Ge and Lan-Zhe Guo. Roles with rails: Contract-preserving role evolution in multi-agent structured reasoning, 2026. URL https://arxiv.org/abs/2605.28433.

[308] Yimeng Wang, Jiaxing Zhao, Hongbin Xie, Hexing Ma, Yuzhen Lei, Shuangxue Liu, Xuan Song, Zichen Zhang, et al. MetaGen: Self-evolving roles and topologies for multi-agent LLM reasoning, 2026. URL https://arxiv.org/abs/2601.19290.

[309] Jiahao Huang, Peilan Xu, Xiaoya Nan, and Wenjian Luo. Co-evolving agent architectures and interpretable reasoning for automated optimization, 2026. URL https://arxiv.org/abs/2604.17708.

[310] Xinshun Feng, Xinhao Song, Lijun Li, Gongshen Liu, and Jing Shao. SEARL: Joint optimization of policy and tool graph memory for self-evolving agents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.1125.

[311] Yingqi Zhang. Agent libOS: A runtime substrate for capability-controlled self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2606.03895.

[312] Chen Ling, Pei Chen, Albert Guan, Jiaming Qu, Shayan Ali Akbar, Madhu Gopinathan, and Erwin Cornejo. PACE: Two-timescale self-evolution for small language model agents, 2026. URL https://arxiv.org/abs/2605.23019.

[313] Yuxuan Zhang, Penghui Du, Bo Li, Cong Wei, Junwen Miao, Huaisong Zhang, Songcheng Cai, Yubo Wang, et al. RewardHarness: Self-evolving agentic post-training, 2026. URL https://arxiv.org/abs/2605.08703.

[314] Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026. URL http://arxiv.org/abs/2604.25850.

[315] Xinyu Zhu, Yuzhu Cai, Zexi Liu, Cheng Wang, Fengyang Li, Wenkai Jin, Wanxu Liu, Zehao Bing, et al. EvoMaster: A foundational evolving agent framework for agentic science at scale, 2026. URL https://arxiv.org/abs/2604.17406.

[316] Wentao Zhang, Zhe Zhao, Haibin Wen, Yingcheng Wu, Cankun Guo, Ming Yin, and Bo An. Autogenesis: A self-evolving agent protocol, 2026. URL https://arxiv.org/abs/2604.15034.

[317] Youhe Jiang, Ran Yan, You Peng, Wenshuang Li, Taiyi Wang, Fangcheng Fu, and Binhang Yuan. Autopoiesis: A self-evolving system paradigm for LLM serving under runtime dynamics, 2026. URL https://arxiv.org/abs/2604.07144.

[318] Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. Group-evolving agents: Open-ended self-improvement via experience sharing, 2026. URL https://arxiv.org/abs/2602.04837.

[319] Zhang Zhang, Shuqi Lu, Hongjin Qian, Di He, and Zheng Liu. AgentFactory: A self-evolving framework through executable subagent accumulation and reuse. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 819–828, 2026. URL https://aclanthology.org/2026.acl-demo.81/.

[320] Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. EvoTest: Evolutionary test-time learning for self-improving agentic systems. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=JFnnajbkvP.

[321] Juan Diego Toscano, Zhaojie Chai, and George Em Karniadakis. GRAFT-ATHENA: Self-improving agentic teams for autonomous discovery and evolutionary numerical algorithms, 2026. URL https://arxiv.org/abs/2605.11117.

[322] Sha Li and Naren Ramakrishnan. Experience as a compass: Multi-agent RAG with evolving orchestration and agent prompts, 2026. URL https://arxiv.org/abs/2604.00901.

[323] Yiwen Zhu, Joyce Cahoon, Anna Pavlenko, Qiushi Bai, Nima Shahbazi, Divya Vermareddy, Meina Wang, Mathieu Demarne, et al. GraphMind: From operational traces to self-evolving workflow automation, 2026. URL https://arxiv.org/abs/2605.17617.

[324] Yufan Dang, Chen Qian, Xueheng Luo, Jingru Fan, Zihao Xie, Ruijie Shi, Weize Chen, Cheng Yang, et al. Multi-agent collaboration via evolving orchestration. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/f1320d2e2842169c6fc89dcbd80e94d0-Abstract-Conference.html.

[325] Chen Xu, Yicheng Hu, Ruizi Wang, Xinyu Lin, Wenjie Wang, Dongrui Liu, and Fuli Feng. TacoMAS: Test-time co-evolution of topology and capability in LLM-based multi-agent systems, 2026. URL https://arxiv.org/abs/2605.09539.

[326] Yaolun Zhang, Tianyi Xu, Shengyu Dai, Zhenwen Shao, Qingyun Wu, and Huazheng Wang. EVOCHAMBER: Test-time co-evolution of multi-agent system at individual, team, and population scales, 2026. URL https://arxiv.org/abs/2605.11136.

[327] Zheng Nie, Ruolin Shen, Xinlei Yu, Bo Yin, Jiangning Zhang, and Xiaobin Hu. SkillGraph: Self-evolving multi-agent collaboration with multimodal graph topology, 2026. URL https://arxiv.org/abs/2604.17503.

[328] Zhaohui Geoffrey Wang. Universe routing: Why self-evolving agents need epistemic control. In *ICLR 2026 Workshop on Lifelong Agents*, 2026. URL https://openreview.net/forum?id=K97fqD2ffS.

[329] Xu Jiang, Bin Chen, Gehui Li, Yule Duan, Ronggang Wang, and Jian Zhang. OctoT2I: A self-evolving agentic text-to-image router. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Jiang_OctoT2I_A_Self-Evolving_Agentic_Text-to-Image_Router_CVPR_2026_paper.html.

[330] Xingjian Wu, Junkai Lu, Siyu Yan, Xiangfei Qiu, Jilin Hu, Chenjuan Guo, and Bin Yang. Differentiable mixture-of-agents incentivizes swarm intelligence of large language models, 2026. URL https://arxiv.org/abs/2605.15706.

[331] Zike Yuan, Yukun Cao, Han Zhang, Jianzhi Yan, Le Liu, Cai ke, Yue Yu, Hui Wang, et al. EGL-SCA: Structural credit assignment for co-evolving instructions and tools in graph reasoning agents, 2026. URL https://arxiv.org/abs/2605.10366.

[332] Yihe Fan, Changyi Li, Lichen Xu, Xudong Pan, Jiarun Dai, Hong Geng, and Min Yang. CyberEvolver: Structured self-evolution for cybersecurity agents on the fly, 2026. URL https://arxiv.org/abs/2605.26195.

[333] Cunxi Yu, Chenhui Deng, Nathaniel Pinckney, and Brucek Khailany. Agentic hardware design as repository-level code evolution, 2026. URL https://arxiv.org/abs/2606.28279.

[334] Changxin Lao, Fei Pan, Guozhuang Ma, Han Li, Huihuang Lin, Jijun Shi, Kangzhi Zhao, Kun Gai, et al. AgentX: Towards agent-driven self-iteration of industrial recommender systems, 2026. URL http://arxiv.org/abs/2606.26859.

[335] Aodi Wu, Haodong Han, Xubo Luo, Ruisuo Wang, Shan He, and Xue Wan. SpaceMind: A modular and self-evolving embodied vision-language agent framework for autonomous on-orbit servicing, 2026. URL https://arxiv.org/abs/2604.14399.

[336] Hanwen Liu, Qihan Zhang, Ryan Marcus, and Ibrahim Sabek. SEFRQO: A self-evolving fine-tuned RAG-based query optimizer. *Proceedings of the ACM on Management of Data*, 3:1–27, 2025. URL https://dl.acm.org/doi/10.1145/3769826.

[337] Rajat Agarwal, Suvidha Tripathi, and Shubham Sharma. PulseCX: Breaking the closed-world assumption in real-time CX, 2026. URL https://arxiv.org/abs/2606.21124.

[338] Haoqin Tu, Jianwen Chen, Zijun Wang, Siwei Han, Juncheng Wu, Hardy Chen, Haonian Ji, Kaiwen Xiong, et al. VisualClaw: A real-time, personalized agent for the physical world, 2026. URL https://arxiv.org/abs/2606.16295.

[339] Mingyue Cheng, Shuo Yu, Daoyu Wang, Qingchuan Li, Xiaoyu Tao, Qingyang Mao, Yitong Zhou, and Qi Liu. TabClaw: An interactive and self-evolving agent for spreadsheet manipulation and table reasoning, 2026. URL https://arxiv.org/abs/2606.10316.

[340] Yiming Lu, Sihang Zeng, Zhengxu Tang, Max Lau, Fei Liu, and Wei Jin. EpiEvolve: Self-evolving agents for streaming pandemic forecasting under regime shifts, 2026. URL https://arxiv.org/abs/2606.05513.

[341] Hejia Geng and Leo Liu. Parthenon law: A self-evolving legal-agent framework, 2026. URL https://arxiv.org/abs/2606.04602.

[342] Sihang Zeng, Matthew Thompson, Ruth Etzioni, and Meliha Yetisgen. Traj-evolve: A self-evolving multi-agent system for patient trajectory modeling in lung cancer early detection, 2026. URL https://arxiv.org/abs/2606.02812.

[343] Yohei Nakajima. The log is the agent: Event-sourced reactive graphs for auditable, forkable agentic systems, 2026. URL https://arxiv.org/abs/2605.21997.

[344] Yuxuan Huang, Yihang Chen, Zhiyuan He, Yuxiang Chen, Ka Yiu Lee, Huichi Zhou, Weilin Luo, Meng Fang, et al. Web2BigTable: A bi-level multi-agent LLM system for internet-scale information search and extraction, 2026. URL https://arxiv.org/abs/2604.27221.

[345] Bochao Liu, Zhipeng Qian, Yang Zhao, Xinyuan Jiang, Zihan Liang, Yufei Ma, Junpeng Zhuang, Ben Chen, et al. Bian que: An agentic framework with flexible skill arrangement for online system operations, 2026. URL https://arxiv.org/abs/2604.26805.

[346] Zhenbo Fu, Yuanzhe Zhang, Qiange Wang, Hao Yuan, Yuehao Xu, Enze Yi, Yanfeng Zhang, and Ge Yu. EvoRAG: Making knowledge graph-based RAG automatically evolve through feedback-driven backpropagation, 2026. URL https://arxiv.org/abs/2604.15676.

[347] Zihao Liu, Hantao Zhou, Jiguo Li, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Peng Wang. MUSE: Multi-domain Chinese user simulation via self-evolving profiles and rubric-guided alignment, 2026. URL https://arxiv.org/abs/2604.13828.

[348] Yao Qin, Yangyang Yan, Jinhua Pang, and Xiaoming Zhang. BloClaw: An omniscient, multi-modal agentic workspace for next-generation scientific discovery, 2026. URL https://arxiv.org/abs/2604.00550.

[349] Yang Zou, Zijian Ding, Yizhou Sun, and Jason Cong. AgRefactor: Self-evolving agentic workflow for HLS compatibility and performance, 2026. URL https://arxiv.org/abs/2606.30949.

[350] Hang Lu, Guochang Li, Qianyu Chen, Huiyan Gao, Shaogang Wang, Xuanyu He, Yiwei Liu, Gaopeng Chen, et al. RFAmpDesigner: A self-evolving multi-agent LLM framework for automated radio frequency amplifier design, 2026. URL https://arxiv.org/abs/2605.10093.

[351] Yi Huang, Bowen Zheng, Yunxi Dong, Hong Tang, Huan Zhao, S. M. Rakibul Hasan Shawon, and Hualiang Zhang. A self-evolving agentic framework for metasurface inverse design, 2026. URL https://arxiv.org/abs/2604.01480.

[352] Buxin She, Brian Chen, Luanzheng Guo, and Fangxing Li. PFAgent: A tractable and self-evolving power-flow agent for interactive grid analysis, 2026. URL https://arxiv.org/abs/2604.10846.

[353] Zhilong Song, Zongmin Zhang, and Lixue Cheng. Autonomous heterogeneous catalyst discovery with a self-evolving multi-agent digital twin, 2026. URL https://arxiv.org/abs/2606.05050.

[354] Junlin He, Yihong Tang, Tong Nie, Ao Qu, Yuebing Liang, Hamzeh Alizadeh, Bang Liu, Wei Ma, et al. MobEvolve: An agentic self-evolving heuristic system for interpretable human mobility generation, 2026. URL https://arxiv.org/abs/2606.01640.

[355] Tong Nie, Yuewen Mei, Yihong Tang, Junlin He, Jie Deng, Jian Sun, and Wei Ma. EvoDrive: Pareto evolution for safety-critical autonomous driving via self-improving LLM agents, 2026. URL https://arxiv.org/abs/2606.03678.

[356] Yuyang Zhou, Guang Cheng, Kang Du, Zihan Chen, and Yuyu Zhao. Toward intelligent and secure cloud: Large language model empowered proactive defense. *IEEE Communications Magazine*, pp. 1–7, 2026. URL https://ieeexplore.ieee.org/document/11373463/.

[357] Yuwen Du, Tian Jin, Jing Kang, Xianghe Pang, Jingyi Chai, Tingjia Miao, Fenyi Liu, WenHao Wang, et al. Towards recursive self-evolving agentic literature retrieval, 2026. URL https://arxiv.org/abs/2605.14306.

[358] Varun Khurana, Vijval Ekbote, Vashu Chauhan, Yaman Kumar Singla, Rajiv Ratn Shah, and Balaji Krishnamurthy. Bridging expert knowledge and automated feature engineering via self-evolution, 2026. URL https://arxiv.org/abs/2606.08800.

[359] Maoqi Liu, Quan Fang, Yuhao Wu, Can Zhao, Yang Yang, and Kaiquan Cai. NOTAM-Evolve: A knowledge-guided self-evolving optimization framework with LLMs for NOTAM interpretation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. URL https://doi.org/10.1609/aaai.v40i1.37043.

[360] Dongjie Huo, Haoyun Liu, Guoqing Liu, Dekang Qi, Zhiming Sun, Maoguo Gao, Jianxin He, Yandan Yang, et al. ABot-Claw: A foundation for persistent, cooperative, and self-evolving robotic agents, 2026. URL https://arxiv.org/abs/2604.10096.

[361] Marco Robol and Paolo Giorgini. Self-evolving software agents, 2026. URL https://arxiv.org/abs/2604.27264.

[362] Ming-Ming Yu, Fei Zhu, Wenzhuo Liu, Yirong Yang, Qunbo Wang, Wenjun Wu, and Jing Liu. C-NAV: Towards self-evolving continual object navigation in open world. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/b78fef5ae6944767b9c1a5a2ae392cde-Abstract-Conference.html.

[363] Alberto Pepe, Chien-Yu Lin, Despoina Magka, Bilge Acun, Yannan Nellie Wu, Anton Protopopov, Carole-Jean Wu, and Yoram Bachrach. Agentic discovery of neural architectures: AIRA-compose and AIRA-design, 2026. URL https://arxiv.org/abs/2605.15871.

[364] Mohammed AbuSadeh, Lan Wei, and Dandan Zhang. TacEvo: Self-evolving architecture discovery for robotic tactile perception via LLM-driven quality-diversity search, 2026. URL https://arxiv.org/abs/2606.30109.

[365] Sixue Xing, Haoyu He, Kerui Wu, Zhuo Yang, Haozheng Luo, Tianfan Fu, and Aarthy Nagarajan. Compute allocation in evolutionary search: From depth-breadth to multi-armed bandits, 2026. URL https://arxiv.org/abs/2605.29268.

[366] Mingxu Tao, Jiawei Hu, Xian Zhou, Wenpeng Hu, Jiajun Cheng, Yunbo Cao, Zhunchen Luo, and Guotong Geng. When rules learn: A self-evolving agent for legal case retrieval. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.2152/.

[367] Marc-Antoine Allard, Arnaud Teinturier, Victor Xing, and Gautier Viaud. Experiential reflective learning for self-improving LLM agents. In *ICLR 2026 Workshop on Memory for Self-Evolving Agents (MemAgents)*, 2026. URL https://openreview.net/forum?id=hQgSl6kj1W.

[368] Shuzheng Si, Haozhe Zhao, Yu Lei, Qingyi Wang, Dingwei Chen, Zhitong Wang, Zhenhailong Wang, Kangyang Luo, et al. From context to skills: Can language models learn from context skillfully?, 2026. URL https://arxiv.org/abs/2604.27660.

[369] Ziqing Zhuang, Linhai Zhang, Jiasheng Si, Deyu Zhou, and Yulan He. Beyond meta-reasoning: Metacognitive consolidation for self-improving LLM reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 23884–23913, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.1095.

[370] Atharva Sehgal, Patrick Yuan, Ziniu Hu, Yisong Yue, Jennifer J Sun, and Swarat Chaudhuri. Self-evolving visual concept library using vision-language critics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13124–13134, 2025. URL https://doi.org/10.1109/CVPR52734.2025.01225.

[371] Gang Liao, Yujia He, Abdullah Ozturk, Zhouyang Li, Ying Wang, Zhitong Guo, Hongsen Qin, Yaobin Qin, et al. Experience graphs: The data foundation for self-improving agents, 2026. URL http://arxiv.org/abs/2606.29823.

[372] Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Zhigang Hua, Yan Xie, et al. ExpGraph: Model-agnostic experience learning with graph-structured memory for LLM agents, 2026. URL https://arxiv.org/abs/2605.30712.

[373] Yuxin Jin, Siyuan Zhang, Hanchen Wang, Lu Qin, Ying Zhang, and Wenjie Zhang. EXG: Self-evolving agents with experience graphs, 2026. URL https://arxiv.org/abs/2605.17721.

[374] Junwei Liao, Haoting Shi, Ruiwen Zhou, Jiaqian Wang, Shengtao Zhang, Wei Zhang, Ying Wen, Zhiyu Li, et al. MemQ: Integrating Q-learning into self-evolving memory agents over provenance DAGs, 2026. URL https://arxiv.org/abs/2605.08374.

[375] Suyash Mishra. Prism: An evolutionary memory substrate for multi-agent open-ended discovery, 2026. URL https://arxiv.org/abs/2604.19795.

[376] Yuqing Yang, Tengxiao Liu, Wang Bill Zhu, Taiwei Shi, Linxin Song, and Robin Jia. Self-evolving LLM memory extraction across heterogeneous tasks, 2026. URL https://arxiv.org/abs/2604.11610.

[377] Yitao Liu, Chenglei Si, Karthik Narasimhan, and Shunyu Yao. Contextual experience replay for self-improvement of language agents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025. URL https://doi.org/10.18653/v1/2025.acl-long.694.

[378] Tianxiang Fei, Mingyang Song, Mao Zheng, and Xiang Yu. Memory beyond recall: A dual-process cognitive memory system for self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2606.09483.

[379] Elzo Brito dos Santos Filho. ESAA-Conversational: An event-sourced memory layer for continuity, handoff, and curation across heterogeneous LLM coding agents, 2026. URL https://arxiv.org/abs/2606.23752.

[380] Weixiang Zhao, Yingshuo Wang, Yichen Zhang, Yanyan Zhao, Yu Zhang, Yang Wu, Dandan Tu, Bing Qin, et al. Rethinking experience utilization in self-evolving language model agents, 2026. URL https://arxiv.org/abs/2605.07164.

[381] Jiaqi Liu, Xinyu Ye, Peng Xia, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. EvolveMem:self-evolving memory architecture via AutoResearch for LLM agents, 2026. URL https://arxiv.org/abs/2605.13941.

[382] Jinglong Gao, Xiao Ding, Yiming Cui, Jianbai Zhao, Hepeng Wang, Ting Liu, and Bing Qin. Self-evolving GPT: A lifelong autonomous experiential learner. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 6385–6432, Bangkok, Thailand, 2024. URL https://doi.org/10.18653/v1/2024.acl-long.346.

[383] Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, et al. AutoSkill: Experience-driven lifelong learning via skill self-evolution, 2026. URL https://arxiv.org/abs/2603.01145.

[384] Ziyue Wang, Cheuk Wang Maurice Ng, Chenchen Yu, Strick Sheng, Kaihua Qin, and Liyi Zhou. Transferable self-evolving playbooks for agentic security auditing, 2026. URL https://arxiv.org/abs/2606.16420.

[385] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, et al. From interactions to principles: Experience-driven self-distillation for evolving LLM agents. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/65641.

[386] Jie-Jing Shao, Haiyan Yin, Yueming Lyu, Xingrui Yu, Lan-Zhe Guo, Ivor Tsang, James Kwok, and Yu-Feng Li. Lifting traces to logic: Programmatic skill induction with neuro-symbolic learning for long-horizon agentic tasks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/65500.

[387] Safayat Bin Hakim, Keyan Guo, Wenkai Tan, Alvaro Velasquez, Shouhuai Xu, and Houbing Herbert Song. ANNEAL: Adapting LLM agents via governed symbolic patch learning, 2026. URL https://arxiv.org/abs/2605.16309.

[388] Zijie Dai, Siuhin He, Hui Li, Qihui Zhou, Jiajun Li, Mingcong Song, Guoping Long, Hongjie Si, et al. Metis: Bridging text and code memory for self-evolving agents, 2026. URL https://arxiv.org/abs/2606.24151.

[389] Jiaqing Liang, Jinyi Han, Weijia Li, Xinyi Wang, Zhoujia Zhang, Zishang Jiang, Ying Liao, Tingyun Li, et al. GenericAgent: A token-efficient self-evolving LLM agent via contextual information density maximization (V1.0), 2026. URL https://arxiv.org/abs/2604.17091.

[390] Wenhan Wang and Zeyu Sun. KBSpec: LLM-driven formal specification generation with evolving domain knowledge base, 2026. URL https://arxiv.org/abs/2606.21339.

[391] Yu Tian, Jiawei Chen, Lifan Zheng, Mingxiang Tao, Xinyi Zeng, Zhaoxia Yin, Hang Su, and Xian Sun. Skills-coach: A self-evolving skill optimizer via training-free GRPO, 2026. URL https://arxiv.org/abs/2604.27488.

[392] Yibo Li, Jiashuo Yang, Zhi Zheng, Zhiyuan Hu, Yuan Sui, Shizun Wang, Yufei He, and Bryan Hooi. APEX: Autonomous policy exploration for self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2605.21240.

[393] Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, et al. SkillOS: Learning skill curation for self-evolving agents, 2026. URL https://arxiv.org/abs/2605.06614.

[394] Ziyang Yu, Qiyue Li, and Liang Zhao. CoCoDA: Co-evolving compositional DAG for tool-augmented agents, 2026. URL https://arxiv.org/abs/2605.08399.

[395] Ruiyi Yang, Zechen Li, Hao Xue, Imran Razzak, and Flora D. Salim. MAGE: Multi-agent self-evolution with co-evolutionary knowledge graphs, 2026. URL https://arxiv.org/abs/2605.10064.

[396] Pan Wang, Yihao Hu, Xiujin Liu, Jingchu Yang, Hang Wang, and Zhihao Wen. AtlasVA: Self-evolving visual skill memory for teacher-free VLM agents, 2026. URL https://arxiv.org/abs/2605.17933.

[397] Juntong Wang, Haoyue Zhao, guanghui Pan, Xiyuan Wang, Yanbo Wang, Qiyan Deng, and Muhan Zhang. SAGE: A self-evolving agentic graph-memory engine for structure-aware associative memory, 2026. URL https://arxiv.org/abs/2605.12061.

[398] Igor Bogdanov, Chung-Horng Lung, Thomas Kunz, Jie Gao, Adrian Taylor, and Marzia Zaman. FORGE: Self-evolving agent memory with no weight updates via population broadcast. In *Proceedings of the ACM Conference on AI and Agentic Systems (CAIS 2026)*, pp. 292–310, 2026. URL https://doi.org/10.1145/3786335.3813155.

[399] Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, Yiyan Ji, et al. SkillRevise: Improving LLM-authored agent skills via trace-conditioned skill revision, 2026. URL https://arxiv.org/abs/2606.01139.

[400] Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. MUSE-Autoskill: Self-evolving agents via skill creation, memory, management, and evaluation, 2026. URL https://arxiv.org/abs/2605.27366.

[401] Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. SkillForge: Forging domain-specific, self-evolving agent skills in cloud technical support. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2026. URL https://doi.org/10.1145/3805712.3808466.

[402] Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. EvoSkill: Automated skill discovery for multi-agent systems, 2026. URL https://arxiv.org/abs/2603.02766.

[403] Qi Zhang, Zhaopeng Feng, Xiaonan Shi, Xiaomeng Hu, Chu Liu, Pengjun Xie, Xiaobin Wang, Jieping Ye, et al. SkillComposer: Learning to evolve agent skills for specification and generalization, 2026. URL https://arxiv.org/abs/2606.06079.

[404] Junli Zha, Jinbo Wang, Chao Zhou, and Xiang Song. Trace2Policy: From expert behavior traces to self-evolving decision agents, 2026. URL https://arxiv.org/abs/2606.10457.

[405] Yangbo Wei, Zhen Huang, Shaoqiang Lu, Junhong Qian, Qifan Wang, Chen Wu, and Lei He. SkillSmith: Co-evolving skills and tools for self-improving agent systems, 2026. URL https://arxiv.org/abs/2606.01314.

[406] Hongjin Qian and Zheng Liu. MetaAgent: Toward self-evolving agent via tool meta-learning, 2025. URL https://arxiv.org/abs/2508.00271.

[407] Xiyang Wu, Zongxia Li, Guangyao Shi, Alexander Duffy, Tyler Marques, Matthew Lyle Olson, Tianyi Zhou, and Dinesh Manocha. Co-evolving LLM decision and skill bank agents for long-horizon tasks, 2026. URL https://arxiv.org/abs/2604.20987.

[408] Shouang Wei, Houcheng Min, Xinpeng Dong, Xin Lin, Sen Cui, Bo Jiang, Zhongxiang Dai, Kun Kuang, et al. MetaForge: A self-evolving multimodal agent that retrieves, adapts, and forges tools on demand, 2026. URL https://arxiv.org/abs/2606.01801.

[409] Mohd Ariful Haque, Justin Williams, Sunzida Siddique, Md. Hujaifa Islam, Hasmot Ali, Kishor Datta Gupta, and Roy George. Advanced tool learning and selection system (ATLASS): a closed-loop framework using LLM. In *Proceedings of the IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2025. URL https://doi.org/10.1109/SOSE67019.2025.00012.

[410] Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, et al. SkillX: Automatically constructing skill knowledge bases for agents, 2026. URL https://arxiv.org/abs/2604.04804.

[411] Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. SKILLFOUNDRY: Building self-evolving agent skill libraries from heterogeneous scientific resources, 2026. URL https://arxiv.org/abs/2604.03964.

[412] Xinyu Zhang, Zhicheng Dou, Deyang Li, Jianjun Tao, Shuo Cheng, Ruifeng Shi, Fangchao Liu, Enrui Hu, et al. Swarm skills: A portable, self-evolving multi-agent system specification for coordination engineering, 2026. URL https://arxiv.org/abs/2605.10052.

[413] Jingbo Yang, Guanyu Yao, Yang Zhang, Ramana Rao Kompella, Gaowen Liu, and Shiyu Chang. FederatedSkill: Federated learning for agentic skill evolution, 2026. URL https://arxiv.org/abs/2606.03143.

[414] Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. SkillClaw: Let skills evolve collectively with agentic evolver, 2026. URL https://arxiv.org/abs/2604.08377.

[415] Huaqing Xie. Forage V2: Knowledge evolution and transfer in autonomous agent organizations, 2026. URL https://arxiv.org/abs/2604.19837.

[416] Zhiling Yan, Dingjie Song, Hanrong Zhang, Wei Liang, Yuxuan Zhang, Yutong Dai, Lifang He, Philip S. Yu, et al. OpenSkill: Open-world self-evolution for LLM agents, 2026. URL https://arxiv.org/abs/2606.06741.

[417] Senwei Xie, Yuntian Zhang, Ruiping Wang, and Xilin Chen. Uni-skill: Building self-evolving skill repository for generalizable robotic manipulation. In *2026 IEEE International Conference on Robotics and Automation (ICRA)*, 2026. URL https://arxiv.org/abs/2603.02623.

[418] Haoran Sun, Wenjie Li, Yujie Zhang, Zekai Lin, Fanrui Zhang, Kaitao Chen, Xingqi He, Yichen Li, et al. Experience makes skillful: Enabling generalizable medical agent reasoning via self-evolving skill memory, 2026. URL https://arxiv.org/abs/2606.09365.

[419] Zherui Yang, Fan Liu, Yansong Ning, and Hao Liu. EvoDS: Self-evolving autonomous data science agent with skill learning and context management. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2026)*, 2026. URL https://doi.org/10.1145/3770855.3818002.

[420] Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. CoEvolve: Training LLM agents via agent-data mutual evolution. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.1055.

[421] Wenjia Jiang, Zongyuan Cai, Yuanhang Shao, Chenru Wang, Boyan Han, Zhixue Song, Keyu Chen, Shengwei An, et al. ManimAgent: Self-evolving multimodal agents for visual education, 2026. URL https://arxiv.org/abs/2606.30296.

[422] Kailin Lyu, Zhiqiang Yuan, Jianwei He, Qiwei Yan, Xuanbo Su, Nanxing Hu, Yang Liu, Ce Hao, et al. PhotoCraft: Agentic reasoning with hierarchical self-evolving memory for deep image search, 2026. URL https://arxiv.org/abs/2606.03099.

[423] Kailin Zhuang, Jiawei Wu, and Zhi Jin. EvoIR-Agent: Self-evolving image restoration agentic system via experience-driven learning, 2026. URL https://arxiv.org/abs/2605.22208.

[424] Rongjun Li, Ziyu Zhou, and Yihang Wu. FlyRoute: Self-evolving agent profiling via data flywheel for adaptive task routing, 2026. URL https://arxiv.org/abs/2605.22057.

[425] Yutao Yang, Junsong Li, Qianjun Pan, Jie Zhou, Kai Chen, Qin Chen, Jingyuan Zhao, Ningning Zhou, et al. PsychAgent: An experience-driven lifelong learning agent for self-evolving psychological counselor, 2026. URL https://arxiv.org/abs/2604.00931.

[426] Xuan Zhang, Wenxuan Zhang, See-Kiong Ng, and Yang Deng. Self-evolving world models for LLM agent planning, 2026. URL https://arxiv.org/abs/2606.30639.

[427] Yijun Ma, Zehong Wang, Yiyang Li, Ziming Li, Xiaoguang Guo, Weixiang Sun, Chuxu Zhang, and Yanfang Ye. ProPlay: Procedural world models for self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2606.12780.

[428] Yi Yu and Tetsunari Inamura. Self-evolving cognitive framework via causal world modeling for embodied scientific intelligence, 2026. URL https://arxiv.org/abs/2606.22449.

[429] Guangya Hao, Yunbo Long, and Zhuokai Zhao. Self-evolving multi-agent systems via decentralized memory, 2026. URL https://arxiv.org/abs/2605.22721.

[430] Jianzong Wang, Botao Zhao, Yayun He, Junqing Peng, and Xulong Zhang. Evolvable embodied agent for robotic manipulation via long short-term reflection and optimization. In *Proceedings of the 2026 International Joint Conference on Neural Networks (IJCNN 2026)*, 2026. URL https://arxiv.org/abs/2604.13533.

[431] Haoyuan Li, Zhengdong Hu, Jun Wang, Hehe Fan, and Yi Yang. Skill-3D: Evolving scene-aware skills for agentic 3D spatial reasoning, 2026. URL https://arxiv.org/abs/2606.07436.

[432] Nga Teng Chan, Yi Zhang, Yechi Liu, Renwen Cui, Fanhu Zeng, Zeyuan Ding, Xiancong Ren, Zhang Zhang, et al. Robo-cortex: A self-evolving embodied agent via dual-grain cognitive memory and autonomous knowledge induction, 2026. URL https://arxiv.org/abs/2605.18729.

[433] Ruofei Ju, Xinrui Wang, Xin Ding, Yifan Yang, Hao Wu, Shiqi Jiang, Qianxi Zhang, Hao Wen, et al. EmbodiSkill: Skill-aware reflection for self-evolving embodied agents, 2026. URL https://arxiv.org/abs/2605.10332.

[434] Zuhao Ge, Xiaosong Jia, Chao Wu, Yuchen Zhou, Zuxuan Wu, and Yu-Gang Jiang. EvoMemNav: Efficient self-evolving fine-grained memory for zero-shot embodied navigation, 2026. URL https://arxiv.org/abs/2606.03509.

[435] Qi Chai, Wenhao Shen, Nanjie Yao, Yue Xia, Kaiyong Zhao, Jie Ma, Guosheng Lin, and Hao Wang. EvolveNav: Proactive preflection and self-evolving memory for zero-shot object goal navigation, 2026. URL https://arxiv.org/abs/2606.18235.

[436] Yunfei Wang, Xiaohao Xu, Yang Li, and Xiaonan Huang. When search becomes memory: Turning robot design trials into transferable skills, 2026. URL https://arxiv.org/abs/2605.25832.

[437] Weixiang Shen, Bailiang Jian, Jun Li, Che Liu, Johannes Moll, Xiaobin Hu, Daniel Rueckert, Hongwei Bran Li, et al. Evo-MedAgent: Beyond one-shot diagnosis with agents that remember, reflect, and improve, 2026. URL https://arxiv.org/abs/2604.14475.

[438] Zihang Fu, Fanxiao Li, Jianyang Gu, Haonan Wang, Preslav Nakov, Bryan Hooi, Min-Yen Kan, and Jiaying Wu. Better with experience: Self-evolving LLM agents for evidence-grounded health community notes, 2026. URL https://arxiv.org/abs/2606.02215.

[439] Jie Zhu, Huaixia Dou, Shuo Jiang, Junhui Li, Lifan Guo, Feng Chen, Chi Zhang, and Fang Kong. ESC-Skills: Discovering and self-evolving skills for emotional support conversations, 2026. URL https://arxiv.org/abs/2605.27908.

[440] Yikun Zhang, Xiwei Cheng, Tianyu Liu, Yuanqi Du, and Wengong Jin. DrugSAGE:self-evolving agent experience for efficient state-of-the-art drug discovery, 2026. URL https://arxiv.org/abs/2605.15461.

[441] Lingyu Mu, Hao Deng, Haibo Xing, Jinxin Hu, Yu Zhang, and Xiaoyi Zeng. EvoRec: Self evolving agentic recommender systems, 2026. URL https://arxiv.org/abs/2606.28368.

[442] Xidong Wu, Yue Zhuan, Ruoqiao Wei, Hangxin Chen, Di Bai, Jintao Liu, Xinyi Wang, Xue Wang, et al. AgenticRecTune: Multi-agent with self-evolving skillhub for recommendation system optimization, 2026. URL https://arxiv.org/abs/2604.26969.

[443] Zhen Tao, Riwei Lai, Chenyun Yu, Weixin Chen, Li Chen, Beibei Kong, Lei Cheng, Chengxiang Zhuo, et al. SAGER: Self-evolving user policy skills for recommendation agent, 2026. URL https://arxiv.org/abs/2604.14972.

[444] Ruofan Jin, Zaixi Zhang, Mengdi Wang, and Le Cong. STELLA: Self-evolving LLM agent for biomedical research, 2025. URL https://arxiv.org/abs/2507.02004.

[445] Zihang Zhou, Ziqian Ren, Yukai Wu, Yingjie Xiong, Wei Zhou, Chao Peng, Dong Zhang, Bingheng Yan, et al. SetupX: Can LLM agents learn from past failures in functionality-correct code repository setup?, 2026. URL https://arxiv.org/abs/2605.26186.

[446] Haichuan Hu, Guoqing Xie, Quanjun Zhang, Jiawei Liu, Shengcheng Yu, Chunrong Fang, Zhenyu Chen, and Liang Xiao. EvoRepair: Enhancing vulnerability repair agents through experience-based self-evolution, 2026. URL https://arxiv.org/abs/2605.30105.

[447] Jincheng Ren, Siwei Wu, Yizhi Li, Kang Zhu, Shu Xu, Boyu Feng, Ruibin Yuan, Wei Zhang, et al. A self-evolving framework for efficient terminal agents via observational context compression, 2026. URL https://arxiv.org/abs/2604.19572.

[448] Haoliang Ming, Feifei Li, Xiaoqing Wu, and Wenhui Que. Retrieval as reasoning: Self-evolving agent-native retrieval via LLM-wiki, 2026. URL https://arxiv.org/abs/2605.25480.

[449] Jingjing Liu, Ziye Huang, Zihao Cheng, Zeming Liu, Jiahong Wu, Yuhang Guo, Kehai Chen, Yunhong Wang, et al. DocOS: Towards proactive document-guided actions in GUI agents. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/65725.

[450] Pianran Guo, Pengcheng Zhou, Yucheng Jian, Shuhua Chen, Zhonfliang Yang, and Linna Zhou. FinAcumen: Financial multimodal reasoning via self-evolving experience memory harness, 2026. URL https://arxiv.org/abs/2606.17642.

[451] Hang Yu, Zifan Zheng, Jeff Z. Pan, Tongliang Liu, Zhiyong Wang, and Fengxiang He. AlphaMemo: Structured search-process memory for self-evolving Alpha mining agents, 2026. URL https://arxiv.org/abs/2606.20625.

[452] Zihao Deng, Yining Zhu, Leiming Wang, Jingfei Lu, Junbo Wang, Chuncheng Ran, Yu Yang, Dixuan Yang, et al. Tree-of-experience: A structured experience-management solution for self-evolving agents under low-repetition and implicit-reward environments, 2026. URL https://arxiv.org/abs/2606.06960.

[453] Zongxia Li, Dawei Liu, Fuxiao Liu, Yuhang Zhou, Xiyang Wu, Jingxi Chen, Jing Xie, Xiaomin Wu, et al. COMFYCLAW: Self-evolving skill harnesses for image generation workflows, 2026. URL https://arxiv.org/abs/2607.01709.

[454] Haiwen Li, Jing Tang, Rui Chen, Lei Sun, and Xiangxiang Chu. M2Note: Continual evolution of vision language models via mistake notebook learning, 2026. URL https://arxiv.org/abs/2607.00685.

[455] Pan Wang. REFLEX: Reflective evolution from LLM experience, 2026. URL https://arxiv.org/abs/2606.16496.

[456] Xingtian Pei, Yukun Song, Changwei Wang, Shunpeng Chen, Rongtao Xu, Shengpeng Xu, and Shibiao Xu. DeliCIR: Memory-guided test-time deliberation via multi-agent collaboration for composed image retrieval, 2026. URL https://arxiv.org/abs/2605.22478.

[457] Lin Li, Jiawei Huang, Qihao Quan, Dan Li, Boxin Li, Xiao Zhang, Erli Meng, Wenjie Feng, et al. Empowering VLMs for few-shot multimodal time series classification via tailored agentic reasoning, 2026. URL https://arxiv.org/abs/2605.09395.

[458] Feng Xiong, Zengbin Wang, Yong Wang, Xuecai Hu, Jinghan He, Liang Lin, Yuan Liu, and Xiangxiang Chu. Ace-skill: Bootstrapping multimodal agents with prioritized and clustered evolution, 2026. URL https://arxiv.org/abs/2605.08887.

[459] Xinyu Che, Junqi Xiong, Yunfei Ge, Xinping Lei, Shihao Li, Hang Yan, Han Li, Yuanxing Zhang, et al. MMG2Skill: Can agents distill in-the-wild guides into self-evolving skills?, 2026. URL https://arxiv.org/abs/2606.01993.

[460] Wenlun Zhang, Jun Yin, and Kentaro Yoshioka. Detect in any scene: An agentic framework for object detection with experience-aware reasoning, 2026. URL https://arxiv.org/abs/2605.31174.

[461] Wangding Xia, Ye Du, Jiashi Lin, Meng Wang, Danli Shi, and Shujun Wang. Evo-RAD: Navigating rare retinal disease diagnosis via self-evolving agentic retrieval. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI 2026)*, 2026. URL https://arxiv.org/abs/2606.22955.

[462] Tao Feng, Pengrui Han, Guanyu Lin, Ge Liu, and Jiaxuan You. Thought-retriever: Don't just retrieve raw data, retrieve thoughts for memory-augmented agentic systems. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=emCcuhtENL.

[463] Tran Chi Nguyen, Dao Sy Duy Minh, Trung Kiet Huynh, Pham Phu Hoa, Nguyen Lam Phu Quy, and Vu Nguyen. MEMRES: A memory-augmented resolver with confidence cascade for agentic Python dependency resolution. In *Companion Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering (FSE Companion 2026)*, 2026. URL https://doi.org/10.1145/3803437.3808242.

[464] Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents, 2026. URL https://arxiv.org/abs/2602.02474.

[465] Haidong Xin, Xinze Li, Zhenghao Liu, Yukun Yan, Shuo Wang, Cheng Yang, Yu Gu, Ge Yu, et al. MetaMem: Evolving meta-memory for knowledge utilization through self-reflective symbolic optimization. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 5473–5492, 2026. URL https://aclanthology.org/2026.findings-acl.270/.

[466] Haoran Xu, Jiacong Hu, Ke Zhang, Lei Yu, Yuxin Tang, Xinyuan Song, Yiqun Duan, Lynn Ai, et al. SEDM: Scalable self-evolving distributed memory for agents. In *NeurIPS 2025 Workshop on Scaling Environments for Agents (SEA)*, 2025. URL https://openreview.net/forum?id=TA1Ocu9ZZp.

[467] Jiawei Yu, Yixiang Fang, Xilin Liu, and Yuchi Ma. H-Mem: A novel memory mechanism for evolving and retrieving agent memory via a hybrid structure, 2026. URL https://arxiv.org/abs/2605.15701.

[468] Shilong Jin, Lanjun Wang, and Zhuosheng Zhang. SE-GA: Memory-augmented self-evolution for GUI agents. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/65853.

[469] Tooraj Helmi. Decentralizing AI memory: SHIMI, a semantic hierarchical memory index for scalable agent reasoning, 2025. URL https://arxiv.org/abs/2504.06135.

[470] Yaxiong Wu, Yongyue Zhang, Sheng Liang, and Yong Liu. SGMem: Sentence graph memory for long-term conversational agents, 2025. URL https://arxiv.org/abs/2509.21212.

[471] Sheng Jin, Haoming Wang, Zhiqi Gao, Yongbo Yang, Bao Chunjia, and Chengliang Wang. Evolution in simulation: AI-agent school with dual memory for high-fidelity educational dynamics. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025. URL https://aclanthology.org/2025.findings-emnlp.312/.

[472] Xiaoxing Wang, Ning Liao, Shikun Wei, Chen Tang, and Feiyu Xiong. AutoAgent: Evolving cognition and elastic memory orchestration for adaptive agents, 2026. URL https://arxiv.org/abs/2603.09716.

[473] Namyoung Kim, Kai Tzu-iunn Ong, Yeonjun Hwang, Minseok Kang, Iiseo Jihn, Gayoung Kim, Minju Kim, and Jinyoung Yeo. PRINCIPLES: Synthetic strategy memory for proactive dialogue agents. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025. URL https://aclanthology.org/2025.findings-emnlp.1164/.

[474] Prince Zizhuang Wang and Shuli Jiang. PRIME: Training free proactive reasoning via iterative memory evolution for user-centric agent, 2026. URL https://arxiv.org/abs/2604.07645.

[475] Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. OS-copilot: Towards generalist computer agents with self-improvement. In *ICLR 2024 Workshop on Large Language Model Agents*, 2024. URL https://openreview.net/forum?id=3WWFrg8UjJ.

[476] Zihao Cheng, Zeming Liu, Yingyu Shan, Xinyi Wang, Xiangrong Zhu, Yunpu Ma, Hongru Wang, Yuhang Guo, et al. Mem^{2}evolve: Towards self-evolving agents via co-evolutionary capability expansion and experience distillation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 20784–20831, 2026. URL https://doi.org/10.18653/v1/2026.acl-long.952.

[477] Xuan Yao, Junyu Gao, and Changsheng Xu. NavMorph: A self-evolving world model for vision-and-language navigation in continuous environments. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. URL https://doi.org/10.1109/ICCV51701.2025.00525.

[478] Zhenyu Guan, Xiangyu Kong, Fangwei Zhong, and Yizhou Wang. Richelieu: Self-evolving LLM-based agents for AI Diplomacy. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/df2d62b96a4003203450cf89cd338bb7-Abstract-Conference.html.

[479] Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, Liming Zhu, and Wenjie Zhang. MemoTime: Memory-augmented temporal knowledge graph enhanced large language model reasoning. In *Proceedings of the ACM Web Conference (WWW)*, 2026. URL https://doi.org/10.1145/3774904.3792581.

[480] Yufei He, Ruoyu Li, Alex Chen, Yue Liu, Yulin Chen, Yuan Sui, Cheng Chen, Yi Zhu, et al. Enabling self-improving agents to learn at test time with human-in-the-loop guidance. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, October 2025. URL https://aclanthology.org/2025.emnlp-industry.115/.

[481] Yaolun Zhang, Yiran Wu, Yijiong Yu, Qingyun Wu, and Huazheng Wang. Live-evo: Online evolution of agentic memory from continuous feedback, 2026. URL https://arxiv.org/abs/2602.02369.

[482] Dylan Zhang, Yanshan Lin, Zhengkun Wu, Yihang Sun, Bingxuan Li, Dianqi Li, and Hao Peng. Useful memories become faulty when continuously updated by LLMs, 2026. URL https://arxiv.org/abs/2605.12978.

[483] Chuyang Wei, Maohang Gao, Zhixin Han, Kefei Chen, Yu Zhuang, Haoxiang Guan, Yanzhi Zhang, Yilin Cheng, et al. Harnessing pre-resolution signals for future prediction agents, 2026. URL https://arxiv.org/abs/2604.15719.

[484] Md Nayem Uddin, Amir Saeidi, Eduardo Blanco, and Chitta Baral. LedgerAgent: Structured state for policy-adherent tool-calling agents, 2026. URL https://arxiv.org/abs/2606.20529.

[485] Wentao Hu, Zhendong Chu, Yiming Zhang, Junda Wu, Ming Jin, Xiangyu Zhao, Yilei Shao, Yanfeng Wang, et al. SkillBrew: Multi-objective curation of skill banks for LLM agents, 2026. URL https://arxiv.org/abs/2605.29440.

[486] Xing Zhang, Yanwei Cui, Guanghui Wang, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Ratchet: A minimal hygiene recipe for self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2605.22148.

[487] Hui Zhang and Shuren Song. Written by AI, managed by AI: Semantic space control and index sickness elimination across 391 consecutive sessions, 2026. URL https://arxiv.org/abs/2606.19121.

[488] Jiazheng Li, Emine Yilmaz, Bei Chen, and Dieu-Thu Le. Towards self-improving error diagnosis in multi-agent systems. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.98/.

[489] Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable LLM agents: Diagnosing and repairing harness flaws, 2026. URL https://arxiv.org/abs/2606.06324.

[490] Xing Zhang, Yanwei Cui, Guanghui Wang, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Library drift: Diagnosing and fixing a silent failure mode in self-evolving LLM skill libraries. In *Proceedings of the ICML 2026 Workshop on Failure Modes in Agentic AI (FAGEN)*, 2026. URL https://arxiv.org/abs/2605.19576.

[491] Andrew Borthwick, Stephen Ash, and Anthony Galczak. RoboPhD: Evolving diverse complex agents under tight evaluation budgets, 2026. URL https://arxiv.org/abs/2604.04347.

[492] Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents, 2026. URL http://arxiv.org/abs/2603.19461.

[493] Qianshu Cai, Yonggang Zhang, Xianzhang Jia, Huajiang Zheng, Wei Xue, Jun Song, Xinmei Tian, and Yike Guo. MOSS: Self-evolution through source-level rewriting in autonomous agent systems, 2026. URL http://arxiv.org/abs/2605.22794.

[494] Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-SWE-agent: Can software engineering agents self-evolve on the fly?, 2025. URL http://arxiv.org/abs/2511.13646.

[495] Chengyu Huang, Sheng-Yen Chou, Zhengxin Zhang, and Claire Cardie. Bootstrapping post-training signals for open-ended tasks via rubric-based self-play on pre-training text, 2026. URL https://arxiv.org/abs/2604.20051.

[496] Chi Zhang, Haibo Qiu, Qiming Zhang, Yufei Xu, Xinbo Gao, and Jing Zhang. Seirênes: Adversarial self-play with evolving distractions for LLM reasoning, 2026. URL https://arxiv.org/abs/2605.11636.

[497] Ran Li, Zeyuan Liu, Yinghao Chen, Bingxiang He, Jiarui Yuan, Zixuan Fu, Weize Chen, Jinyi Hu, et al. CPMobius: Iterative coach-player reasoning for data-free reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/64791.

[498] Leheng Sheng, Wenchang Ma, Ruixin Hong, Xiang Wang, An Zhang, and Tat-Seng Chua. Reinforcing chain-of-thought reasoning with self-evolving rubrics, 2026. URL http://arxiv.org/abs/2602.10885.

[499] Shuqi Lu, Chaofan Li, Kun Luo, Zhang Zhang, Hui Wang, Hongwang Xiao, Zheng Liu, Lei Xiong, et al. AREX: Towards a recursively self-improving agent for deep research, 2026. URL http://arxiv.org/abs/2607.21461.

[500] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery, 2025. URL http://arxiv.org/abs/2506.13131.

[501] Gang Liu, Yihan Zhu, Jie Chen, and Meng Jiang. Scientific algorithm discovery by augmenting AlphaEvolve with deep research, 2025. URL http://arxiv.org/abs/2510.06056.

[502] Alexey Kravatskiy, Valentin Khrulkov, and Ivan Oseledets. ImprovEvolve: Basin-hopping meets LLM-guided evolutionary search, 2026. URL http://arxiv.org/abs/2602.10233.

[503] Kazuki Ota, Takayuki Osa, and Tatsuya Harada. Self-supervised theorem discovery in a formal axiomatic system. In *Proceedings of the 3rd AI for Math Workshop at ICML 2026*, 2026. URL https://openreview.net/forum?id=1c85W1WW9k.

[504] Tao Liu, Ye Lu, Ruohua Zhang, Siyu Song, Wentao Liu, Aimin Zhou, and Hao Hao. Elmes*: Automated construction of fine-grained evaluation rubrics for large language models in long-tail educational scenarios, 2026. URL https://arxiv.org/abs/2606.06546.

[505] Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, et al. Self-taught evaluators, 2024. URL https://arxiv.org/abs/2408.02666.

[506] Tianhao Wu, Weizhe Yuan, Olga Golovneva, Jing Xu, Yuandong Tian, Jiantao Jiao, Jason E. Weston, and Sainbayar Sukhbaatar. Meta-rewarding language models: Self-improving alignment with llm-as-a-meta-judge. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 11537–11554, 2025. URL https://doi.org/10.18653/v1/2025.emnlp-main.583.

[507] Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, et al. Dynabench: Rethinking benchmarking in NLP. In *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pp. 4110–4124, 2021. URL https://aclanthology.org/2021.naacl-main.324/.

[508] Jia Li, Ge Li, Xuanming Zhang, Yunfei Zhao, Yihong Dong, Zhi Jin, Binhua Li, Fei Huang, et al. Evocodebench: An evolving code generation benchmark with domain-specific evaluations. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/6a059625a6027aca18302803743abaa2-Abstract-Datasets_and_Benchmarks_Track.html.

[509] Zhiqiu Xu, Shibo Jin, Shreya Arya, and Mayur Naik. MathDuels: Evaluating LLMs as problem posers and solvers, 2026. URL https://arxiv.org/abs/2604.21916.

[510] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. POET: Open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2019. URL https://doi.org/10.1145/3321707.3321799.

[511] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020. URL https://papers.nips.cc/paper_files/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

[512] Samuel Cahyawijaya, Delong Chen, Yejin Bang, Leila Khalatbari, Bryan Wilie, Ziwei Ji, Etsuko Ishii, and Pascale Fung. High-dimension human value representation in large language models. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 5303–5330, Albuquerque, New Mexico, April 2025. URL https://doi.org/10.18653/v1/2025.naacl-long.274.

[513] Tianyi Qiu, Yang Zhang, Xuchuan Huang, Jasmine Xinze Li, Jiaming Ji, and Yaodong Yang. ProgressGym: Alignment with a millennium of moral progress. In *Advances in Neural Information Processing Systems (NeurIPS)*, October 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/1a6d49c1a298ebb799d005b7b90ab31d-Abstract-Datasets_and_Benchmarks_Track.html.

[514] Jing Yao, Xiaoyuan Yi, and Xing Xie. CLAVE: An adaptive framework for evaluating values of LLM generated responses. In *Advances in Neural Information Processing Systems*, volume 37, pp. 58868–58900, 2024. URL http://papers.nips.cc/

paper_files/paper/2024/hash/6c1d2496c04d1ef648d58684b699643f-Abstract-Datasets_and_Benchmarks_Track.html.

[515] Jingnan Zheng, Han Wang, An Zhang, Tai D. Nguyen, Jun Sun, and Tat-Seng Chua. ALI-Agent: Assessing LLMs' alignment with human values via agent-based evaluation. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/b35c38f70065ac6c694089ca93a015bb-Abstract-Conference.html.

[516] Grandee Lee, Yue Wang, Che Yee Lye, and Luke Peh. Generative-evaluative agreement: A necessary validity criterion for LLM-enabled adaptive assessment. In *Proceedings of the 21st Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2026)*, pp. 798–812, 2026. URL https://aclanthology.org/2026.bea-1.54/.

[517] Weixiang Zhao, Yingshuo Wang, Yichen Zhang, Yang Deng, Yanyan Zhao, Wanxiang Che, Bing Qin, and Ting Liu. Large language model agents are not always faithful self-evolvers. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2026. URL https://icml.cc/virtual/2026/poster/62034.

[518] Zayx Shawn. PACE: Anytime-valid acceptance tests for self-evolving agents, 2026. URL https://arxiv.org/abs/2606.08106.

[519] Biswa Sengupta. Self-evolving agents with anytime-valid certificates, 2026. URL https://arxiv.org/abs/2607.00871.

[520] Michael Nguyen, Quoc Nguyen, and Paul Vuong. Recursive self-evolving agents via held-out selection, 2026. URL https://arxiv.org/abs/2606.28374.

[521] Ruixiao Lin, Xinhao Deng, Qingming Li, Jianan Ma, Yunhao Feng, Yuqi Qing, Zhenyuan Li, Yechao Zhang, et al. Safety in self-evolving LLM agent systems: Threats, amplification, and case studies, 2026. URL https://arxiv.org/abs/2606.23075.

[522] Xianglin Yang, Yufei He, Shuo Ji, Bryan Hooi, and Jin Song Dong. Zombie agents: Persistent control of self-evolving LLM agents via self-reinforcing injections. In *ICLR 2026 Workshop on Lifelong Agents*, 2026. URL https://openreview.net/forum?id=OdXgAvBiCl.

[523] Chien-Ping Lu. The computational boundary of inference: Capability internalization, training, and the Turing jump, 2026. URL https://arxiv.org/abs/2605.27381.

[524] Congjie Zheng, Chuanyi Xue, Bin Liang, Jun Yang, and Changshui Zhang. SEAGym: An evaluation environment for self-evolving LLM agents, 2026. URL https://arxiv.org/abs/2606.17546.

[525] Jiahao Huang, Fei Cheng, Junfeng Jiang, Zefan Yu, and Akiko Aizawa. BenchTrace: A benchmark for testing reflection ability and controlled evolution in LLM agents, 2026. URL https://arxiv.org/abs/2605.29225.

[526] Linyue Pan, Yaoming Zhu, Lin Qiu, Xuezhi Cao, and Xunliang Cai. SAGE: A quantitative evaluation of socialized evolution in agent ecosystems, 2026. URL https://arxiv.org/abs/2606.03544.

[527] Joshua Sherwood, Ben Aybar, and Benjamin Kaplan. Frontier coding agents can now implement an AlphaZero self-play machine learning pipeline for connect four that performs comparably to an external solver, 2026. URL https://arxiv.org/abs/2604.25067.

[528] Xinyu Lu, Tianshu Wang, Pengbo Wang, zujie wen, Zhiqiang Zhang, Jun Zhou, Boxi Cao, Yaojie Lu, et al. The meta-agent challenge: Are current agents capable of autonomous agent development?, 2026. URL https://arxiv.org/abs/2606.04455.

[529] Sihang Jiang, Lipeng Ma, Zhonghua Hong, Keyi Wang, Zhiyu Lu, Tengfei Wang, Shisong Chen, Jinghao Zhang, et al. SEA-Eval: A benchmark for evaluating self-evolving agents beyond episodic assessment, 2026. URL https://arxiv.org/abs/2604.08988.

[530] Yuyao Wang, Zhongjian Zhang, Mo Chi, Kaichi Yu, Yuhan Li, Miao Peng, Bing Tong, Chen Zhang, et al. EvoMemBench: Benchmarking agent memory from a self-evolving perspective, 2026. URL http://arxiv.org/abs/2605.18421.

[531] Jiarui Yuan, Tailin Jin, Weize Chen, and Zeyuan Liu. SE-Bench: Benchmarking self-evolution with knowledge internalization, 2026. URL https://arxiv.org/abs/2602.04811.

[532] Shuhan Xue, Zixin Ding, Yichen Shen, Yinjie Wang, Zhenfei Yin, Yingcheng Wu, Yuxin Chen, Mengdi Wang, et al. PAST-Bench: Benchmarking the foundations of recursive self-improvement in personal agents, 2026. URL https://arxiv.org/abs/2608.04003.

[533] Fanqing Meng, Lingxiao Du, Qiguang Chen, Ziqi Zhao, Haocheng Lu, Mengkang Hu, and Michael Qizhe Shieh. RSIBench-Data: Benchmarking data-centric research for recursive self-improvement, 2026. URL https://arxiv.org/abs/2607.25886.

[534] Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. PostTrainBench: Can LLM agents automate LLM post-training?, 2026. URL https://arxiv.org/abs/2603.08640.

[535] Wanyi Chen, Xiao Yang, Xu Yang, Tianming Sha, Qizheng Li, Zhuo Wang, Bowen Xian, Fang Kong, et al. Agent2 RL-bench: Can LLM agents engineer agentic RL post-training?, 2026. URL https://arxiv.org/abs/2604.10547.

[536] Tim R. Davidson, Veniamin Veselovsky, Michal Kosinski, and Robert West. Evaluating language model agency through negotiations. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=3ZqKxMHcAg.

[537] Haonian Ji, Kaiwen Xiong, Siwei Han, Peng Xia, Shi Qiu, Yiyang Zhou, Jiaqi Liu, Jinlong Li, et al. ClawArena: Benchmarking AI agents in evolving information environments, 2026. URL https://arxiv.org/abs/2604.04202.

[538] Yizhe Chi, Deyao Hong, Dapeng Jiang, Tianwei Luo, Kaisen Yang, Boshi Zhang, Zhe Cao, Xiaoyan Fan, et al. Frontier-eng: Benchmarking self-evolving agents on real-world engineering tasks with generative optimization, 2026. URL https://arxiv.org/abs/2604.12290.

[539] Tristan Thrush, Jared Moore, Miguel Monares, Christopher Potts, and Douwe Kiela. I am a strange dataset: Metalinguistic tests for language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 8888–8907, Bangkok, Thailand, 2024. URL https://doi.org/10.18653/v1/2024.acl-long.482.

[540] Yuming, Huang, Yao Liu, Pengjie Ding, Lei Wang, and Junchen Wan. Does capability transfer to subjective behavior – and would our instruments tell us? A self-evolving, trust-by-construction evaluation paradigm, 2026. URL https://arxiv.org/abs/2605.27914.

[541] Shuaimin Li, Liyang Fan, Zeyang Li, Zhuoyue Wan, Yufang Lin, Shiwen Ni, Feiteng Fang, Hamid Alinejad-Rokny, et al. SrDetection: A self-referential framework for data leakage detection in code large language models, 2026. URL https://arxiv.org/abs/2606.29815.

[542] Leyi Sheng, Han Sun, Zhen Sun, Yuntao Yue, Jinlin Wu, Xinlei He, and Jiaheng Wei. PixJail: Self-evolving paper-to-pipeline reproduction for text-to-image jailbreak evaluation, 2026. URL https://arxiv.org/abs/2606.24081.

[543] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. In *NeurIPS 2025 Workshop on Scaling Environments for Agents (SEA)*, 2025. URL https://neurips.cc/virtual/2025/124651.

[544] Han Xiao, Guozhi Wang, Hao Wang, Shilong Liu, Yuxiang Chai, Yue Pan, Yufeng Zhou, Xiaoxin Chen, et al. UI-Mem: Self-evolving experience memory for online reinforcement learning in mobile GUI agents, 2026. URL https://arxiv.org/abs/2602.05832.

[545] Fenglin Yu, Fangkai Yang, Xiaoting Qin, Zhiyang Zhang, Jue Zhang, Qingwei Lin, Hongyu Zhang, Yingnong Dang, et al. Enabling autonomic microservice management through self-learning agents, 2025. URL https://arxiv.org/abs/2501.19056.

[546] Wenli Xiao, Jia Xie, Tonghe Zhang, Haotian Lin, Letian "Max" Fu, Haoru Xue, Jalen Lu, Yi Yang, et al. ENPIRE: Agentic robot policy self-improvement in the real world, 2026. URL https://arxiv.org/abs/2606.19980.

[547] Yanlong Wang, Jian Xu, Hongkang Zhang, Shao-Lun Huang, Danny Dongning Sun, and Xiao-Ping Zhang. FactorMiner: a self-evolving agent with skills and experience memory for financial alpha discovery. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=TTsecyqrW3.

[548] Debdeep Sanyal, Agniva Maiti, Umakanta Maharana, Dhruv Kumar, Ankur Mali, C. Lee Giles, and Murari Mandal. Investigating pedagogical teacher and student LLM agents: Genetic adaptation meets retrieval-augmented generation across learning styles. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 13348–13389, Suzhou, China, November 2025. ISBN 979-8-89176-332-6. URL https://doi.org/10.18653/v1/2025.emnlp-main.675.

[549] Sico Team. Agentic evolution: From self-improving agents to co-evolving human–AI systems, 2026. URL https://www.microsoft.com/en-us/research/publication/agentic-evolution-from-self-improving-agents-to-co-evolving-human-ai-systems/.