LEARNING TO IMPROVE: TOWARDS LEARNABLE RECURSIVE SELF-IMPROVEMENT FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive self-improvement (RSI) enables large language model (LLM) agents to continuously improve from interaction experience. However, existing RSI methods typically rely on fixed improvement strategies, making it difficult for agents to learn how to improve from previous improvement outcomes. To address this issue, we propose Learnable Recursive Self-Improvement (LRSI), which makes the self-improvement process itself learnable. LRSI introduces an Improvement Agent that diagnoses task failures and selects appropriate improvement actions to guide the Answer Agent. It further uses the outcomes of previous improvement decisions to refine how future failures are diagnosed and how improvement actions are selected and executed. Candidate updates are validated through replay and regression checks before being accepted. Experiments on executable agent tasks show that LRSI improves task success over fixed self-improvement baselines across multiple LLMs. Further analyses show that LRSI can correct inappropriate improvement decisions and produce more reliable interventions. These results demonstrate that LLM agents can learn not only to improve their task-solving capabilities, but also to improve how they improve.

1 INTRODUCTION

Large language model (LLM) agents have shown strong capabilities in solving complex tasks through reasoning, tool use, and environment interaction (Yao et al., 2023; Qin et al., 2024; Ichter et al., 2022). Recent studies further enable agents to improve from their own experience. For example, agents can refine previous responses, accumulate verbal feedback, construct reusable skills, and even optimize their workflows or programs (Madaan et al., 2023; Shinn et al., 2023; Wang et al., 2023; Hu et al., 2025; Zhang et al., 2025a). These advances move LLM agents from static task solvers toward systems that can continuously improve through interaction.

However, improving an agent requires more than simply receiving feedback. After a failed attempt, the agent must first determine *what should be improved*. A failure may come from incorrect execution, missing task knowledge, an inappropriate skill, or even an incorrect improvement decision. These cases require different interventions. For example, if an agent fails to generate a required artifact because it does not follow an existing skill, rewriting the skill is unnecessary. In contrast, if the required procedure is absent from the available skills, repeated execution feedback cannot provide the missing capability. Therefore, effective self-improvement requires not only feedback, but also an appropriate decision about how that feedback should be used.

Existing self-improvement methods have substantially expanded the objects that an agent can update, ranging from responses and memories to skills, workflows, and agent programs (Zhao et al., 2023; Zheng et al., 2025; Xia et al., 2026; Zhang et al., 2025b; Zelikman et al., 2023). However, the procedure that determines *how to improve* is commonly predefined. For example, a method may repeatedly apply reflection, revise a skill after failure, or search over a fixed space of agent programs. Such procedures can improve task performance, but they do not necessarily improve the agent's ability to make better improvement decisions. If the improvement procedure repeatedly attributes execution failures to skill defects, accumulating more task skills does not correct this behavior.

This observation motivates a more general question: *can an agent learn how to improve?* Recursive self-improvement (RSI) provides a natural framework for this problem because it allows the mechanism responsible for improvement to itself become an object of improvement (Zelikman et al., 2023;

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Zhang et al., 2025a). We argue that RSI for LLM agents should operate at two levels. The first level improves the task-solving capability of the agent. The second level improves the process that determines how such improvements are performed. In other words, an agent should learn not only from task experience, but also from its previous *improvement experience*.

To this end, we propose **Learnable Recursive Self-Improvement (LRSI)**, which makes the self-improvement process explicitly learnable. LRSI introduces an *Improvement Agent* that analyzes verifier feedback and execution trajectories before selecting an appropriate improvement action for the *Answer Agent*. Different failures can therefore trigger different interventions, including execution guidance, memory updates, or persistent task-skill changes. Persistent updates are further evaluated through candidate replay and regression checks before they are accepted. More importantly, LRSI records the outcomes of these improvement decisions as improvement experience. This experience provides evidence for diagnosing failures of the improver itself and for refining how future failures are diagnosed and how improvement actions are selected.

Our main contributions are summarized as follows:

- To the best of our knowledge, we are the first to define learnable recursive self-improvement for LLM agents as a two-level problem: improving task-solving capabilities while also improving the mechanism that determines how future improvements are performed.
- We propose LRSI, an evidence-grounded self-improvement framework in which an Improvement Agent diagnoses failures from verifier feedback and execution evidence, selects among different improvement actions, and validates persistent updates through candidate replay and regression checks before adoption.
- We conduct a comprehensive evaluation of LRSI on executable agent tasks from SkillLearnBench (Zhong et al., 2026), covering task recovery, intervention diagnostics, cross-model outcome analysis, and controlled update validation. The results show that LRSI improves observed task outcomes over fixed self-evolution while supporting diagnosis and validation of improvement-process failures. Notably, in the available Qwen3.8-max archives, fixed self-evolution and the RSI-enabled condition achieve observed success rates of 65.8% and 71.6%, and the historical GPT archive contains 16 fail-to-pass cases versus one pass-to-fail case.

2 Related Work

LLM-based Agents. LLMs have increasingly been used as agents to solve complex tasks through reasoning, planning, tool use, and environment interaction. One line of work focused on improving agent reasoning and planning. For example, language models decomposed high-level instructions into executable plans, while ReAct further interleaved reasoning traces with external actions (Huang et al., 2022; Yao et al., 2023). Another line of work equipped LLMs with external tools and models. Toolformer learned when and how to invoke external APIs, while HuggingGPT used an LLM as a controller to coordinate multiple specialized models (Schick et al., 2023; Shen et al., 2023). Recent studies further investigated more complex tool-use and embodied settings, where agents needed to plan, interact with the environment, and adapt their actions according to intermediate observations (Huang et al., 2023). Beyond task execution, another important direction is to improve agents through accumulated interaction experience. For example, Reflexion stored verbal feedback from previous attempts for subsequent decisions, while ExpeL extracted reusable insights from successful and failed trajectories (Shinn et al., 2023; Zhao et al., 2023). Generative Agents maintained and retrieved past experiences to support reflection and planning, while MemGPT introduced hierarchical memory management to support long-term interactions beyond the context window (Park et al., 2023; Packer et al., 2023). These studies demonstrated that LLM agents can benefit from reasoning, external tools, and accumulated experience.

Agent Self-Improvement. Self-improvement has become an increasingly important direction for enhancing the capabilities of LLM agents. Existing studies can be broadly grouped into three categories. The first focused on response-level refinement. Self-Refine iteratively revised model outputs using self-generated feedback, while Reflexion leveraged feedback from previous trials to improve subsequent attempts (Madaan et al., 2023; Shinn et al., 2023). The second focused on converting

interaction experience into reusable capabilities. Voyager accumulated executable skills through environment exploration, while SkillWeaver and SkillRL further constructed, refined, and evolved reusable skill libraries for future tasks (Wang et al., 2023; Zheng et al., 2025; Xia et al., 2026). The third moved beyond individual skills and optimized the agent system itself. ADAS and AFlow searched for improved agent designs or workflows, while STOP and the Darwin G"odel Machine explored self-improvement through program or code modification (Hu et al., 2025; Zhang et al., 2025b; Zelikman et al., 2023; Zhang et al., 2025a). These studies substantially expanded the scope of agent self-improvement, from responses and skills to workflows and agent programs. However, the strategy for diagnosing failures and selecting appropriate improvement actions is typically predefined by the underlying refinement or evolution procedure. In contrast, our work makes the improvement process itself learnable. The agent learns to select improvement skills from feedback and further refines these skills based on their observed outcomes, enabling the self-improvement mechanism to evolve from previous improvement experience.

3 PRELIMINARIES

Agent Self-Improvement. We first introduce the basic formulation of self-improvement for LLM agents. Given a task instance x, an agent A interacts with an environment and produces an execution trajectory τ and an output o. The environment or verifier then provides feedback f according to the execution result:

$$(\tau_t, o_t) = A(x; S_t) \quad (1)$$

$$f_t = V(x, \tau_t, o_t) \quad (2)$$

where S_t denotes the current agent capability, such as skills, memory, prompts, or other reusable instructions. The feedback f_t may contain task rewards, verifier results, execution errors, or textual feedback. A self-improvement method uses this feedback to update the agent capability:

$$S_{t+1} = \mathcal{I}(S_t, \tau_t, f_t) \quad (3)$$

where $\mathcal{I}$ denotes an improvement procedure. For example, $\mathcal{I}$ can revise an existing skill, add new memory, provide execution guidance, or construct a new skill. The updated capability S_{t+1} is then used in subsequent task execution. In this way, an LLM agent can improve its behavior without necessarily updating its model parameters.

Recursive Self-Improvement. Standard self-improvement mainly focuses on updating the capabilities used for task execution. However, the improvement procedure $\mathcal{I}$ itself is usually predefined and remains unchanged during the improvement process. RSI extends this process by allowing the mechanism that performs improvement to also be modified (Zelikman et al., 2023; Zhang et al., 2025a). We can write this process as

$$S_{t+1} = \mathcal{I}_t(S_t, \tau_t, f_t) \quad (4)$$

$$\mathcal{I}_{t+1} = \mathcal{M}(\mathcal{I}_t, H_t) \quad (5)$$

where $\mathcal{I}_t$ denotes the improvement mechanism at step t, H_t denotes previous improvement experience, and $\mathcal{M}$ denotes a meta-level update process. Unlike standard self-improvement, RSI updates not only the task capability S_t, but also the mechanism that governs how future improvements are performed. This naturally leads to two levels of evolution. The first level improves the task-solving capability of the Answer Agent, such as by revising skills or updating memory. The second level improves the self-improvement process itself, including how failures are diagnosed, which improvement actions are selected, and how improvement skills are executed. Our work focuses on making this second-level process learnable from previous improvement experience.

4 LEARNABLE RECURSIVE SELF-IMPROVEMENT

Figure 1 illustrates the overall framework of LRSI. The method addresses a specific ambiguity in agent self-improvement: after a failed attempt, the system must decide whether to guide the next execution, update reusable task knowledge, or revise the improvement procedure itself. Treating every failure as a skill defect creates unnecessary and sometimes harmful edits, while treating every failure as a one-off execution error prevents the agent from acquiring missing capabilities.

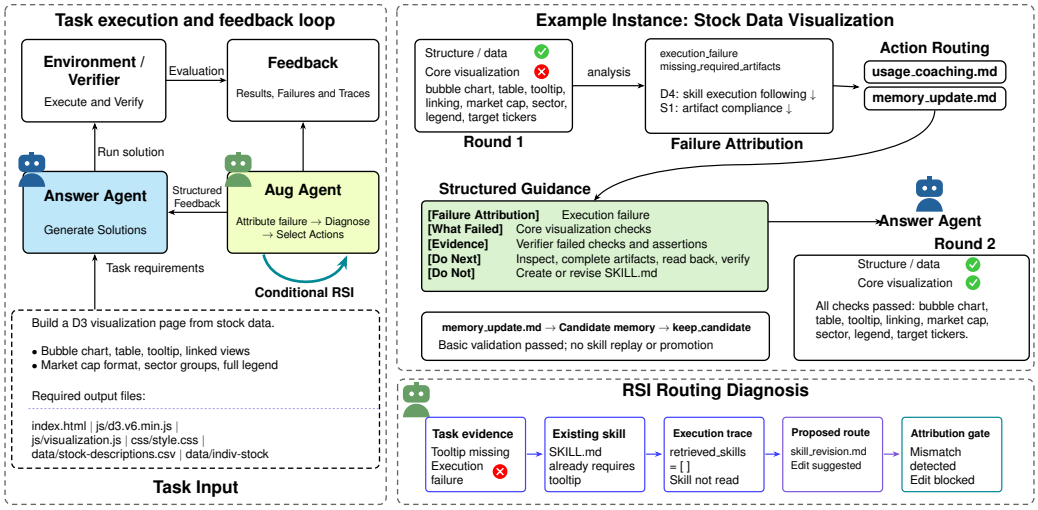

Figure 1: Overview of LRSI on a representative stock-data visualization task. The left panel shows the task execution and feedback loop: the Answer Agent produces artifacts, the verifier returns structured feedback, and the Improvement Agent attributes failures before selecting an intervention. The top-right panel illustrates a failed first round in which missing visualization artifacts are routed to execution guidance and memory rather than immediate skill rewriting. The bottom-right panel shows the recursive diagnosis layer, which compares task evidence, existing skills, execution traces, and proposed routes to block unsupported edits and expose improvement-policy errors.

LRSI resolves this ambiguity with two coupled improvement processes. The inner process improves the Answer Agent's task execution by diagnosing the failure and selecting an appropriate improvement skill. The outer process improves the improvement mechanism by recording which routing decisions helped or failed. Formally, given evidence E_t collected from the current attempt, the Improvement Agent selects a route r_t from the current improvement-skill library U_t and applies the corresponding improvement skill u_{r_t}:

$$r_t = \pi_{\text{imp}}(E_t; U_t) \tag{6}$$

$$b_{t+1} = u_{r_t}(E_t) \tag{7}$$

Here π_{imp} is the learnable improvement policy, and b_{t+1} is the intervention passed to the next Answer Agent attempt. The intervention may be a coaching brief, a memory update, or a proposal for a persistent task-skill change.

4.1 Learnable Improvement Policy

LRSI first turns a raw task failure into a routing decision. This subsection describes the inner-loop policy: how the Improvement Agent reads evidence, chooses among improvement skills, and converts the choice into guidance for the next attempt. The goal is not simply to generate more feedback, but to choose the level at which feedback should operate.

Feedback diagnosis. A failed attempt only indicates that the current solution is incorrect; it does not identify the right intervention. The same missing artifact can arise because no relevant skill exists, because the agent failed to retrieve an available skill, or because it retrieved the skill but did not execute one of its required checks. LRSI therefore asks the Improvement Agent to diagnose the failure source before selecting an action. The diagnosis uses verifier results, execution traces, available-skill evidence, and prior feedback to distinguish reusable skill defects.

Improvement skill selection. Instead of applying one fixed refinement rule, LRSI represents improvement operations as reusable improvement skills. The current action set includes usage coaching, memory update, skill revision, skill decomposition, skill composition, and skill acquisition.

This makes the routing decision explicit: an execution lapse can receive a concrete next-run checklist, while a missing reusable capability can trigger candidate skill construction. For example, in the stock-data visualization case, a missing tooltip should not automatically produce a new visualization skill if the existing skill already requires tooltips but the trace shows that the skill was not read. The appropriate route is usage coaching, possibly with a memory entry, because the evidence points to execution or retrieval rather than a defective procedure. At a high level, the route separates three intervention levels:

$$r_t \in \begin{cases} \mathcal{R}_{\text{exec}} & \text{if evidence indicates an execution or retrieval lapse} \\ \mathcal{R}_{\text{task}} & \text{if evidence indicates a reusable task-skill defect or gap} \\ \mathcal{R}_{\text{meta}} & \text{if evidence indicates a failure of the improvement procedure} \end{cases} \tag{8}$$

Here $\mathcal{R}_{\text{exec}}$ contains routes such as usage coaching, $\mathcal{R}_{\text{task}}$ contains persistent task-skill updates such as revision or acquisition, and $\mathcal{R}_{\text{meta}}$ contains changes to the improver. This notation clarifies that a low verifier score does not by itself determine the update target; the evidence must support the level of intervention.

Structured intervention. The selected improvement skill converts the diagnosis into an actionable intervention for the Answer Agent. The intervention states the failure attribution, supporting evidence, recommended actions, verification steps, and constraints for the next attempt. For execution-level failures, this intervention directly guides the next rollout without changing the task-skill library. For reusable capability gaps, it can initiate a persistent update, but only through the validation path below.

4.2 Improvement Validation

The routing policy can propose persistent changes, but those changes should not be trusted merely because they are plausible. This subsection describes the gate between a proposed update and the active task-skill library. The core idea is to keep construction and adoption separate: a candidate can be written from failure evidence, but it must pass structural checks and replay before it changes future executions.

Candidate construction. Persistent updates are not applied directly to the active task-skill library. When the diagnosis indicates a reusable skill gap or defect, LRSI first constructs a candidate update from the selected improvement skill and the supporting evidence. The original skill remains available during candidate generation. This separation prevents noisy feedback from being accumulated as durable knowledge before it is tested.

Replay and promotion. Each candidate is validated before promotion. Structural checks reject empty, instance-specific, or non-actionable updates. Replay then evaluates the candidate under the same task setting. A candidate is retained only when it passes these checks and improves the verified outcome without introducing an observed regression. Otherwise, the original skill is preserved. Thus, an LLM recommendation is treated as a proposal, not as sufficient evidence for modifying persistent task knowledge.

Let q_t denote whether the candidate passes structural checks, v_t^{base} the verifier result before the candidate, v_t^{cand} the verifier result under replay, and ρ_t whether a regression is observed on protected checks. Promotion is allowed only when

$$p_t = q_t \wedge (v_t^{\text{cand}} > v_t^{\text{base}}) \wedge \neg \rho_t \tag{9}$$

$$S_{t+1} = \begin{cases} S_t \oplus \Delta S_t & \text{if } p_t \\ S_t & \text{otherwise} \end{cases} \tag{10}$$

For instance, if a candidate skill adds a missing export step and replay changes the verifier result from failure to success, it can be promoted. If replay still fails, or if the candidate hard-codes the development instance, the original skill library is kept.

4.3 Recursive Improvement of the Improver

LRSI also treats the improvement policy as something that can fail. A wrong route, such as editing a skill when the trace shows that the skill was never read, is not only a task-level failure; it is evidence

about the improver. This subsection describes how such evidence is recorded and used to propose changes to the improvement skills themselves.

Improvement experience. Each improvement attempt provides not only task experience, but also evidence about the improvement process itself. We represent an improvement record as

$$h_t \quad = \quad (E_t, r_t, b_{t+1}, o_t) \tag{11}$$

where o_t records the task outcome, validation result, and any replay or regression signal. These records make routing errors observable. For example, if the system proposes a skill revision even though the trace shows that the relevant skill was never read, the record exposes a mismatch between the evidence and the selected route.

Improvement skill evolution. LRSI uses accumulated improvement records (denoted as H_t) to revise the improvement skills themselves. The outer updater proposes a change ΔU_t to the current improvement-skill library:

$$\Delta U_t \quad = \quad \mathcal{M}(U_t, H_t) \tag{12}$$

where $\mathcal{M}$ denotes the meta-level update procedure. The proposed change can adjust the routing policy, strengthen evidence requirements, or refine an improvement skill's instructions. In this way, the mechanism used to improve the Answer Agent becomes an object of improvement itself.

As a concrete example, repeated records in which skill revision was proposed despite absent skill retrieval can support a meta-level change that raises the evidence threshold for revision and prefers usage coaching when skill access is not observed.

Regression-based validation. An updated improvement skill is accepted only after independent validation. The candidate is compared with its parent version on development cases under the same task and computation budget. The update is accepted only when it improves downstream outcomes while preserving previously correct improvement decisions:

$$U_{t+1} \quad = \quad \begin{cases} U_t \oplus \Delta U_t & \text{if validation succeeds} \\ U_t & \text{otherwise} \end{cases} \tag{13}$$

This validation prevents the meta-level update from repairing a small set of failures at the cost of degrading the overall improvement process.

5 Experiments

We evaluated whether LRSI supports evidence-routed self-improvement on executable agent tasks.

5.1 Setups

We conducted experiments on SkillLearnBench, a benchmark for agent skill generation on real-world executable tasks (Zhong et al., 2026). The benchmark contains 100 task instances across 20 task families, evaluated by task-specific artifact verifiers. The main instance-level analysis used Qwen3.8-max archives, and a historical GPT-5.6-luna archive provided a paired static-versus-evolution view. Additional available records for Qwen3-coder-plus, DeepSeek-v4-flash, and GPT-5.6-luna are reported in Table 7. More details about task grouping, record selection, and execution units can be found in Appendix C.

5.2 Baselines

We compared LRSI-related runs against three practical baselines. The no-skill condition solved tasks without human-authored task skills. The static-skill condition used the available task skills but did not perform autonomous self-evolution. The fixed self-evolution condition received feedback and performed task-level improvement, but did not include the RSI-enabled outer diagnosis and parser-management components. The RSI-enabled condition added the framework-level diagnosis described in Section 4; it should be interpreted as an RSI-enabled framework rather than as evidence that every task benefited from an accepted autonomous policy update.

Evaluation scope	Inst.	Pass	Succ.(%)
Recorded Qwen3.8-max	92	50	54.35
Full manifest	100	50	50.00

Table 1: Static-skill outcomes and coverage.

Condition	Inst.	Pass	Succ.(%)
Static skill	45	20	44.44
Evolution archive	45	35	77.78

Table 2: GPT outcome comparison.

5.3 Evaluation Methods

We reported task success, failure-to-success recovery, intervention correctness, and planned matched-budget efficiency. In the result tables, "Inst." denotes the number of evaluated instances, "Pass" denotes verified successes, and "Succ." denotes the success rate in percent. Higher task success indicates that more produced artifacts passed their task verifiers. Higher recovery indicates that more final successes came after an initially failed attempt, showing the effect of iterative improvement. Higher intervention correctness indicates that routing and candidate updates were better supported by evidence and replay; lower cost per success is better under a fixed-budget comparison. Metric formulas, latest-record selection, and missing-record handling are provided in Appendix C.

5.4 Results

Static-skill coverage. Table 1 separates full-manifest coverage from observed execution performance. Static skills solve 50 of the 92 recorded Qwen3.8-max instances, giving an observed success rate of 54.35%; the conservative score over all 100 benchmark instances is 50.00%. Recorded successes span document generation, planning, and numerical analysis. Court-form and stock-visualization cases remain unresolved in this static archive (Figure 5). The contrast identifies artifact production and verification as concrete recovery targets: a failed form or visualization requires check-level diagnosis before deciding whether execution guidance or a skill edit is appropriate. These baseline outcomes motivate that distinction without by themselves measuring the benefit of selective evolution.

Iterative recovery. Figure 3 shows where iterative execution changes an outcome: six fixed-strategy cases and ten RSI-enabled cases pass after initially failing. The later-pass segments establish within-instance recovery; the unresolved segments show that repeated execution does not resolve every failure.

Cross-model outcomes. The GPT archive supplies a complementary paired view of 45 benchmark instances across nine task categories. Table 2 reports 20 static-skill successes and 35 instances with at least one successful evolution record. The paired transitions comprise 16 fail-to-pass cases, one pass-to-fail case, 19 cases successful in both conditions, and nine unsuccessful in both (Figure 2). The 16-to-one asymmetry shows that improvements outnumber reversals in this paired archive. Repeated opportunities in the evolution arm nevertheless prevent attributing the difference to RSI alone; source availability is documented in Appendix C.

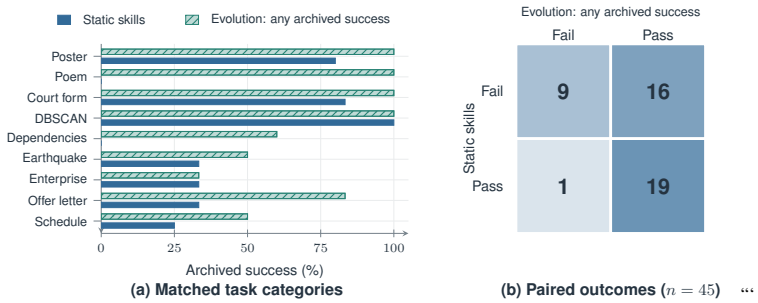

Figure 2: GPT outcomes across nine task families and 45 paired instances. Left: per-family rates on matched identifiers. Right: paired outcome counts. The evolution arm pools repeated archival opportunities; the bars and transitions do not isolate the effect of an improvement mechanism.

Iterative execution resolves initially failed cases. In the earlier snapshot, six fixed-strategy instances and ten RSI-enabled instances pass after an initial failure. These account for 6/21 (28.6%) and 10/42 (23.8%) of the observed successes, respectively. Figure 3 places those recoveries alongside first-attempt successes and unresolved outcomes.

The chart retains all 32 fixed-strategy and 56 RSI-enabled instances, including two still-running records. It describes recovery within each archive; separating the effects of coaching, skill edits, and additional sampling requires budget-matched ablations. These earlier cohorts differ from the latest 38- and 81-instance snapshots.

Separate latest snapshots contain 25/38 fixed-strategy and 58/81 RSI-enabled successes (65.8% and 71.6%). Their instance sets and runtime histories differ from each other and from the earlier snapshot in Figure 3. Together, the exports document coverage and verified recovery, without isolating the effect of outer-loop learning.

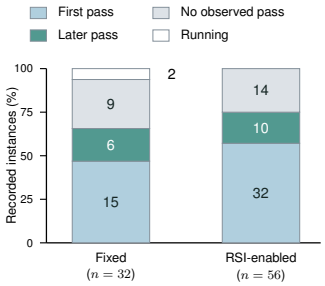

Figure 3: Earlier Qwen outcomes. Bar segments show shares of recorded instances; labels give instance counts. Unresolved records are not verified budget-exhausted failures.

Update correctness.

5.5 Intervention diagnostics and controlled update validation

The outer layer makes improvement errors explicit evaluation targets. An eight-case event audit exercises the actual meta-diagnostic functions: five inappropriate-update or replay-failure scenarios trigger a diagnostic signal, and two valid control scenarios produce none. One deliberately suppressed acquisition, despite evidence of a genuine skill gap, is missed. The resulting seven-of-eight agreement is a deterministic branch audit, not a population detection rate. It identifies a concrete requirement for improvement-policy evaluation: test both unjustified intervention and failure to perform a justified intervention.

An isolated PDF-form experiment then connects update decisions to artifact correctness. Omitting checkbox and radio assignments leaves the original binary reward unchanged at 1, but changes the underlying checks from 47 passes to 35 passes and 12 skips. Strict field and appearance-state checks distinguish the defective helper (one of six synthetic fixtures passed) from the intact helper (six of six). The proposed promotion lacks replay evidence, so the RSI detector emits `promotion_without_replay`. A hand-written quarantine adapter blocks that promotion and retains the intact helper, exactly as a fixed mandatory-replay gate does. The object being corrected is the update-selection decision, not the PDF task itself. Figure 4(a) exposes the checks hidden by binary reward; panel (b) shows the artifact consequence of selecting each helper. This is a controlled decision replay using measured helper outputs, not an additional set of agent runs.

From diagnostics to a testable repair process. Evidence-bound revision, action constraints, memory retrieval, and accepted snapshots support the repair protocol in Section 4.2: freeze a parent, generate a constrained candidate, validate and replay, then test an accepted version independently. The PDF pilot demonstrates the consequence of update selection under explicit controls. Autonomous policy-repair efficacy requires a model-generated accepted revision; two earlier generation calls produced no usable candidate (Appendix D).

Planned controlled comparison. The detailed rate table in Appendix C reveals model- and task-dependent outcomes. For Qwen3.8-max, the available static, fixed-evolution, and RSI-enabled cohorts reach 54.3%, 65.8%, and 71.6% success. Because cohort coverage differs, causal comparison requires a matched-budget study that isolates attribution, skill updates, replay, and outer updates on held-out families.

Appendix C specifies the intervention records, resource accounting, and component-replacement protocol needed to separate improved strategies from retained task solutions or extra execution attempts.

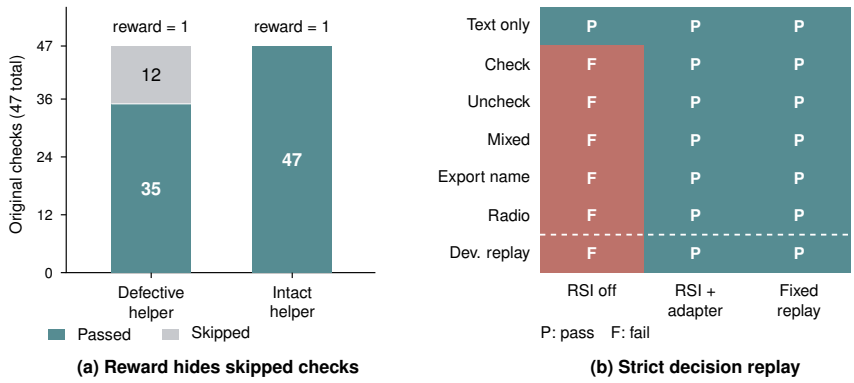

Figure 4: Update validation links diagnostic evidence to artifact outcomes. (a) Binary reward remains 1 for both helpers, while detailed checks expose 12 skipped checks for the defective helper. (b) The defective helper passes one of six strict synthetic fixtures; the intact helper passes all six. The dashed line separates the fixtures from a reused development case. RSI with a hand-written quarantine adapter and a fixed replay gate select the same intact reference helper, so their columns reuse its deterministic outcomes. The figure evaluates controlled update selection; it contains no generated candidate or independent stochastic trial repetitions.

6 Conclusion

We introduced LRSI, a framework for learnable recursive self-improvement in LLM agents. LRSI separates task-level recovery from improvements to the improvement process itself, using verifier evidence, failure attribution, candidate replay, and regression checks to decide which interventions should persist. Across archival SkillLearnBench results and controlled update cases, iterative improvement recovers failed tasks, while validation helps prevent unsupported updates. Future matched-budget, held-out evaluations should test whether accepted changes to the improver produce better subsequent interventions after task state is reset.

The central design principle is to distinguish a failure of task execution from a failure of the improvement decision. A missing artifact or an unmet visualization requirement does not, by itself, establish that a task skill needs revision. Diagnosis must also consider whether the relevant skill was retrieved and followed. This distinction connects the two levels of the framework: task-level guidance addresses the immediate failure, while framework-level validation checks whether the proposed intervention is appropriate before allowing it to persist.

The empirical findings support complementary conclusions. Archived trajectories demonstrate that iterative execution can recover initially unsuccessful tasks, and the controlled cases show why outcome-level scores should be accompanied by checks of intervention correctness. These forms of evidence motivate selective, validated updates, while leaving the causal contribution of learned improvement policies to be established under matched conditions. A decisive next evaluation should jointly measure recovery, retained performance, and execution cost, with held-out tasks and explicit controls for retries. This would test whether improvements to the decision process transfer beyond the cases that triggered them.

AI USE STATEMENT

Generative AI assisted with manuscript organization and drafting, literature discovery, source-code inspection, data-audit scripts, figure code, and experimental-design suggestions. Language models also serve as the Answer and Improvement Agents described in the method. The numeric figures here are calculated from archived records rather than generated measurements. Automated consistency checks and rendered-page inspection were used during preparation. Human authors must verify the scientific interpretation, citations, and all AI-assisted content before submission and take responsibility for the final manuscript. This disclosure should be updated after that review and to reflect any additional AI use in the completed experiments.

REFERENCES

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/36b7acf6f6010652b3f2a433774a66fe-Abstract-Conference.html.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 9118–9147. PMLR, 2022.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the 6th Conference on Robot Learning*, pp. 1769–1782. PMLR, 2023.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pp. 287–318. PMLR, 2022.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

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

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.

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

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

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

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. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026. URL https://arxiv.org/abs/2602.08234.

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

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023. URL https://arxiv.org/abs/2310.02304.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025a. URL https://arxiv.org/abs/2505.22954.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, XiongHui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5492ecbce4439401798dcd2c90be94cd-Abstract-Conference.html.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023. URL https://arxiv.org/abs/2308.10144.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025. URL https://arxiv.org/abs/2504.07079.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. SkillLearnBench: Benchmarking continual learning methods for agent skill generation on real-world tasks. *arXiv preprint arXiv:2604.20087*, 2026. URL https://arxiv.org/abs/2604.20087.

Dimension	Evidence of a defect	Appropriate response
D1: skill quality	Reusable procedure absent or incorrect	Revise; acquire only a missing sub-capability
D2: agent–skill fit	Trace links confusion to procedural complexity	Decompose or clarify supported steps
D3: invocation	Relevant skill available but not adopted	Invocation coaching and scoped memory
D4: following	Invoked procedure skipped; output contract unmet	Execution feedback; no skill creation by default
D5: composition	Wrong ordering or producer–consumer handoff	Repair the composition procedure

Table 3: Diagnostic dimensions and admissible responses. Scores below 2 require evidence; task failure alone is insufficient.

A FEEDBACK AND SKILL CONTRACTS

The implementation calls the Improvement Agent the Aug Agent. Its JSON includes core and scenario dimension feedback, a route decision, and an execution brief. The corresponding fields are `core_dimension_feedback`, `scenario_dimension_feedback`, `route_decision`, and `feedback_to_answer_agent`. Each dimension records a score, rationale, evidence, and confidence. Route decisions specify whether an update is needed, primary issue dimensions, capability-gap evidence, and requested improvement skills. A normalized attribution can suppress an inappropriate skill action.

The next Answer Agent receives a rendered brief with these sections: Failure Attribution, What Failed, Evidence, Aug Agent Brief, Do Next, Verify Before Final, and an optional Do Not section. The schema stabilizes the interface without fixing the substantive repair. A formal task skill should instead contain a reusable executable procedure. A raw traceback, instance-specific answer, or generic instruction to retry is not an adequate skill.

Implementation map. Table 4 links the paper's control decisions to repository components. The Qwen fixed-strategy sibling retains the task-level loop but lacks the RSI parser-management package; the RSI-enabled project supplies the outer path. Paths are repository-relative rather than machine-specific.

Mechanism	Implementation surface
Task execution and verifier	`core/eval_runner.py`; immutable task tests
Master decision and D1–D5 diagnosis	`core/aug_agent.py`; `aug_agent_builtin_skills/master_guidance.md` and `feedback_dimension_scoring.md`
Action routing and skill admission	`core/self_evolution.py`; action documents in `aug_agent_builtin_skills/`
Evidence-bound edits and snapshots	`core/skill_lifecycle.py`; parent hashes and verified inventories
Accepted memory retrieval	`core/skill_lifecycle.py`; scoped, deduplicated lessons
Candidate validation and replay	`core/self_evolution.py`; separate candidate and replay records
Online meta-diagnosis	`evaluate_rsi_meta_update` in `core/self_evolution.py`
Parser version pilot	`core/rsi/framework_registry.py` and `framework_evaluator.py`

Table 4: Main mechanism and corresponding implementation surface.

Step	Operation
Task-level self-evolution	
1	Execute the Answer Agent with current skills and feedback; verify artifacts.
2	Preserve failure evidence and classify infrastructure, execution, adoption, or skill defects.
3	Obtain and normalize the Improvement Agent diagnosis; render the next execution brief.
4	If admitted, generate a minimal skill candidate, validate it, and run a separate replay.
5	Record candidate eligibility and the next skill source; retry within the main-loop budget.
Framework adaptation: proposals and a separate parser pilot	
6	In the RSI-enabled implementation, record framework issues and meta-update proposals.
7	Separately freeze a parser parent; isolate a candidate and run regression and paired development checks before registration.

Table 5: Implemented inner loop and separately controlled outer pilot.

B IMPLEMENTATION SCOPE

Diagnostic triggers and parser isolation. The detector covers skill generation after non-skill attribution, suppressed inappropriate actions, an overly broad gate, raw failure dumps, non-actionable candidates, failed replay, and promotion without replay. High- or critical-severity issues can immediately produce proposals. Medium-severity issues are observed first unless earlier triggered observations exist in recent history; recurrence counts prior triggers rather than identical issues. Parser candidates run in a read-only, non-root container without network access, with a 30-second limit, 256 MiB memory, one CPU, and bounded output. These limits apply to parsing, not task execution.

B.1 EXECUTION SEQUENCE

The parser registry versions the function `_read_verifier_output` and supports parent hashes, candidate manifests, regression reports, and paired evaluation records. Its generation budget is two model calls per frozen experiment. The development evaluator bounds configured iterations to 1–3 and requested steps to 1–100, and tracks a worst-case launch budget. A requested step value is not necessarily a hard Codex execution limit: runtime records in the evolution-only export explicitly mark `max_steps_enforced=false`. A controlled study must enforce its resource budget at the execution layer and record actual consumption.

The two current implementations share the conceptual inner loop, but the evolution-only project contains additional Qwen gateway and runtime handling. They are not byte-identical runtimes with an RSI switch. A causal RSI ablation should therefore use one frozen execution stack and vary only the specified improvement component.

C DATA SELECTION AND REPRODUCIBILITY PROTOCOL

Execution-record accounting. The separately frozen Qwen attempt audit contains 139 unique condition–instance–trial records: 56 fixed-strategy and 83 RSI-labelled records, including two running fixed-strategy records and legacy RSI records without explicit terminal status. These are execution records, not 139 independent completed tasks. This audit and later instance-level snapshots are separate exports. Candidate replays are counted separately from main attempts.

Budgeted evaluation definitions. For a model–condition cohort with N recorded instances, the observed success rate is

$$\mathrm{SR}_{\mathrm{obs}} \quad = \quad \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}[y_i = 1] \tag{14}$$

Algorithm: Evidence-routed improvement with guarded candidate use

Require: task x; answer agent A; verifier V; skills S; memory M; improvement skills U; budget B
Ensure: verified output, next-run brief, accepted memory, accepted skill candidates, and meta-update records

def `run_lrsi`(x, A, V, S, M, U, B):
  $\mathcal{H} \leftarrow [\,]$; $\mathcal{C}_{\text{skill}} \leftarrow [\,]$; $\mathcal{C}_{\text{mem}} \leftarrow [\,]$
  for $t = 1$ **to** B **do**
    $\tau_t, o_t \leftarrow A.\texttt{solve}(x, S, M)$
    $f_t \leftarrow V.\texttt{check}(x, \tau_t, o_t)$
    if $f_t.\texttt{pass}$ **then return** $(o_t, S, M, \mathcal{H})$
    $E_t \leftarrow \texttt{collect_evidence}(x, \tau_t, f_t, S, M)$
    $d_t, r_t \leftarrow \texttt{attribute_and_route}(E_t, U)$
    $r_t \leftarrow \texttt{apply_action_constraints}(r_t, E_t)$
    if $r_t = \texttt{usage_coaching}$ **then**
      $M \leftarrow \texttt{keep_candidate}(M, \texttt{make_memory}(E_t, d_t))$
      $b_{t+1} \leftarrow \texttt{render_brief}(d_t, E_t)$
    else if $r_t = \texttt{skill_revision}$ **and** $\texttt{skill_gap}(E_t)$ **then**
      $c_t \leftarrow \texttt{draft_skill_candidate}(E_t, S)$
      $\rho_t \leftarrow \texttt{replay_and_regress}(c_t, S, V)$
      if $\rho_t.\texttt{accepted}$ **then** $S \leftarrow S \oplus c_t$
      else $\mathcal{C}_{\text{skill}}.\texttt{append}(\texttt{blocked}(c_t, \rho_t))$
      $b_{t+1} \leftarrow \texttt{render_brief}(d_t, E_t, \rho_t)$
    else
      $b_{t+1} \leftarrow \texttt{render_brief}(d_t, E_t, \texttt{do_not_edit_skill})$
    end if
    $\mathcal{H}.\texttt{append}(x, \tau_t, f_t, d_t, r_t, b_{t+1})$
    if $\texttt{routing_failure}(E_t, r_t)$ **then**
      $\mathcal{H}.\texttt{append}(\texttt{meta_proposal}(U, E_t, r_t))$
    end if
  end for
  return $(\texttt{unresolved}, S, M, \mathcal{H})$

Target	Implemented operation	Evaluation endpoint
Task execution	Structured diagnosis and execution brief	Recovery versus budget-matched feedback/retry controls
Task skills	Candidate admission, validation, and replay	Independent-instance repair and preservation of intact behavior
Improvement policy	Meta-observation and policy proposals	Accepted policy changes with improved subsequent routing
Evidence interface	Isolated, versioned parser pilot	Paired parent–candidate gains and independent transfer
Recursive adaptation	Explicit parent–candidate acceptance contract	Multiple accepted generations with attributable downstream effects

Table 6: Adaptation targets, implemented operations, and evaluation endpoints.

where y_i is the recorded verifier outcome. For the static Qwen baseline, conservative full-manifest scoring assigns zero to missing records and alternate-model failures:

$$\mathrm{SR}_{100} \quad = \quad \frac{1}{100} \sum_{i=1}^{100} \mathbf{1}[y_i = 1] \tag{15}$$

Failure-to-success recovery measures whether an instance that initially failed later obtained a verified success:

$$\mathrm{RecoveryShare} \quad = \quad \frac{\#\{\text{successful instances with an earlier failed attempt}\}}{\#\{\text{successful instances}\}} \tag{16}$$

Model	Evolution level	Overall	Doc./plan.	Research/ info	Code/sec.	Data/ quant.
Qwen3.8-max						
	No skills	35.0%	37.9%	33.3%	0.0%	40.0%
	Static human-authored skills	54.3%	52.6%	50.0%	60.0%	56.3%
	Fixed self-evolution	65.8%	80.0%	0.0%	12.5%	100.0%
	RSI-enabled	71.6%	80.6%	53.8%	57.1%	78.3%
DeepSeek-v4-flash						
	No skills	19.7%	13.3%	25.0%	16.7%	27.8%
	Static human-authored skills	55.6%	37.5%	58.7%	60.9%	81.8%
	Fixed self-evolution	50.0%	50.0%	50.0%	50.0%	50.0%
	RSI-enabled	92.6%	87.5%	81.8%	86.1%	100.0%
GPT-5.6-luna						
	No skills	50.0%	50.0%	50.0%	50.0%	50.0%
	Static human-authored skills	36.1%	39.4%	33.3%	9.1%	50.0%
	Fixed self-evolution	68.9%	73.1%	33.3%	60.0%	72.7%
	RSI-enabled	77.1%	61.4%	70.9%	50.0%	80.6%

Table 7: Reported success rates across models, evolution settings, and task families. Rates are percentages. Model–setting cohorts differ in coverage and retry budgets; source records and aggregation details are retained with the data export.

For future controlled runs, report success at a fixed total launch budget B and cost per success:

$$\mathrm{SR}(B) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\text{instance } i \text{ succeeds within } B], \qquad \mathrm{CPS} = \frac{\sum_i C_i}{\sum_i y_i} \tag{17}$$

C_i includes Answer Agent, Improvement Agent, replay, and outer-update costs under a declared accounting policy. Token counts and monetary cost should be reported separately; missing usage is not zero. Candidate-learning costs and final-test costs must also be distinguished.

Task categories for Table 7. We group the 20 benchmark task families by their primary output and operation. Document/planning comprises anthropic-poster-design, chinese-poem-generator, court-form-filling, offer-letter-generator, organize-messy-files, schedule-planning, and travel-planning. Research/info comprises enterprise-information-search, financial-analysis, and nlp-paper-reproduction. Code/security comprises dependency-vulnerability-check, fix-security-bug, github-repo-analytics, and python-scala-translation. Data/quantitative comprises dbscan-parameter-tuning, earthquake-plate-calculation, stock-data-visualization, temperature-simulation, weighted-gdp-calculation, and video-object-counting. Each family belongs to exactly one category; denominators include only recorded instances under the corresponding condition. In the order overall, document/planning, research/info, code/security, and data/quantitative, the observed pass/recorded counts underlying the measured Qwen3.8-max entries in Table 7 are: static skills, 50/92, 20/38, 6/12, 6/10, 18/32; fixed-strategy evolution, 25/38, 12/15, 0/3, 1/8, 12/12; and RSI-labelled evolution, 58/81, 25/31, 7/13, 8/14, 18/23. The fixed-strategy 100% data/quantitative cell is based on only twelve recorded instances and must not be extrapolated to the full task category.

Updated author-supplied results. The latest GPT-5.6-luna RSI rates are supplied by the author and recorded as a separate update within the rate export. They replace the older aggregate row for presentation; the underlying archived instance records are retained unchanged. Success counts, denominators, and per-instance outcomes for this update have not yet been supplied, so they are not reconstructed from rounded percentages. These results are excluded from count-based pooling and paired uncertainty calculations until their associated records are available.

Updated evolution snapshots. The audit deduplicates by condition, model, and instance ID, rejecting inconsistent copied outcomes. For Table 7, it selects the latest terminal record for each model–condition–instance tuple using the timestamp in the trial identifier, with filesystem modification time used only as a tie-breaker. Running records are not counted as terminal failures. Both

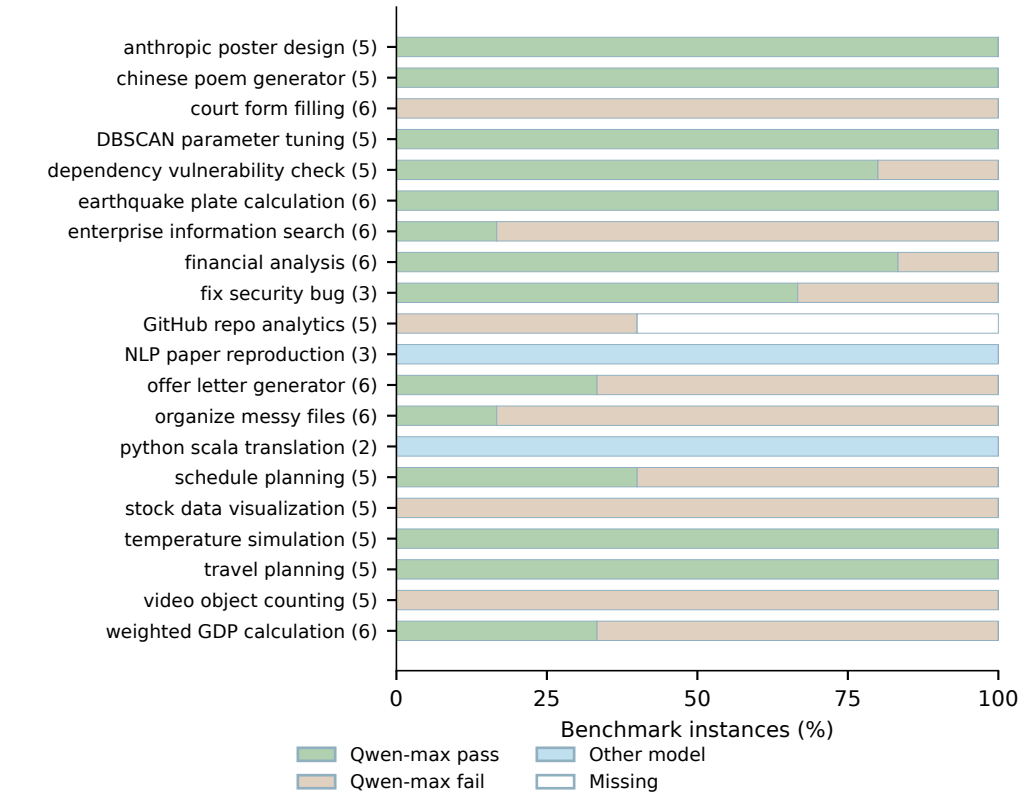

Figure 5: Coverage and outcomes behind the static-skill baseline. All 100 manifest instances are retained: 50 Qwen3.8-max passes, 42 Qwen3.8-max failures, five alternate-model failures, and three missing records. Alternate-model and missing records receive zero under the conservative scoring in Table 1, while remaining separate evidence categories. The plot describes one archival snapshot; no repeated-run uncertainty is estimated.

source and reward hashes are retained where available; source histories are preserved in the attempt-level CSV. Under this latest-record policy, the Qwen3.8-max fixed-strategy row covers 38 instances with 25 observed passes, while the RSI-labelled row covers 81 instances with 58 observed passes. Summary-only records without a corresponding result file are excluded. Some legacy RSI records lack an explicit terminal status field, so unresolved or incomplete runs remain separated from observed failures when the necessary status evidence is unavailable.

Static Qwen records. The audit reads each `result.json` under the selected static-skill export, requires a Boolean outcome, and rejects duplicate or unknown instance IDs. The standard manifest has 100 unique IDs. Each exported row includes the recorded alias, outcome, trial identifier, selected runtime flags, reward-file value, and source hash. The three missing records are GitHub analytics instances 1, 3, and 4. The five alternate-alias records are NLP reproduction instances 1–3 and Python–Scala translation instances 1–2. Table 1 assigns zero to these eight instances, giving 50 successes and 50 scored failures over 100 instances. A separate scoring CSV preserves the original model and distinguishes observed Qwen-max failures from alternate-model and missing-record assignments. The observed max-alias rate remains 50/92 (54.35%); the conservative 50/100 score must not be described as 100 completed runs of that model. Figure 5 preserves the original evidence categories for transparency.

Historical GPT records. The paired table is imported from a previously prepared instance-level CSV, preserving its any-success aggregation. Table 8 gives its counts. Its reported earliest successful evolution iteration is 1 for 27 instances, 2 for seven, and 3 for one. These are pooled archival labels, not a success-versus-budget curve from a single run. Of 222 indexed source records, 173 remain readable and all record `gpt-5.6-luna`; 49 indexed paths are unavailable. The comparison is therefore secondary archival evidence rather than a fully revalidated controlled experiment.

Family	Inst.	Static	Evol.
Poster	5	4	5
Poem	5	0	5
Court form	6	5	6
DBSCAN	5	5	5
Dependencies	5	0	3
Earthquake	6	2	3
Enterprise	3	1	1
Offer letter	6	2	5
Schedule	4	1	2
Total	**45**	**20**	**35**

Table 8: Historical matched-instance counts.

Future experiment manifest. For every arm, record a run ID, task split, model alias and provider-resolved version if available, decoding settings, code commit, image digest, initial skill hashes, improvement-skill hashes, parser version, time limits, and launch budget. For every attempt, retain result and verifier files, the execution trace, timestamps, token usage by role, and retry reason. For every candidate, retain parent and child hashes, the exact diff, supporting evidence, regression and replay results, and activation time. Resource costs of failed candidates must remain in the denominator of efficiency accounting.

Independent verification. Keep the artifact verifier fixed, and separate development feedback from final held-out evaluation. If an implementation defect in a verifier is corrected, version the correction and rerun all affected comparison arms. Never weaken acceptance conditions selectively. Post-hoc manual repairs are a separate assisted condition. Regression cases may accumulate from development failures, but final test cases must not enter that suite before the frozen comparison.

Required claims and measurements. The full study needs a controlled 100-instance comparison, cost accounting, and unseen-group component replacement. Candidate counts alone are not an efficacy metric. Report accepted and rejected updates, false skill-admission decisions, per-iteration gains, same-instance replay success, and held-out transfer separately. To support recursive improvement, preserve at least two parent–child generations and show that later interventions were produced by the accepted improved procedure rather than by a fixed generator with accumulated task answers.

D CONTROLLED UPDATE SAFETY AND VERIFIER AUDIT

Selection and intervention. This exploratory pilot was selected after an earlier ceiling result on `court-form-filling-3`: the Answer Agent had used a defective PDF helper, written a separate checkbox repair, and ultimately passed all 47 original checks. We reused that trajectory as development evidence, not as held-out evaluation. A private copy of the helper omits checkbox and radio assignments while retaining text writes. Reconstructing the recorded field mapping on the saved blank PDF isolates this defect from subsequent Answer Agent repairs. Expert and defective helpers use the same invocation interface and inputs; the expert implementation is a reference, not a generated skill.

The corresponding decision and outcome comparison appears in Figure 4.

Original verifier and strict checks. We leave the benchmark's `test.sh` and acceptance criterion unchanged. Its checkbox tests skip when checkbox data cannot be extracted, producing 35 passes and 12 skips for the defective output, yet reward 1; the intact output produces 47 passes and no skips. The raw pytest log supplies these counts because the CTRF export groups parameterized checks into five entries. This finding does not retrospectively re-score other archived runs. It motivates reporting skip counts alongside reward and versioning any future verifier correction for all comparison arms.

Separately, six synthetic AcroForm fixtures cover text-only input, checking, unchecking, mixed states, a nonstandard checkbox export name, and radio selection. The deterministic evaluator compares requested field values, untouched fields, and widget appearance states. The same inputs are used for every helper. The defective helper passes only the text control; the intact helper passes all six. A seventh evaluation reuses the development mapping and is reported separately. These are diagnostic fixtures, not seven independent benchmark tasks; no confidence interval or population-level success estimate is inferred.

RSI signal and deployment-decision replay. We inject one proposal with the known defective helper and truthful metadata indicating no validation replay, while an intentionally weakened promotion path labels it ready. The original RSI function emits `promotion_without_replay`. An experimental, hand-written adapter quarantines a flagged proposal and retains the intact helper. We replay each branch by selecting the already measured deterministic outputs of its chosen helper, rather than claiming additional agent runs or a live deployment. RSI-off selects the defective helper; RSI with the adapter and a fixed mandatory-replay gate both select the intact helper. A single healthy proposal with genuine passing replay evidence elicits no RSI flag. The evidence establishes conditional prevention, not autonomous policy modification, broad false-positive calibration, or an advantage over ordinary validation.

Stage	Observation	Supported interpretation
Bad proposal	RSI flags promotion without replay	Detects the injected promotion defect
Healthy control	No flag with a genuine passing replay	One negative control, not a calibrated error rate
Outcome selection	Quarantine and fixed replay retain the intact helper	Conditional protection; no advantage over the fixed gate
Skill generation	Two calls, no usable candidate; retry times out	Utility unmeasured, not zero accuracy
Full-task transfer	Planned paired runs not executed	No transfer or efficiency claim

Table 9: Evidence boundaries of the isolated pilot. A completed detection or replay is not an accepted autonomous framework update.

Generation outcome and protocol deviations. An explicitly separate offline consolidation adapter asks the recorded `qwen3.8-max` model to turn the successful episode's transient repair into a reusable skill. It receives the development trace and defective implementation, but not expert code or held-out inputs. The first call returns no usable text; the sole retry times out at 540 seconds with only startup events. No candidate is substituted by hand or scored. Planned full-task transfer

runs are suspended after discovering the verifier blind spot. Therefore neither generated-skill transfer nor token savings is established. These calls also do not validate ordinary Aug routing after a failed task, since the original development episode ultimately succeeded.

Isolation and reproducibility. All work runs in a new workspace with copied sources and skills. Model calls are sequential; worker containers are limited to 0.5 CPU and 1 GiB. All 56 monitored production source, task, and skill files retain their initial hashes. A dependency-installation timeout and an evaluator error resolving a PDF indirect object are retained as infrastructure/evaluator failures, not scored task failures. Corrected evaluator results, raw test logs, original RSI decisions, source hashes, and the plot's derived data are retained. The numeric figure requires no API access to reproduce.