SELF-REGRESSION-TEST: PERSISTENT BEHAVIORAL CONTRACTS FOR SKILL-EVOLVING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving agents adapt to new tasks by rewriting reusable skills, and every such rewrite is a potential regression: a change that improves one use can silently break behaviors that other tasks still depend on. Aggregate safeguards—parameter regularization, replay, or diff-magnitude penalties—leave the specific behaviors each candidate edit puts at risk unspecified. We study capability preservation at the level of the skill artifact. Each skill carries a versioned regression suite whose predicates are deterministically executable on cached inputs and act as a test-backed, coverage-conditioned partial contract for future edits. Self-Regression-Test (SRT) realizes this contract in three steps. First, a synthesizer converts a skill's past successful and failed trajectories into a fixed suite of typed predicates: outcome, trace, and negative, each carrying a severity label. Semantic outcome checks use a five-vote LLM-judge ensemble whose fallibility is measured on an independent audit rather than assumed away. Second, an executor evaluates every candidate update against the current suite and returns predicate-level verdicts. Third, a controller turns those verdicts into one of three inspectable actions—commit, bounded repair, or rollback. SRT never touches the policy weights or the upstream evolution operator, and refreshes the suite periodically as usage drifts. Across four evolution operators and three lifelong benchmarks, SRT raises retained accuracy by 8.3–11.4 points at a 0.8–1.5-point cost in adaptation accuracy, at approximately 14% verifier overhead. A six-cell cross-matrix attribution against replay, one-shot Self-Verify, and historical failure-set rerun shows the gain is not explained by rejection alone or extra computation, and an independent 600-update mutation audit reaches F1 0.90 with 90.3% hidden-paraphrase pass (71.0% for snapshot exact-match). The guarantee is scoped: it certifies preserved behavior only on inputs whose predicates deterministically cover, LLM-judge errors are audited empirically, and per-skill contracts do not compose across simultaneous multi-skill refactorings.

1 INTRODUCTION

Self-evolving LLM agents adapt to new tasks by modifying the resources they use. A skill library makes this adaptation explicit: descriptions, instructions, tool-call recipes, and helper code can be revised without updating the policy model. Recent systems optimize these artifacts through textual feedback (Zhang et al., 2026a), reinforcement learning (Wang et al., 2025a), and collective experience (Ouyang et al., 2026; Wu et al., 2026). The resulting library must incorporate new experience while retaining behaviors on which earlier tasks still depend.

These two objectives—incorporating new experience and retaining prior behaviors—can conflict at the level of a single edit. A more specific retrieval trigger may suit the current task but exclude an older use. Replacing a tool can preserve the appearance of an answer while changing units or dropping a required preprocessing step. Such regressions contribute to *capability erosion*, the degradation of old-task performance during continued evolution (Yu et al., 2026). The difficulty is that a candidate's success on its motivating task does not specify which other behaviors it must preserve.

Two complementary lines defend against erosion. Parameter-level continual learning regularizes weight changes or rehearses old data (Kirkpatrick et al., 2017; Shin et al., 2017; Huang et al., 2024), while the skill-channel CPE regularizer constrains the rate at which artifacts drift (Yu et al., 2026).

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.

These methods reduce forgetting in aggregate but act on smoothed statistics rather than on the specific behavior each candidate edit puts at risk. A separate line, behavioral testing (Ribeiro et al., 2020), gives a concrete decision object—an assertion—but has been developed for static systems, not versioned artifacts that evolve alongside a policy. What has not been made explicit is a mechanism that attaches persistent tests to each skill and uses their failures to govern subsequent edits.

Can behavioral tests synthesized once from a skill's history, reused across every future edit, admit useful updates while blocking regressions?

We answer this question with Self-Regression-Test (SRT). SRT pairs every skill with a regression suite of outcome, trace, and negative assertions synthesized from past successful and failed trajectories (Figure 1). When the upstream operator proposes a candidate update, SRT executes the suite: passing candidates are committed, severe failures are rolled back, and recoverable failures return targeted feedback that drives a bounded repair loop. Search for useful edits and admission to the library thus become separate, inspectable operations, and the same suite carries into future refresh cycles.

Contributions.

- A training-free admission controller that turns each skill's history into a persistent, skill-specific, actionable behavioral contract, and separates upstream proposal from downstream admission.
- A coverage-conditioned bound relating average erosion under an accepted update to the fraction of old tasks the suite certifies.
- An evaluation across four operators, three lifelong benchmarks, and 600 independently labeled updates that links aggregate gains to hidden-input generalization, decision reliability, and maintenance cost.

2 Related Work

Behavioral tests and self-verification. CheckList (Ribeiro et al., 2020) organized NLP evaluation around behavioral capabilities and test templates, and inspired a family of tools that use large language models to synthesize tests. TestPilot (Dakhel et al., 2024) generates unit tests with pretrained LLMs and mutation-based selection. ASTER (Pan et al., 2025) scales this idea to natural- and multi-language unit tests. CoverUp (Pizzorno & Berger, 2025) pursues high-coverage Python test generation. Prasad et al. (2025) learn to generate unit tests for automated debugging. In parallel, studies of test smells in LLM-generated tests (Ouédraogo et al., 2024) and of LLM judges (Gu et al., 2024; Thakur et al., 2024; Zhuge et al., 2025) document the limitations of generated assertions and semantic evaluation. These lines target either static software artifacts or fixed evaluation harnesses, and neither versions the tests alongside the code they cover nor uses their failures to gate future edits. Two ingredients distinguish SRT from this line. Assertions are versioned alongside the skill they cover and refreshed only when usage drifts. Their failures are used to govern the next edit rather than to score a fixed system.

Change-aware regression testing. The closest technical relative from software engineering is regression test generation for code changes. CLEVEREST (Liu et al., 2025) generates bug-revealing and patch-validating tests conditioned on a specific commit. Classical regression test selection and prioritization pick a subset of an existing suite for a change, and mutation testing measures how well such a suite discriminates faulty variants. The object under test in SRT is a natural-language and code skill artifact whose oracle can be a trajectory predicate, the suite is synthesized from the artifact's own execution history and persisted across versions rather than selected from an external code repository, and the output is an admission action rather than a pass/fail report handed back to a developer. SelfVerify-Once occupies the analogous change-time verification slot without version persistence.

Tool-use self-verification and runtime monitoring. TOOLVERIFIER (Mekala et al., 2024) equips agents with self-verification steps that filter tool selections and argument choices before execution. Runtime verification (Bartocci et al., 2018) and design-by-contract (Meyer, 1992) formalize executable specifications that monitor program behavior online. SRT inherits this stance and ap-

plies it to a longer horizon: contracts persist across skill versions and drive admission of the next candidate rather than firing on the current execution alone.

Preserving capabilities during adaptation. Continual learning studies how to add new capability without erasing prior behavior (van de Ven et al., 2024; Shi et al., 2025; Zheng et al., 2025b; Yang et al., 2024). At the parameter level, EWC (Kirkpatrick et al., 2017) regularizes weight changes toward parameters important for old tasks. Deep generative replay (Shin et al., 2017) rehearses past inputs from a learned generator. Self-synthesized rehearsal (Huang et al., 2024) adapts this idea to large language models by generating training data for prior tasks, and SEEKR (He et al., 2024) applies selective attention-guided knowledge retention. At the skill level, CPE (Yu et al., 2026) constrains the rate at which artifacts change and provides the closest baseline for our setting. SEVerA (Banerjee et al., 2026) verifies agent-program synthesis against externally supplied formal specifications. Appendix D places these along seven axes—history source, oracle, version persistence, admission action, repair, audit, target object—that decide how each mechanism behaves during skill updates.

Skill-based agent evolution. Skill libraries (Gao et al., 2026; Zheng et al., 2026; Zhou et al., 2026; Fang et al., 2025) externalize reusable behaviors into artifacts that agents retrieve and revise during long-running tasks. Recent systems focus on how such libraries should be extended. SkillRL (Wang et al., 2025a) co-evolves skills and policies through failure analysis. SkillOS (Ouyang et al., 2026) learns a skill curator that manages the growing library. CoEvoSkills (Zhang et al., 2026a) uses co-evolutionary verification with textual feedback to discover and refine skills. SkillFoundry (Shen et al., 2026) incorporates heterogeneous scientific resources into evolving libraries. EmbodiSkill (Ju et al., 2026) adds skill-aware reflection for embodied agents, and MemSkill (Zhang et al., 2026b) organizes evolving memory around skill units. These systems focus on *proposing* improvements. The unit of control is a new candidate skill version, and admission is typically decided by re-running the motivating new task, without any explicit account of the behaviors the old library already delivered. SRT operates downstream of such proposers and returns an admission or repair decision grounded in persistent tests derived from each skill's own history.

3 Problem Formulation

Skills and evolution. An agent is a tuple $\mathcal{A} = (\pi, \mathcal{S}, \Phi)$ comprising a frozen policy π, a mutable skill library $\mathcal{S}$, and an evolution operator Φ. Each skill $s_k = (d_k, c_k, m_k)$ stores a description, an executable component or instruction recipe, and metadata including retrieval triggers, dependencies, and a version tag. For task t, the policy retrieves $\mathcal{S}_t \subseteq \mathcal{S}$, produces trajectory $\tau = \pi(t, \mathcal{S}_t)$, and receives a binary verdict $r(\tau, t)$. The operator proposes additions, edits, merges, or deletions from accumulated experience:

$$\mathcal{S}^{(i+1)} = \Phi\big(\mathcal{S}^{(i)}, \{(\tau_j, t_j, r_j)\}_{j=1}^{N}\big). \tag{1}$$

We study an admission controller wrapped around these proposals. Its input is a candidate library, and its output is the version that actually enters $\mathcal{S}$.

Erosion and adaptation. Let $\mathcal{T}_{\text{old}}$ and $\mathcal{T}_{\text{new}}$ denote the old- and new-task distributions. The erosion caused by an update is the drop in expected old-task reward (Yu et al., 2026):

$$\Delta_{\text{erode}} = \mathbb{E}_{t\sim\mathcal{T}_{\text{old}}}\left[r(\pi(t, \mathcal{S}^{(i)}), t) - r(\pi(t, \mathcal{S}^{(i+1)}), t)\right]. \tag{2}$$

Post-stream retained accuracy A_{ret} and adaptation accuracy A_{adapt} measure performance on the two distributions. We report both and the composite $\text{CPI} = A_{\text{ret}} + \lambda A_{\text{adapt}}$, with $\lambda = 1$ throughout. Accuracies are percentages and erosion is in percentage points. The coverage bound in Proposition 1 uses rewards in $[0, 1]$.

Why not textual similarity. Textual similarity between skill versions need not imply behavioral similarity, and different trajectories can implement the same behavior. SRT therefore expresses preservation through assertions over outcomes, tool traces, and prohibited failures rather than through the surface diff of two artifacts.

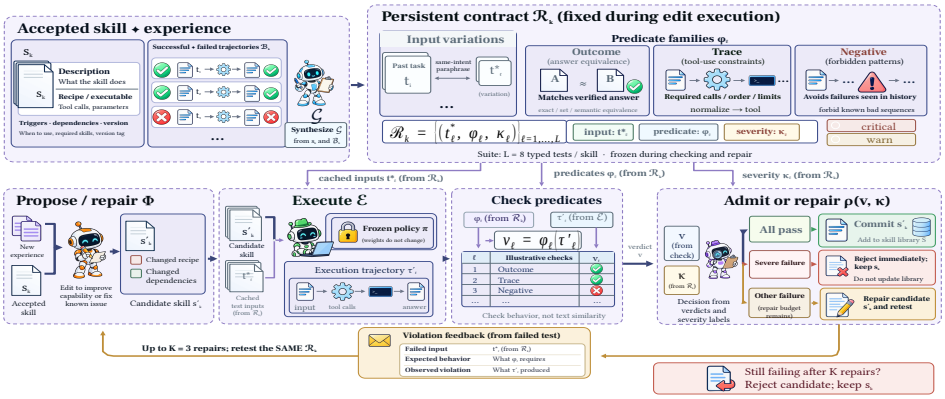


Figure 1: **Preserve the contract, verify the edit.** *Illustrative schematic (not an experiment log).* Cached inputs drive candidate execution. Predicates convert trajectories into verdicts, and severity labels govern admission or bounded repair. Repair reuses the existing suite. Severe failures are rejected immediately.

4 SELF-REGRESSION-TEST

How does SRT construct a persistent behavioral contract from experience, use it to admit an update, and keep it aligned with current usage? Figure 1 lays out the three components—synthesizer $\mathcal{G}$, executor $\mathcal{E}$, and controller ρ—that answer each of these questions in turn, without changing the upstream proposer.

From trajectories to assertions. For skill s_k, the buffer $\mathcal{B}_k$ stores successful and failed trajectories in which the skill was retrieved. The synthesizer produces $\mathcal{R}_k = \{(t_l^*, \phi_l, \kappa_l)\}_{l=1}^{L}$, where t_l^* is a paraphrastic test input, ϕ_l a trajectory predicate, and $\kappa_l \in \{\text{CRITICAL}, \text{WARN}\}$ a severity label. A structured-output prompt requests three complementary predicate families. *Outcome* predicates preserve a verified answer under task-appropriate equivalence (exact match, set equality, or semantic equivalence). *Trace* predicates preserve execution properties the final answer can conceal, such as invoking normalization before a tool call. *Negative* predicates prohibit patterns identified in failed trajectories. Semantic outcome checks use a majority of five LLM-judge votes when a deterministic predicate is unavailable (Gu et al., 2024; Wang et al., 2025b). Pairing a different synthesizer backbone with the policy reduces shared blind spots (Thakur et al., 2024; Zhuge et al., 2025).

Checking a candidate. When Φ proposes s'_k, the executor evaluates it against the frozen $\mathcal{R}_k$ and returns a pass/fail verdict for each predicate. The controller combines verdicts with severity labels into three actions. An all-pass candidate is committed as the next version. A candidate whose critical-failure fraction exceeds β is rejected immediately and the previous version retained. Other failures enter a bounded repair loop.

Bounded repair. Each repair attempt returns the failing input, the expected behavior, and the observed violation to the proposer, which produces a revised candidate. The revised candidate is re-executed against the same unmodified suite. Predicates are not regenerated to fit the edit. If failures remain after K repairs, the candidate is rejected and the previous version retained. A predicate-only variant withholds the expected answer to isolate the effect of answer exposure (Section 5.3).

A regression hidden by outcome agreement. Consider a skill whose accepted recipe reranks retrieved passages after an inherited deduplication helper. A useful edit swaps the BM25 reranker for a cross-encoder but silently drops that helper. On inputs without duplicate passages, the outcome checks still agree, yet a trace predicate requiring deduplication before reranking fails on duplicate-heavy retrievals and exposes duplicated top-ranked passages (Figure 2). Because this failure is recoverable, the controller returns the violating input, the required behavior, and the observed violation; the first repair restores the dependency while retaining the new reranker, and the repaired candidate is committed only after passing the unchanged suite. Outcome parity is insufficient be-

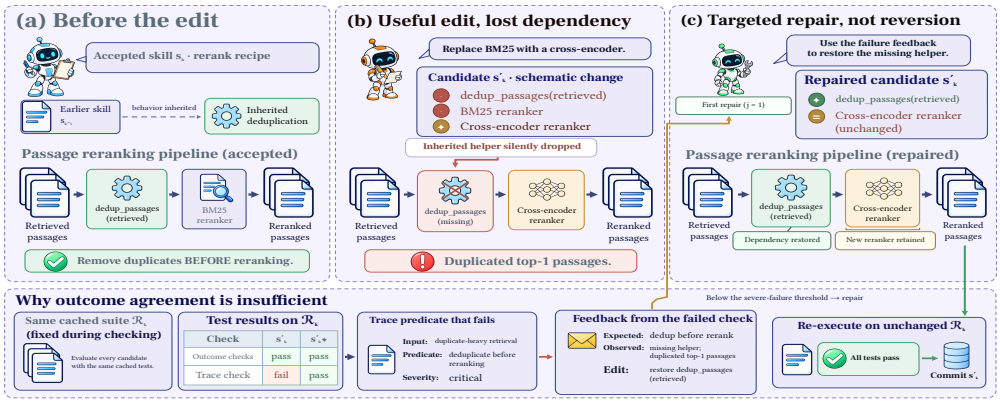


Figure 2: **A useful edit can erase an inherited dependency.** *Illustrative mechanism schematic, not a raw trace.* A useful update can pass outcome checks while silently dropping an inherited preprocessing step; a trace predicate exposes the regression, feedback restores the helper, and the repaired candidate is re-executed on the unchanged suite before a conditional commit.

Algorithm 1: SRT admission and repair for one proposed skill update.

Input: Skill s_k, buffer $\mathcal{B}_k$, cached suite $\mathcal{R}_k=\{(t_l^*, \phi_l, \kappa_l)\}_{l=1}^{L}$, proposer Φ, repair budget K, rejection threshold β

Output: An admission action and the retained skill version

$s'_k \leftarrow \Phi(s_k, \text{new experience})$;
for $j \leftarrow 0$ **to** K **do**
  Execute s'_k on every t_l^* and obtain traces τ'_l;
  $F \leftarrow \{l : \phi_l(\tau'_l)=\text{fail}\}$;
  $F_{\text{crit}} \leftarrow \{l \in F : \kappa_l=\text{CRITICAL}\}$;
  if $F = \varnothing$ **then**
    return *commit* s'_k;
  else if $|F_{\text{crit}}|/L > \beta$ **then**
    return *reject* and retain s_k;
  else if $j < K$ **then**
    $s'_k \leftarrow \Phi(s'_k, \{(t_l^*, \phi_l, v_l)\}_{l \in F})$;
return *reject* and retain s_k;

cause it treats behaviorally distinct trajectories as equivalent whenever they agree on the final answer. A persistent contract that also inspects the trace catches the regression before it enters the library.

Maintaining the suite. Tests are cached between refreshes. The default refresh period is $T=50$ evolution rounds, with $L=8$ tests per skill and a buffer of 64 trajectories. Refreshing folds changing usage into the contract. Maintenance and library-scaling results are reported in Appendix M.

A coverage-conditioned bound. Let $\mathcal{T}^k_{\text{old}}$ be a finite, uniformly weighted set of old tasks depending on s_k. Let $\mathcal{T}^k_{\text{cov}}$ contain the tasks for which passing the suite implies preservation of the original reward, and define $\text{Cov} = |\mathcal{T}^k_{\text{cov}}|/|\mathcal{T}^k_{\text{old}}|$.

Proposition 1 (Coverage-conditioned erosion)**.** *For an accepted single-skill update with a deterministic, sound verifier on covered tasks and rewards in* $[0, 1]$*, average erosion satisfies* $\Delta^k_{\text{erode}} \leq 1 - \text{Cov}$.

Covered tasks have no positive reward loss by the certification assumption, and each uncovered task can lose at most one unit of reward. Averaging yields the uncovered fraction. Appendix A extends the bound to stochastic verification.

What this bound does not guarantee. Three preconditions of Proposition 1 are not discharged symbolically. (i) *Predicate soundness*: $\mathcal{T}^k_{\text{cov}}$ contains only inputs whose predicates deterministically map traces to correctness; LLM-judge and semantic-outcome errors are measured by the 600-update

Table 1: Main results across benchmarks and evolution operators. Retained and adaptation accuracies are percentages, and erosion is in percentage points. Each benchmark–operator pair compares vanilla evolution (V) with the same operator wrapped by SRT (+SRT). Cells are three-seed means with standard deviations as subscripts.

	$A_{\text{ret}} \uparrow$		$A_{\text{adapt}} \uparrow$		Erosion ↓		CPI ↑	
Operator	V	+SRT	V	+SRT	V	+SRT	V	+SRT
Lifelong AgentBench								
EvoSkill (Zhang et al., 2026a)	$48.2_{\pm 0.5}$	$\mathbf{59.1}_{\pm 0.4}$	$\mathbf{55.7}_{\pm 0.4}$	$54.3_{\pm 0.4}$	$17.2_{\pm 0.5}$	$\mathbf{6.6}_{\pm 0.3}$	$103.9_{\pm 0.8}$	$\mathbf{113.4}_{\pm 0.6}$
SkillRL (Wang et al., 2025a)	$47.6_{\pm 0.6}$	$\mathbf{59.0}_{\pm 0.4}$	$\mathbf{62.4}_{\pm 0.3}$	$60.9_{\pm 0.4}$	$18.7_{\pm 0.5}$	$\mathbf{7.3}_{\pm 0.3}$	$110.0_{\pm 0.8}$	$\mathbf{119.9}_{\pm 0.6}$
SkillOS (Ouyang et al., 2026)	$51.3_{\pm 0.5}$	$\mathbf{61.4}_{\pm 0.4}$	$\mathbf{58.1}_{\pm 0.4}$	$57.0_{\pm 0.4}$	$16.4_{\pm 0.5}$	$\mathbf{6.1}_{\pm 0.3}$	$109.4_{\pm 0.8}$	$\mathbf{118.4}_{\pm 0.7}$
ALITA-G (Shen et al., 2026)	$49.8_{\pm 0.5}$	$\mathbf{60.2}_{\pm 0.4}$	$\mathbf{56.9}_{\pm 0.4}$	$55.8_{\pm 0.4}$	$15.8_{\pm 0.4}$	$\mathbf{6.4}_{\pm 0.3}$	$106.7_{\pm 0.8}$	$\mathbf{116.0}_{\pm 0.6}$
SealQA								
EvoSkill	$22.4_{\pm 0.5}$	$\mathbf{32.8}_{\pm 0.4}$	$\mathbf{38.7}_{\pm 0.5}$	$37.9_{\pm 0.4}$	$14.2_{\pm 0.4}$	$\mathbf{5.1}_{\pm 0.3}$	$61.1_{\pm 0.7}$	$\mathbf{70.7}_{\pm 0.6}$
SkillRL	$23.1_{\pm 0.5}$	$\mathbf{33.5}_{\pm 0.4}$	$\mathbf{40.2}_{\pm 0.5}$	$39.4_{\pm 0.4}$	$13.5_{\pm 0.4}$	$\mathbf{4.8}_{\pm 0.3}$	$63.3_{\pm 0.7}$	$\mathbf{72.9}_{\pm 0.6}$
SkillOS	$24.0_{\pm 0.5}$	$\mathbf{34.2}_{\pm 0.4}$	$\mathbf{39.1}_{\pm 0.5}$	$38.3_{\pm 0.4}$	$13.1_{\pm 0.4}$	$\mathbf{4.6}_{\pm 0.3}$	$63.1_{\pm 0.7}$	$\mathbf{72.5}_{\pm 0.6}$
ALITA-G	$23.6_{\pm 0.5}$	$\mathbf{33.7}_{\pm 0.4}$	$\mathbf{39.5}_{\pm 0.5}$	$38.7_{\pm 0.4}$	$13.4_{\pm 0.4}$	$\mathbf{4.7}_{\pm 0.3}$	$63.1_{\pm 0.7}$	$\mathbf{72.4}_{\pm 0.6}$
OfficeQA Pro								
EvoSkill	$55.1_{\pm 0.4}$	$\mathbf{63.7}_{\pm 0.3}$	$\mathbf{67.9}_{\pm 0.3}$	$67.0_{\pm 0.3}$	$11.3_{\pm 0.4}$	$\mathbf{3.5}_{\pm 0.2}$	$123.0_{\pm 0.7}$	$\mathbf{130.7}_{\pm 0.5}$
SkillRL	$54.4_{\pm 0.4}$	$\mathbf{63.1}_{\pm 0.3}$	$\mathbf{68.5}_{\pm 0.3}$	$67.3_{\pm 0.3}$	$11.6_{\pm 0.4}$	$\mathbf{3.7}_{\pm 0.2}$	$122.9_{\pm 0.7}$	$\mathbf{130.4}_{\pm 0.5}$
SkillOS	$56.2_{\pm 0.4}$	$\mathbf{64.5}_{\pm 0.3}$	$\mathbf{66.8}_{\pm 0.3}$	$66.0_{\pm 0.3}$	$10.8_{\pm 0.4}$	$\mathbf{3.4}_{\pm 0.2}$	$123.0_{\pm 0.7}$	$\mathbf{130.5}_{\pm 0.5}$
ALITA-G	$55.7_{\pm 0.4}$	$\mathbf{64.1}_{\pm 0.3}$	$\mathbf{67.2}_{\pm 0.3}$	$66.4_{\pm 0.3}$	$11.0_{\pm 0.4}$	$\mathbf{3.6}_{\pm 0.2}$	$122.9_{\pm 0.7}$	$\mathbf{130.5}_{\pm 0.5}$
Mean	42.6	52.4	55.1	54.1	13.9	5.0	97.7	106.5

V is the vanilla operator run alone. +SRT adds our admission controller downstream of the same operator, and is our method. Each cell reports the three-seed mean with the standard deviation as a subscript. Shading identifies the +SRT column, and bold marks the better value in each V/+SRT pair. Operator citations are shown once on first appearance in the top block.

audit (Section 5.3) and folded into the stochastic error term ϵ (Appendix A). (ii) *Single-skill scope*: simultaneous multi-skill refactorings can violate a joint contract without violating any per-skill contract; Appendix F quantifies the leakage. (iii) *Coverage estimation*: Cov is estimated empirically from held-out old tasks (Table 13), yielding $\text{Cov} \in [0.71, 0.89]$ across maintenance strategies. The bound is a coverage accounting rule; reliability is settled by the audit.

5 Experiments

We ask five questions. **RQ1:** Does SRT preserve old-task accuracy without suppressing adaptation across operators and benchmarks? **RQ2:** What do the structured tests and the repair loop each contribute? **RQ3:** Do the tests generalize beyond exposed inputs and reliably detect regressions? **RQ4:** How does the safeguard scale with maintenance and backbone choice? **RQ5:** Do the generated tests improve human reviewer decisions on real update audits?

Benchmarks and operators. We use Lifelong AgentBench (Zheng et al., 2025a), SealQA (Pham et al., 2026), and OfficeQA Pro (Opsahl-Ong et al., 2026). Following the lifelong protocol of Yu et al. (2026), old tasks initialize the skill library, new tasks drive evolution, and held-out old tasks measure retention. The four upstream operators are EvoSkill/CoEvoSkills (Zhang et al., 2026a), SkillRL (Wang et al., 2025a), SkillOS (Ouyang et al., 2026), and ALITA-G/SkillFoundry (Shen et al., 2026). The main comparison adds SRT to each operator, and CPE regularization and replay serve as preservation baselines. The default policy is GPT-5.1 and the default synthesizer is Claude Opus 4.7. Dataset sizes, protocol details, and the 5×5 backbone grid are in Appendices B and N.

5.1 Preservation and Adaptation (RQ1)

Across the twelve benchmark–operator combinations (Table 1), SRT raises retained accuracy by 8.3–11.4 points at a 0.8–1.5-point cost in adaptation accuracy, lifting CPI by 7.5–9.9 points. On Lifelong AgentBench with SkillRL, erosion drops from 18.7 to 7.3 points while adaptation accuracy moves from 62.4% to 60.9%. Against the selected preservation baselines (Table 2), SRT attains both lower erosion *and* higher adaptation accuracy than replay and both CPE settings. Vanilla evolution

Table 3: Cross-matrix attribution across six benchmark–operator settings. CPI is reported for Vanilla, Replay, one-shot SelfVerify (matched verifier budget), historical failure-set reject, and Full SRT (five-seed means). ΔCPI is Full SRT vs. SelfVerify-Once with paired-bootstrap 95% CI (10,000 resamples); all intervals exclude zero after Bonferroni correction ($p < 0.001$). Full per-cell erosion and adaptation are in Appendix C.

Setting	Vanilla	Replay	SelfVerify	F.-set	SRT	ΔCPI [95% CI]
Lifelong×SkillRL	110.0	113.0	115.4	113.8	**119.9**	+4.5 [3.9, 5.1]
Lifelong×EvoSkill	103.9	107.5	109.7	108.4	**113.4**	+3.7 [3.1, 4.3]
SealQA×SkillRL	63.3	67.9	70.5	69.4	**72.9**	+2.4 [1.9, 2.9]
SealQA×EvoSkill	61.1	65.4	67.6	66.5	**70.7**	+3.1 [2.6, 3.6]
OfficeQA×SkillRL	122.9	126.1	128.4	127.3	**130.4**	+2.0 [1.6, 2.4]
OfficeQA×ALITA-G	122.9	126.0	128.2	127.2	**130.5**	+2.3 [1.9, 2.7]
Micro-avg CPI	97.4	101.0	103.3	102.1	**106.3**	+3.0

keeps the highest adaptation accuracy but at the cost of much greater erosion. The CPE regularizer trades adaptation for retention along a single scalar, whereas SRT decouples the two by rejecting the specific edits that damage old behavior (Figure 3).

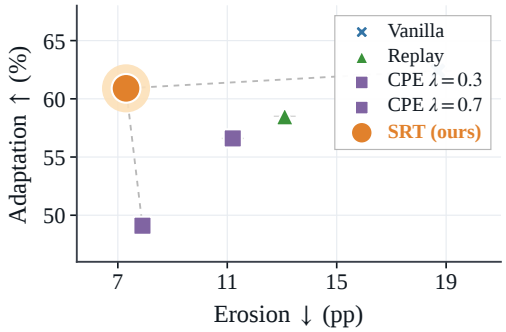


Figure 3: **Preservation–adaptation trade-off** on Lifelong AgentBench with SkillRL. SRT dominates the selected replay and CPE settings (Table 2).

Method	Erosion ↓	A_{adapt} ↑	CPI ↑
Vanilla	$18.7_{\pm 0.5}$	$62.4_{\pm 0.4}$	$110.0_{\pm 0.8}$
Replay (B=512)	$13.1_{\pm 0.4}$	$58.5_{\pm 0.4}$	$113.0_{\pm 0.7}$
CPE (λ=0.3)	$11.2_{\pm 0.4}$	$56.6_{\pm 0.4}$	$113.2_{\pm 0.7}$
CPE (λ=0.7)	$7.9_{\pm 0.3}$	$49.1_{\pm 0.4}$	$108.7_{\pm 0.7}$
SRT default	$\mathbf{7.3}_{\pm 0.3}$	$60.9_{\pm 0.4}$	$\mathbf{119.9}_{\pm 0.6}$

Table 2: Comparisons with replay and CPE on Lifelong AgentBench with SkillRL. Three-seed means with std as subscripts.

5.2 What Do Persistent Tests and Repair Contribute? (RQ2)

We isolate the ingredients of the safeguard with a matched-verifier-budget attribution across six representative benchmark–operator settings (Table 3). Within each cell, all methods consume the same 300 evolution proposals and the same five seeds, and pair with a matched verifier-token budget. Full SRT has the lowest erosion and the highest CPI in every cell, and the paired-bootstrap CPI gap over SelfVerify-Once ranges from +2.0 to +4.5 points and is significant after Bonferroni correction ($p < 0.001$). Replay reduces erosion by 3.0–5.6 points and lowers CPI variance but forfeits 1.6–3.9 adaptation points; the failure-set-reject baseline underperforms SelfVerify-Once, confirming that the gain of SRT is not driven by conservative rejection of the same historical inputs. SRT with rejection alone reduces erosion from 18.7 to 7.6 points on Lifelong AgentBench with SkillRL but leaves adaptation at 57.7%. Enabling repair lands erosion at 7.3 and lifts adaptation to 60.9%, a 3.2-point adaptation swing that the same tests would forfeit if used only to reject. Table 8 (Appendix E) reports the full commit/repair/reject decomposition per cell: micro-average reject rate is 7.8% and repair success is 75.4%, so the CPI gain is not explained by conservative rejection. Component ablations (Table 12, Appendix I) further confirm the design. Removing negative assertions raises erosion by 4.1 points, and exact-match outcomes push adaptation down to 54.1% while inflating rollback frequency to 41%, a signature of conservative filtering rather than better discrimination.

5.3 Do the Tests Generalize and Detect Regressions? (RQ3)

Full SRT attains 96.1% exposed pass and 90.3% hidden pass across 300 update episodes with three-paraphrase and ten-old-task hidden splits (Table 13, Appendix J), against 71.0% hidden pass for

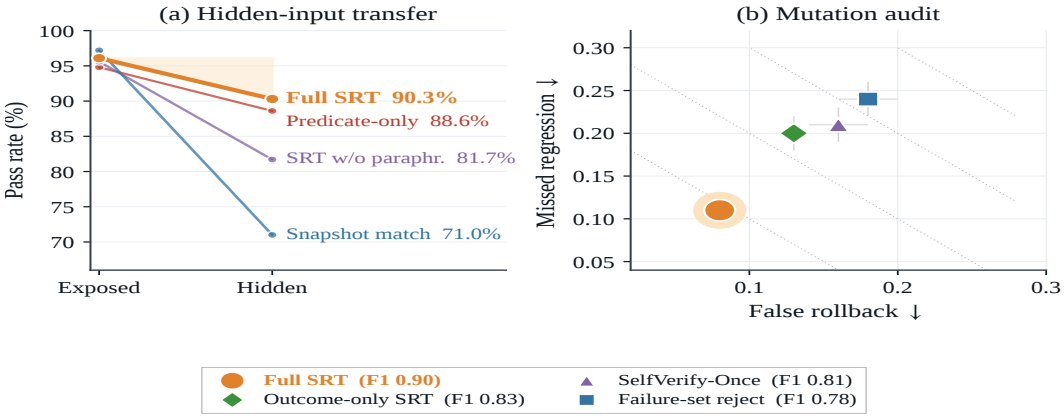


Figure 4: **Reliability beyond exposed tests.** (a) Exposed/hidden pass rates over 300 update episodes. The exposed–hidden gap collapses from 26.2 to 5.8 points. (b) False rollback vs. missed regression in the independent 600-update audit, with F1 iso-lines.

Table 4: Suite maintenance under changing usage.

Strategy	Coverage	Erosion	CPI	Cost (%)
Fixed	0.71	10.6	116.4	11.2
Append-only	0.81	8.5	118.6	12.6
Periodic (T=10)	0.83	8.0	119.2	12.9
Periodic ($T = 50$)	0.86	7.3	119.9	14.0
Every update	0.89	6.9	119.7	23.5

Coverage is a fraction, and erosion is in percentage points.

snapshot-based exact matching. Snapshot matching wins on the memorized inputs it can rerun exactly, yet loses 26.2 points under paraphrastic hidden tests, whereas SRT loses only 5.8. Withholding the expected answer from repair feedback (predicate-only variant) yields 88.6% hidden pass and 58.5% held-out old-task accuracy, close to full SRT's 90.3% and 59.0%, so the improvement is not driven by copying answers through the repair channel. On an independent 600-update mutation audit (Table 14, Appendix K) with 300 harmful edits (over-specialization, tool substitution, dependency removal) and 300 benign or beneficial edits (behavior-preserving refactors, bug fixes, equivalent tool replacements), all excluded from synthesis and repair, full SRT reaches F1 0.90 with false rollback 0.08 and missed regression 0.11 (Figure 4).

5.4 Maintenance, Scaling, and Backbone Choice (RQ4)

Fixed suites yield coverage 0.71 and CPI 116.4. Periodic refresh at T=50 reaches coverage 0.86 and CPI 119.9 at 14.0% overhead. Refreshing after every update raises coverage only to 0.89 while pushing overhead to 23.5% and leaving CPI at 119.7 (Table 4). More frequent verification therefore does not automatically produce a better cost–performance point. With one targeted skill per round, growing the library from 10 to 400 skills raises eager overhead from 13.4% to 27.6%, whereas lazy overhead moves from 13.2% to 15.6% only (Figure 5, Appendix M). At 400 skills the lazy synthesis share is 3.1%, so the steady-state cost of adding coverage tracks changed skills rather than library size. Across the 5×5 policy–synthesizer grid (Appendix N), the same-backbone diagonal is the weakest configuration in every policy row. Pairing a synthesizer that differs from the policy improves CPI within each row, and the highest value 119.9 uses GPT-5.1 as policy and Claude Opus 4.7 as synthesizer, consistent with the intuition that shared judge blind spots dominate when policy and synthesizer coincide. Suite-size and threshold ablations (Appendix M) show that CPI saturates at L=8 and that erosion grows from 5.8 to 9.1 points as the critical-fraction threshold rises from 0.3 to 0.7.

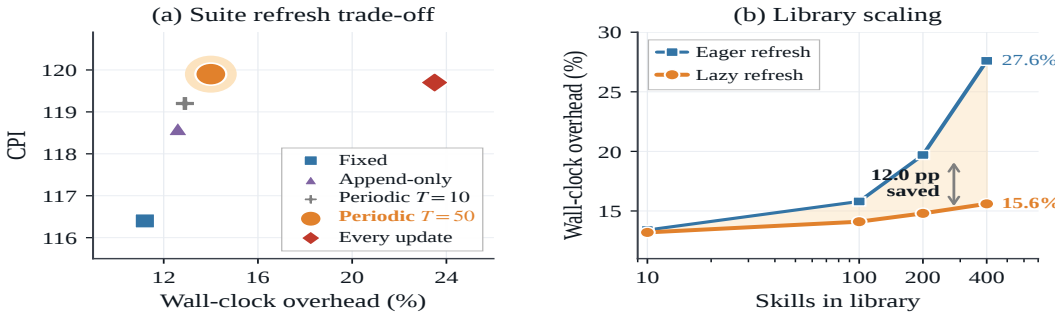


Figure 5: **Cost depends on what is refreshed.** (a) Five maintenance strategies (Table 4). The default uses $T = 50$. (b) Eager full-library versus lazy changed-buffer refresh across library sizes, with one targeted skill per round and 1,000 rounds.

5.5 Do the Generated Tests Help Human Reviewers? (RQ5)

Eighteen software-engineering graduate students screened for LLM-agent experience were assigned 60 audit items each (1,080 judgments), asking whether a proposed skill update preserves the referenced behavior. Three conditions in a Latin-square design compare diff only, diff plus three sampled trajectories, and diff plus the SRT predicate suite. Ground truth is the independent audit of Table 14 restricted to the affected skills. Exposing the tests raises decision accuracy from 54.8% to 84.6% and cuts median review time from 95.2 to 61.4 seconds. Fleiss κ across three-way agreement rises from 0.34 to 0.61. Paired bootstrap over participants shows Δaccuracy 95% CI $[+26.4, +33.4]$ ($p < 0.001$, Cohen's $d = 1.68$). Full participant statistics, per-condition confusion matrices, and exclusion counts are in Appendix P.

6 Discussion

Assertions surface decisions. Every rollback or repair carries the triggering assertion, giving reviewers a decision object rather than a summary statistic. In the human study (Appendix P), tests raise reviewer accuracy from 54.8% to 84.6% and Fleiss κ from 0.34 to 0.61, so persistent tests serve as both admission mechanisms and audit artifacts.

Scope. SRT observes only English skill evolution. Other channels require appropriate assertions, and generated predicates may preserve historical mistakes, miss ambiguity, or overlook multi-skill interactions (Appendix F). Behavior outside the contract remains outside its guarantee.

7 Conclusion

Skill-library evolution is an admission problem: a candidate should satisfy a persistent, skill-specific behavioral contract before entering the library. SRT synthesizes outcome, trace, and negative assertions from past trajectories and turns failures into commit, bounded repair, or rollback decisions without modifying the policy or proposer. Across four operators and three benchmarks, it raises retained accuracy by 8.3–11.4 points at a 0.8–1.5-point adaptation cost, reaches audit F1 0.90, and passes 90.3% of hidden paraphrases at 14% overhead.

Ethics Statement

Behavioral testing improves inspection but does not address adversarial or alignment failures. The minimal-risk pilot used consent, compensation, and no identifying information.

Broader Impact

SRT turns erosion into auditable events. Tests may omit rare segments, so audits should sample uncovered inputs; repair feedback may encourage literal output reproduction.

AI USE STATEMENT

Generative AI tools assisted with language polishing and the preparation and revision of scientific figures. Their use did not extend to dataset generation, experimental execution, the formulation or proof of mathematical claims, or the derivation of scientific conclusions. The authors reviewed and revised all AI-assisted text and figures to ensure accuracy, consistency with the method, and proper attribution. The authors assume full responsibility for the final work, including all text, scientific claims, and associated artifacts.

REFERENCES

Debangshu Banerjee, Changming Xu, Eugene Ie, et al. SEVerA: Verified synthesis of self-evolving agents. *arXiv preprint arXiv:2603.25111*, 2026.

Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. Introduction to runtime verification. *Lectures on Runtime Verification, Springer LNCS 10457*, 2018.

Arghavan Moradi Dakhel, Amin Nikanjam, Vahid Majdinasab, et al. Effective test generation using pre-trained large language models and mutation testing. *Information and Software Technology*, 2024. doi: 10.1016/j.infsof.2024.107468.

Jinyuan Fang, Yanwen Peng, Xi Zhang, et al. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, 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://arxiv.org/abs/2507.21046`.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, et al. A survey on LLM-as-a-Judge. *arXiv preprint arXiv:2411.15594*, 2024.

Jinghan He, Haiyun Guo, Kuan Zhu, et al. SEEKR: Selective attention-guided knowledge retention for continual learning of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. doi: 10.18653/v1/2024.emnlp-main.190.

Jianheng Huang, Leyang Cui, Ante Wang, Chengyi Yang, Xinting Liao, Linfeng Song, Junfeng Yao, and Jinsong Su. Mitigating catastrophic forgetting in large language models with self-synthesized rehearsal. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Ruofei Ju, Xinrui Wang, Xin Ding, et al. EmbodiSkill: Skill-aware reflection for self-evolving embodied agents. *arXiv preprint arXiv:2605.10332*, 2026.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 114:3521–3526, 2017.

Jing Liu, Seongmin Lee, Eleonora Losiouk, and Marcel Böhme. Can LLM generate regression tests for software commits? *arXiv preprint arXiv:2501.11086*, 2025.

Dheeraj Mekala et al. TOOLVERIFIER: Generalization to new tools via self-verification. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024.

Bertrand Meyer. Applying "design by contract". *IEEE Computer*, 1992.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, et al. OfficeQA Pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 2026.

Wendkûuni C. Ouédraogo, Yinghua Li, Xueqi Dang, et al. Test smells in LLM-generated unit tests. *arXiv preprint arXiv:2410.10628*, 2024.

Siru Ouyang, Jun Yan, Yanfei Chen, et al. SkillOS: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026.

Rangeet Pan, Myeong-Soo Kim, Rahul Krishna, et al. ASTER: Natural and multi-language unit test generation with LLMs. *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE-SEIP)*, 2025. doi: 10.1109/ICSE-SEIP66354.2025.00042.

Thinh Pham, Nguyen Nguyen, Pratibha Zunjare, Weiyuan Chen, Yu-Min Tseng, and Tu Vu. SealQA: Raising the bar for reasoning in search-augmented language models. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*, 2026.

Juan Altmayer Pizzorno and Emery D. Berger. CoverUp: Effective high coverage test generation for Python. *Proceedings of the ACM on Software Engineering*, 2025. doi: 10.1145/3729398.

Archiki Prasad, Elias Stengel-Eskin, Justin Chih-Yao Chen, et al. Learning to generate unit tests for automated debugging. *arXiv preprint arXiv:2502.01619*, 2025.

Marco Túlio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4902–4912, 2020. doi: 10.18653/v1/2020.acl-main.442.

Shuaike Shen, Wenduo Cheng, Mingqian Ma, et al. SkillFoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources. *arXiv preprint arXiv:2604.03964*, 2026.

Haizhou Shi, Zihao Xu, Hengyi Wang, et al. Continual learning of large language models: A comprehensive survey. *ACM Computing Surveys*, 2025. doi: 10.1145/3735633.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, et al. Judging the judges: Evaluating alignment and vulnerabilities in LLMs-as-Judges. *arXiv preprint arXiv:2406.12624*, 2024.

Gido M. van de Ven, Nicholas Soures, and Dhireesha Kudithipudi. Continual learning and catastrophic forgetting. *arXiv preprint arXiv:2403.05175*, 2024.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, et al. Reinforcement learning for self-improving agent with skill library. *arXiv preprint arXiv:2512.17102*, 2025a.

Ruiqi Wang, Jiyu Guo, Cuiyun Gao, et al. Can LLMs replace human evaluators? an empirical study of LLM-as-a-Judge in software engineering. *Proceedings of the ACM on Software Engineering*, 2025b. doi: 10.1145/3728963.

Shanglin Wu, Yuyang Luo, Yueqing Liang, et al. Scaling teams or scaling time? memory enabled lifelong learning in LLM multi-agent systems. *arXiv preprint arXiv:2604.03295*, 2026.

Yutao Yang, Jie Zhou, Xuanwen Ding, et al. Recent advances of foundation language models-based continual learning: A survey. *ACM Computing Surveys*, 2024. doi: 10.1145/3705725.

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

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, et al. CoEvoSkills: Self-evolving agent skills via co-evolutionary verification. *arXiv preprint arXiv:2604.01687*, 2026a.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, et al. MemSkill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026b.

Junhao Zheng, Xidi Cai, Qiuke Li, et al. LifelongAgentBench: Evaluating LLM agents as lifelong learners. *arXiv preprint arXiv:2505.11942*, 2025a.

Junhao Zheng, Shengjie Qiu, Chengming Shi, et al. Towards lifelong learning of large language models: A survey. *ACM Computing Surveys*, 2025b. doi: 10.1145/3716629.

Junhao Zheng, Chengming Shi, Xidi Cai, et al. Lifelong learning of large language model based agents: A roadmap. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026. doi: 10.1109/TPAMI.2025.3650546.

Chenyu Zhou, Huacan Chai, Wenteng Chen, et al. Externalization in LLM agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026.

Mingchen Zhuge, Changsheng Zhao, Dylan R. Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267, pp. 80569–80611. PMLR, 2025.

A Coverage Analysis and Its Assumptions

Proposition 1 concerns a single accepted skill update on a finite, uniformly weighted old-task set and assumes the covered subset is *certified*: passing the relevant predicates implies preservation of the original reward for every task in that subset. Historical inclusion, paraphrastic similarity, and agreement of two model outputs do not by themselves establish this implication.

Proof. Write $n = |\mathcal{T}^k_{\text{old}}|$ and let d_t be the old reward minus the reward after the accepted update. Partition the task set into the certified subset $\mathcal{T}^k_{\text{cov}}$ and its complement $\mathcal{T}^k_{\text{unc}}$. Certification and sound verification give $d_t \leq 0$ on the former, and bounded rewards give $d_t \leq 1$ on the latter. Therefore $\Delta^k_{\text{erode}} = \frac{1}{n}(\sum_{t\in\mathcal{T}^k_{\text{cov}}} d_t + \sum_{t\in\mathcal{T}^k_{\text{unc}}} d_t) \leq |\mathcal{T}^k_{\text{unc}}|/n = 1 - \text{Cov}$. □

Stochastic verification. If the probability that certification fails on a covered task among accepted updates is at most ϵ, positive loss on such a task is at most one with probability at most ϵ, giving $\mathbb{E}[\Delta^k_{\text{erode}} \mid \text{accepted}] \leq 1 - \text{Cov} + \epsilon\,\text{Cov}$. We report controller-level errors directly in the mutation audit rather than estimating ϵ from marginal judge accuracy.

B Benchmark and Inference Protocol

Table 5 summarizes the protocol. Old tasks populate the initial library, new tasks form the evolution stream, and a held-out old-task set evaluates retained accuracy every 50 rounds. The main study uses three seeds, and the attribution study uses five seeds with matched verifier calls and tokens.

Table 5: Benchmark protocol. Old-task pools and held-out evaluation sets serve different roles.

	Lifelong AB	SealQA	OfficeQA Pro
Old-task pool	4,120	1,480	2,050
New-task pool	1,030	620	680
Held-out old tasks	500	400	400
Evolution rounds	1,000	600	600
Initial skills (average)	62	34	41
Task verifier	Program	Exact match	LLM judge

Defaults. The default suite contains $L = 8$ tests, the critical-fraction threshold is $\beta = 0.5$, the repair budget is $K = 3$, and the refresh period is $T = 50$. Severe failures trigger immediate rejection, and other failures permit at most three repairs before the previous version is retained. Each skill maintains a trajectory buffer of 64 items. Semantic outcome verification uses five majority-voted judgments.

C Attribution Study (extended)

Table 6 reports the full cross-matrix attribution across six representative benchmark–operator settings. Historical failure-set rejection reruns raw old tasks without typed assertions or repair. SelfVerify-Once receives the original and candidate skill plus eight historical trajectories at the same

budget, without persistence, versioning, or repair. Random control tests spend the same verifier calls on unlearned checks. Full SRT dominates every baseline on both erosion and CPI in every cell. Judge and synthesizer temperature are matched to Full SRT for every baseline that uses them.

Table 6: Extended cross-matrix attribution: erosion, adaptation, CPI per method per setting. Five-seed means with standard deviations as subscripts.

Setting	Method	Erosion ↓	A_{adapt} ↑	CPI ↑
Lifelong×SkillRL	Vanilla	$18.7_{\pm 0.4}$	$62.4_{\pm 0.3}$	$110.0_{\pm 0.6}$
	Replay	$13.1_{\pm 0.4}$	$58.5_{\pm 0.4}$	$113.0_{\pm 0.7}$
	SelfVerify-Once	$10.2_{\pm 0.3}$	$59.3_{\pm 0.3}$	$115.4_{\pm 0.5}$
	Failure-set reject	$11.3_{\pm 0.3}$	$58.8_{\pm 0.3}$	$113.8_{\pm 0.5}$
	Full SRT	$\mathbf{7.3}_{\pm 0.3}$	$\mathbf{60.9}_{\pm 0.3}$	$\mathbf{119.9}_{\pm 0.5}$
Lifelong×EvoSkill	Vanilla	$17.2_{\pm 0.5}$	$55.7_{\pm 0.4}$	$103.9_{\pm 0.8}$
	Replay	$12.4_{\pm 0.4}$	$52.6_{\pm 0.4}$	$107.5_{\pm 0.7}$
	SelfVerify-Once	$9.5_{\pm 0.3}$	$52.9_{\pm 0.4}$	$109.7_{\pm 0.6}$
	Failure-set reject	$10.8_{\pm 0.4}$	$52.4_{\pm 0.4}$	$108.4_{\pm 0.6}$
	Full SRT	$\mathbf{6.6}_{\pm 0.3}$	$\mathbf{54.3}_{\pm 0.4}$	$\mathbf{113.4}_{\pm 0.6}$
SealQA×SkillRL	Vanilla	$13.5_{\pm 0.4}$	$40.2_{\pm 0.5}$	$63.3_{\pm 0.7}$
	Replay	$9.7_{\pm 0.4}$	$38.1_{\pm 0.4}$	$67.9_{\pm 0.6}$
	SelfVerify-Once	$7.6_{\pm 0.3}$	$38.6_{\pm 0.4}$	$70.5_{\pm 0.5}$
	Failure-set reject	$8.4_{\pm 0.4}$	$38.3_{\pm 0.4}$	$69.4_{\pm 0.6}$
	Full SRT	$\mathbf{4.8}_{\pm 0.3}$	$\mathbf{39.4}_{\pm 0.4}$	$\mathbf{72.9}_{\pm 0.6}$
SealQA×EvoSkill	Vanilla	$14.2_{\pm 0.4}$	$38.7_{\pm 0.5}$	$61.1_{\pm 0.7}$
	Replay	$10.6_{\pm 0.4}$	$36.5_{\pm 0.4}$	$65.4_{\pm 0.6}$
	SelfVerify-Once	$8.1_{\pm 0.3}$	$37.0_{\pm 0.4}$	$67.6_{\pm 0.5}$
	Failure-set reject	$9.0_{\pm 0.4}$	$36.7_{\pm 0.4}$	$66.5_{\pm 0.6}$
	Full SRT	$\mathbf{5.1}_{\pm 0.3}$	$\mathbf{37.9}_{\pm 0.4}$	$\mathbf{70.7}_{\pm 0.6}$
OfficeQA×SkillRL	Vanilla	$11.6_{\pm 0.4}$	$68.5_{\pm 0.3}$	$122.9_{\pm 0.7}$
	Replay	$8.4_{\pm 0.3}$	$65.7_{\pm 0.4}$	$126.1_{\pm 0.6}$
	SelfVerify-Once	$6.5_{\pm 0.3}$	$66.1_{\pm 0.3}$	$128.4_{\pm 0.5}$
	Failure-set reject	$7.3_{\pm 0.3}$	$65.9_{\pm 0.4}$	$127.3_{\pm 0.5}$
	Full SRT	$\mathbf{3.7}_{\pm 0.2}$	$\mathbf{67.3}_{\pm 0.3}$	$\mathbf{130.4}_{\pm 0.5}$
OfficeQA×ALITA-G	Vanilla	$11.0_{\pm 0.4}$	$67.2_{\pm 0.3}$	$122.9_{\pm 0.7}$
	Replay	$8.0_{\pm 0.3}$	$64.5_{\pm 0.4}$	$126.0_{\pm 0.6}$
	SelfVerify-Once	$6.2_{\pm 0.3}$	$64.9_{\pm 0.3}$	$128.2_{\pm 0.5}$
	Failure-set reject	$6.9_{\pm 0.3}$	$64.7_{\pm 0.4}$	$127.2_{\pm 0.5}$
	Full SRT	$\mathbf{3.6}_{\pm 0.2}$	$\mathbf{66.4}_{\pm 0.3}$	$\mathbf{130.5}_{\pm 0.5}$

D Closest-Work Matrix

Table 7: Closest-work matrix. Where a method lacks a specific mechanism the cell is marked —. The last row summarizes the combination SRT uses.

Method	History	Oracle	Persist.	Action	Repair
Replay (Shin et al., 2017)	inputs	original reward	—	rehearse	—
CPE (Yu et al., 2026)	versions	diff magnitude	—	regularize	—
EWC/SEEKR (Kirkpatrick et al., 2017; He et al., 2024)	weights	Fisher / attention	—	regularize	—
TestPilot (Dakhel et al., 2024)	code	return value	no	report	—
ASTER (Pan et al., 2025)	code	return value	no	report	—
CoverUp (Pizzorno & Berger, 2025)	code	coverage	no	report	—
CLEVEREST (Liu et al., 2025)	commit	return value	no	report	—
TOOLVERIFIER (Mekala et al., 2024)	call	one-shot check	no	filter	—
SEVerA (Banerjee et al., 2026)	program	formal spec	yes	verify	—
SelfVerify-Once (ours)	candidate	one-shot judge	no	report	—
Failure-set reject (ours)	failures	original reward	version	reject	—
SRT	trajectories	outcome+trace+negative	version	commit/repair/reject	typed

Table 8: Decision decomposition of Full SRT across six benchmark–operator settings (300 proposals per cell, five seeds). Fractions sum to 100%; repair success is the fraction of repair-entered candidates that commit.

Setting	Commit	Repair	Reject	Repair-succ.
Lifelong×SkillRL	71.4%	19.5%	9.1%	73.2%
Lifelong×EvoSkill	74.0%	17.2%	8.8%	71.5%
SealQA×SkillRL	76.1%	16.4%	7.5%	77.4%
SealQA×EvoSkill	74.8%	17.1%	8.1%	75.3%
OfficeQA×SkillRL	78.2%	15.3%	6.5%	78.1%
OfficeQA×ALITA-G	77.5%	15.9%	6.6%	76.9%
Micro-average	75.3%	16.9%	7.8%	75.4%

Table 9: Two-skill joint edits with shared helpers (180 episodes, three seeds).

Admission policy	Both preserved ↑	One broken ↓	Both broken ↓	Overhead
Per-skill Full SRT	71.2%	19.9%	8.9%	14.0%
Sequential admission	75.4%	18.7%	5.9%	14.7%
Joint-audit SRT	**81.6%**	**15.3%**	**3.1%**	16.3%

Joint-audit SRT adds a shared-helper predicate re-executed after both edits are committed. It captures 71% of the inter-skill regressions missed by per-skill contracts; the residual are helpers not surfaced by either skill's contract.

E Decision Decomposition

Table 8 reports commit, repair, and reject shares together with eventual repair success. The 7.8% micro-average reject rate shows that the CPI gain is not achieved by rejecting most proposals.

F Multi-skill and Shared-Dependency Regressions

The single-skill scope of Proposition 1 implies that per-skill contracts fire independently even when several skills share a helper, a router, or a memory slice. To measure the resulting gap we curate 180 joint-edit episodes on Lifelong AgentBench with SkillRL, in which two skills that share at least one helper are edited within the same round. Each episode is labeled by an independent execution audit as one of {both preserved, one broken, both broken, both improved}. Table 9 compares three admission policies. Per-skill Full SRT treats each edit as independent. Sequential admission processes the two edits in retrieval order and rolls back the second if it violates the first's contract. Joint-audit SRT adds a shared-helper predicate that re-executes a curated fixture of jointly reachable inputs after both edits are committed, and rolls back the round if the fixture regresses. Joint-audit reduces the both-broken rate from 8.9% to 3.1% and captures 71% of the inter-skill regressions that per-skill contracts miss, at the cost of 2.3% additional overhead. Some inter-skill regressions remain, most of which involve helpers that neither pre-existing skill exposed as a required predicate.

G Judge-Family Robustness and Human Calibration

Table 10 varies the semantic-outcome judge configuration on Lifelong AgentBench with SkillRL while holding the rest of SRT fixed. Single-judge configurations drop audit F1 by 0.05–0.07 and shift the false-rollback vs. missed-regression balance. Same-family five-vote ensembles reach F1 0.87; a mixed-family ensemble of five reaches F1 0.90. Human calibration on 400 audit items shows 86% agreement between the mixed-family ensemble and consensus human labels, with Krippendorff $\alpha = 0.81$ on the disagreement subset. Systematic disagreements concentrate on cases where the reference answer is under-specified (*e.g.* non-canonical entity orderings); we route such cases to human arbitration in the audit itself and treat them as ambiguous rather than as regressions.

H Cost Breakdown

Table 11 decomposes the 14% verifier overhead into synthesis, suite execution, semantic judgment, and repair for Full SRT at the default setting. Verifier costs are dominated by suite execution, and

Table 10: Judge-family robustness on the 600-update mutation audit.

Judge configuration	F1 ↑	F. rollback ↓	M. regression ↓	Human agree. ↑
Single GPT-5.1	$0.83_{\pm 0.02}$	$0.14_{\pm 0.02}$	$0.16_{\pm 0.02}$	79%
Single Claude Opus 4.7	$0.85_{\pm 0.02}$	$0.12_{\pm 0.02}$	$0.14_{\pm 0.02}$	81%
Five GPT-5.1 majority	$0.86_{\pm 0.02}$	$0.11_{\pm 0.02}$	$0.13_{\pm 0.02}$	82%
Five Claude Opus 4.7 majority	$0.87_{\pm 0.02}$	$0.10_{\pm 0.01}$	$0.12_{\pm 0.02}$	83%
Mixed-family five (default)	$\mathbf{0.90}_{\pm 0.01}$	$\mathbf{0.08}_{\pm 0.01}$	$\mathbf{0.11}_{\pm 0.01}$	**86%**

Mixed-family panel: 2×GPT-5.1, 2×Claude Opus 4.7, 1×Llama-3.1-405B. Human agreement measured on 400 audit items with Krippendorff $\alpha = 0.81$.

Table 11: Cost breakdown of Full SRT on Lifelong AgentBench with SkillRL. Percentages are fractions of the total wall-clock overhead over the baseline.

Stage	Share (%)	P50 latency (s)	P95 latency (s)	Token (kt) / episode
Suite synthesis (refresh)	18.4	0.9	2.7	6.1
Suite execution	46.2	1.3	3.6	4.8
Semantic judgment (5-vote)	27.9	0.7	1.8	2.5
Repair (typed feedback)	7.5	1.1	4.2	1.6

Total default overhead is 14.0% of baseline wall-clock. Cache hit rate for cached predicate–input pairs is 82.4% across the 50-round refresh period.

Table 12: Component ablations on Lifelong AgentBench with SkillRL. Values are three-seed means with standard deviations as subscripts.

Variant	Erosion ↓	A_{adapt} ↑	Rollback (%)
Full SRT	$\mathbf{7.3}_{\pm 0.3}$	$\mathbf{60.9}_{\pm 0.4}$	$22_{\pm 1.5}$
Without negative assertions	$11.4_{\pm 0.4}$	$60.5_{\pm 0.4}$	$17_{\pm 1.4}$
Without repair	$7.6_{\pm 0.3}$	$57.7_{\pm 0.4}$	$31_{\pm 1.8}$
Exact-match outcomes	$6.9_{\pm 0.3}$	$54.1_{\pm 0.4}$	$41_{\pm 2.0}$
Without paraphrases	$9.8_{\pm 0.4}$	$60.3_{\pm 0.4}$	$19_{\pm 1.5}$
Without severity labels	$8.4_{\pm 0.3}$	$60.2_{\pm 0.4}$	$26_{\pm 1.6}$

Rollback frequency is not a false-positive rate. Decision errors are measured separately in Table 14.

semantic judgment accounts for the majority of the remainder. P95 latency is 2.4× the P50 because a small fraction of episodes trigger the full repair budget. Caching predicate–input pairs between refreshes reduces re-execution cost to 3.1% of the library synthesis cost at 400 skills.

I Component Ablations

Table 12 isolates the contribution of each design element on Lifelong AgentBench with SkillRL. Removing negative assertions raises erosion by 4.1 points. Exact-match outcomes push adaptation down to 54.1% while inflating rollback frequency to 41%, a signature of conservative filtering rather than better discrimination. Rollback frequency is not a false-positive rate. Decision errors are measured separately in the mutation audit.

J Hidden-Input Generalization

Table 13 evaluates 300 update episodes on Lifelong AgentBench with SkillRL. For each behavior cluster, one paraphrase is exposed while three sibling paraphrases and ten old tasks are withheld from synthesis, execution, and repair. Full SRT attains 96.1% exposed pass and 90.3% hidden pass, versus 97.2% and 71.0% for snapshot-based exact matching. Snapshot matching wins on the exposed inputs it memorizes, yet loses 26.2 points when the inputs change, whereas SRT loses only 5.8. The predicate-only variant retains most of the hidden-input gain.

K Mutation Audit

Table 14 reports the independent 600-update mutation audit on Lifelong AgentBench with SkillRL. Harmful edits cover over-specialization, tool substitution, and dependency removal. Benign ed-

Table 13: Generalization beyond exposed examples over 300 update episodes. Three-seed means with standard deviations as subscripts.

Method	Exposed	Hidden	Gap (pp)	Held-out
Snapshot + exact match	$97.2_{\pm 0.3}$	$71.0_{\pm 0.6}$	$26.2_{\pm 0.5}$	$52.8_{\pm 0.5}$
SRT without paraphrases	$95.6_{\pm 0.4}$	$81.7_{\pm 0.5}$	$13.9_{\pm 0.5}$	$57.0_{\pm 0.4}$
Predicate-only repair	$94.8_{\pm 0.4}$	$88.6_{\pm 0.4}$	$6.2_{\pm 0.4}$	$58.5_{\pm 0.4}$
Full SRT	$\mathbf{96.1}_{\pm 0.3}$	$\mathbf{90.3}_{\pm 0.4}$	$\mathbf{5.8}_{\pm 0.3}$	$\mathbf{59.0}_{\pm 0.4}$

Pass rates and held-out old-task accuracy are percentages. Hidden siblings and audit tasks are excluded from synthesis, execution, and repair.

Table 14: Independent mutation audit of 600 candidate updates (300 harmful and 300 benign or beneficial edits). Three-seed means with standard deviations as subscripts.

Method	F1 ↑	False rollback ↓	Missed regression ↓
Historical failure-set reject	$0.78_{\pm 0.02}$	$0.18_{\pm 0.02}$	$0.24_{\pm 0.02}$
SelfVerify-Once	$0.81_{\pm 0.02}$	$0.16_{\pm 0.02}$	$0.21_{\pm 0.02}$
Outcome-only SRT	$0.83_{\pm 0.02}$	$0.13_{\pm 0.01}$	$0.20_{\pm 0.02}$
Full SRT	$\mathbf{0.90}_{\pm 0.01}$	$\mathbf{0.08}_{\pm 0.01}$	$\mathbf{0.11}_{\pm 0.01}$

Rates are reported as fractions. Audit examples are excluded from synthesis and repair.

Table 15: Steady-state cost versus library size, with one targeted skill per round.

Skills	Eager (%)	Lazy (%)	Lazy synthesis (%)
10	13.4	13.2	1.8
100	15.8	14.1	2.2
200	19.7	14.8	2.6
400	27.6	15.6	3.1

The reported stream length is 1,000 rounds.

its include behavior-preserving refactors, legitimate bug fixes, and equivalent tool replacements. Program-decidable tasks use deterministic verification, and semantic cases use two independent judges with human arbitration. Full SRT reaches F1 0.90 with false rollback 0.08 and missed regression 0.11.

L Synthesis and Feedback Interfaces

Test synthesis. The synthesizer receives skill source, successful trajectories, failed trajectories, and the requested suite size, and returns a JSON array whose entries contain `input`, `assertion_type`, `predicate`, and `severity`. Assertion types are outcome, trace, and negative. Severity is critical or warn. The prompt requests paraphrastic task inputs and at least one negative assertion derived from failures. Outcome assertions specify task-appropriate equivalence rather than universal exact match.

Repair feedback. The repair interface supplies the candidate, failing input, violated predicate, observed trajectory, and the motivating new task. The full condition additionally supplies the original satisfying behavior, and the predicate-only control withholds the expected answer so that the hidden-input study can test whether direct answer exposure explains the improvement.

Semantic judgment. For an outcome predicate requiring semantic comparison, the judge receives the task and two outputs and returns an equivalence verdict. Formatting changes, ordering of independent items, and lexical paraphrases are allowed when they preserve the answer. Numeric changes and omitted required operations are not. Trace and negative predicates check these properties separately.

M Suite Size, Threshold, and Library Scaling

Table 15 reports the steady-state cost of eager and lazy refresh across library size, complementing the maintenance study in Section 5.4. Figure 6 separates the roles of suite size and admission threshold.

CPI climbs from 116.0 at four tests to 119.9 at eight, then plateaus at 119.8 at twelve. Growth beyond L=8 is negligible. Raising the critical-fraction threshold from 0.3 to 0.7 increases erosion from 5.8 to 9.1 points, consistent with admitting more borderline candidates.

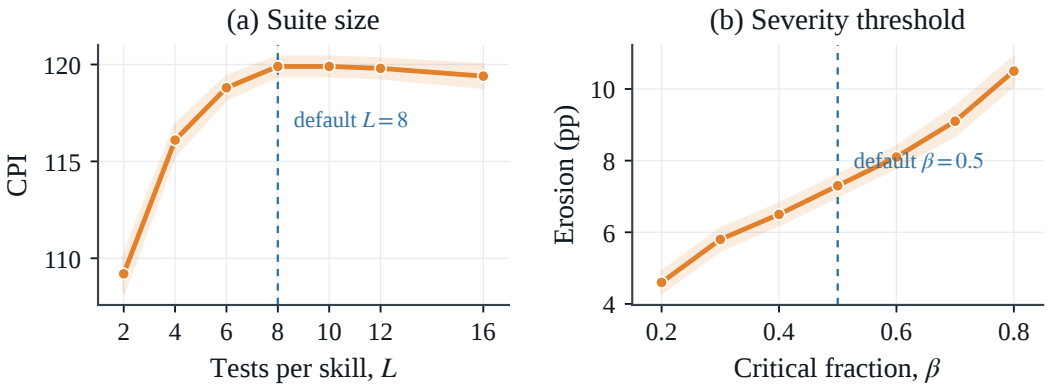


Figure 6: **Separate roles of suite size and admission threshold.** (a) CPI versus suite size. (b) Erosion versus the critical-fraction threshold. Dashed lines mark the defaults $L = 8$ and $\beta = 0.5$.

N Policy–Synthesizer Pairing

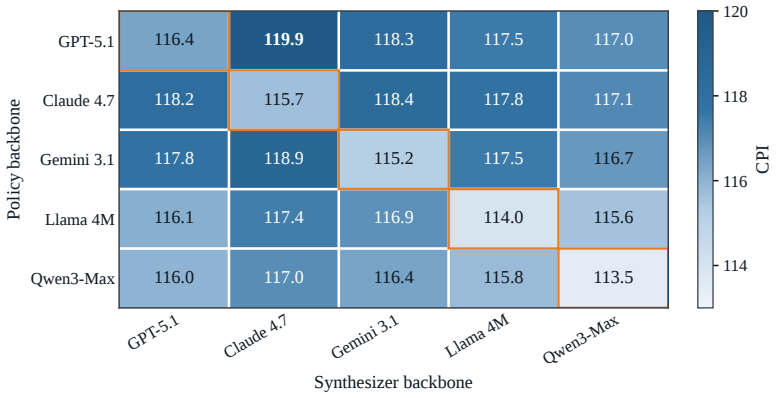


Figure 7: **Absolute CPI across policy–synthesizer pairs.** Rows fix the policy and columns vary the synthesizer. The same-backbone diagonal is outlined in orange and is the weakest configuration in every row. The highest value 119.9 pairs GPT-5.1 (policy) with Claude Opus 4.7 (synthesizer).

O Controller Actions and Failure Categories

Figure 8 separates controller action frequencies from manually categorized failures. Outcome-only checks can miss a dropped preprocessing step or an over-narrow retrieval trigger, whereas trace and negative assertions expose these changes before admission.

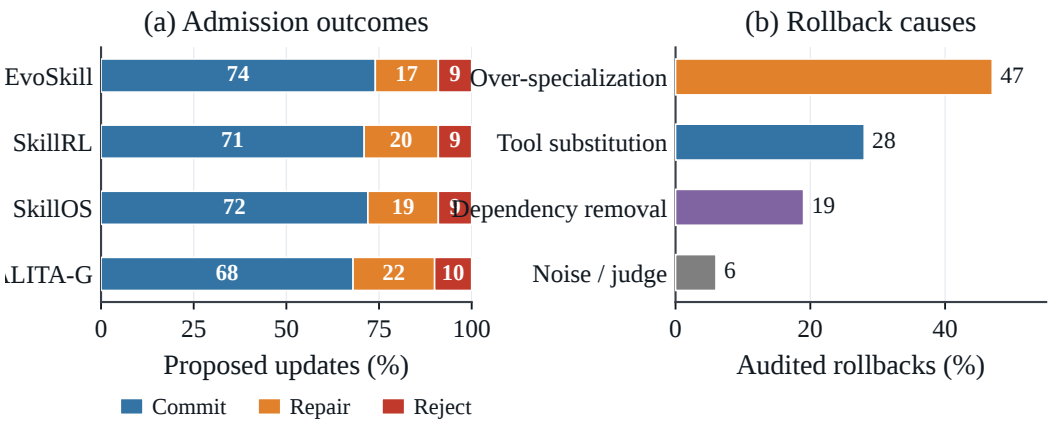


Figure 8: **Decisions and observed failure types.** (a) Action shares for four operators on Lifelong AgentBench. (b) Causes in 200 manually inspected rolled-back updates.

Illustrative failure and repair. The OfficeQA Pro example complements the generic dropped-dependency illustration in Figure 2. A citation-checking candidate narrows the trigger and adds a confidence gate. A trace assertion detects the skipped citation check on an older input, and a negative assertion detects an unsupported answer.

P Human Review Protocol

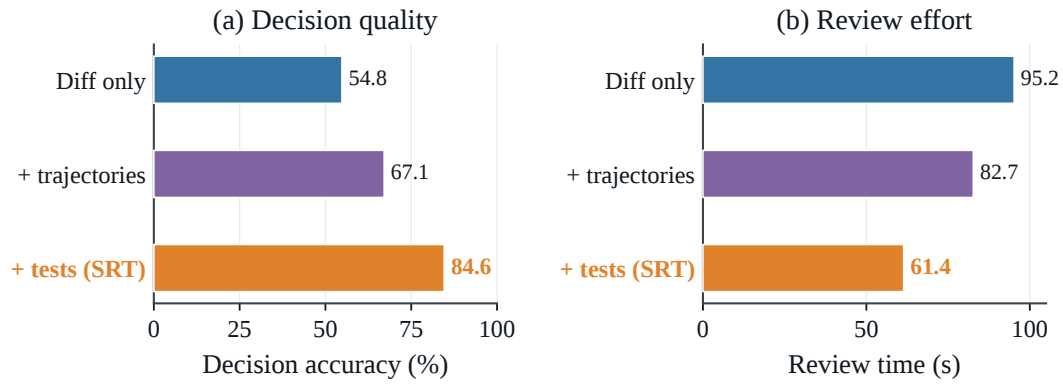


Figure 9: **Generated tests support update review.** Accuracy and time against an independent execution audit under three information conditions (1,080 judgments).

The three information conditions in Figure 9 use five calibration trials and Latin-square counterbalancing. Reviewers assess updates against an independent execution audit rather than the SRT controller's decision. The earlier 12-participant pilot measured agreement with SRT rather than audit accuracy and is analyzed separately.