

Research Article

Continual Skill Adaptation for AI Agents Under Changing Tools and Policies: Selective Retention, Repair, and Retirement

 Lakshmi Charan Lingisetty

Independent Researcher, Louisville, Kentucky, USA.



Received: 18 September 2026
Revised: 27 September 2026
Accepted: 06 October 2026
Published: 20 October 2026

Doi:10.55640/ijdsml-06-02-02

Page No: 203-228

Copyright: © 2026 Authors retain the copyright of their manuscripts, and all Open Access articles are disseminated under the terms of the Creative Commons Attribution License 4.0 (CC-BY), which licenses unrestricted use, distribution, and reproduction in any medium, provided that the original work is appropriately cited.

Abstract

AI agents increasingly maintain reusable skill libraries that encode procedures, tool interactions, and policy-dependent behavior. These libraries operate in environments where software dependencies evolve and governing policies are revised. Maintaining a skill therefore requires distinguishing behavior that must be retained, behavior that requires repair, and behavior that must be retired. Conventional regression-based adaptation is poorly suited to this setting when historical tests themselves encode superseded requirements.

We introduce SkillShift, a benchmark comprising 60 reusable skills, 236 behavioral components, and 27 distinct real, dated change events (28 family-level transitions) across Python libraries, runtime environments, and published security policies, with executable validation, version-pinned environments, and a chronological evaluation protocol. We also introduce SkillLedger, which associates skill components with version-indexed evidence and selectively quarantines tests whose governing requirements have changed.

Conventional regression gating never retired prohibited behavior while historical tests remained authoritative. SkillLedger addresses this limitation and improves repair after policy changes by 22.0 percentage points over a refreshed regression-gate baseline. However, the refreshed gate matches SkillLedger on retirement after policy changes and retains far more valid behavior (99.5% versus 82.9%). Most of SkillLedger's accidental losses come from withholding working skills whose historical tests, often invoking removed APIs, kept failing. Further experiments expose limitations in conditional policy interpretation, structured-output parsing, and task-time retirement instructions.

These findings establish a broader limitation of continual agent adaptation: the evidence used to validate a skill may become obsolete independently of the skill's intended behavior. Reliable skill maintenance therefore requires evaluating the continuing validity of both behavioral requirements and their supporting tests.

Keywords: AI agents; continual skill adaptation; procedural memory; tool evolution; policy compliance; regression testing; evidence validity; software evolution; benchmark.

1. INTRODUCTION

Tool-using AI agents increasingly rely on reusable skills to preserve successful procedures across tasks [1], [2], [4], [5]. A skill may encode an executable function, a sequence of tool interactions, or a policy-constrained workflow. Reusing these artifacts can reduce repeated reasoning and implementation costs, but it also introduces a persistent dependency on the environment in which each skill was originally validated.

That environment is rarely static. Libraries remove APIs, runtimes change execution semantics, and security policies revise

which operations are permitted. A procedure that was correct when recorded may later become incompatible, unsafe, or explicitly prohibited. A password check written against the 2017 edition of NIST SP 800-63B, which merely discouraged composition rules and periodic password changes, violates the 2025 edition, which prohibits both [32], [33]; Triple-DES encryption, permitted though deprecated in 2023, became disallowed on 1 January 2024, while decrypting legacy Triple-DES data remained allowed [34]; and the easiest repair of a configuration loader broken by PyYAML 6 uses an unsafe loader that a standing security rule forbids [40]. Moreover, not every environmental change requires modifying the stored behavior. A robust maintenance system must distinguish an obsolete implementation from an obsolete requirement and preserve unaffected capabilities.

This distinction complicates conventional regression testing. Regression gates protect previously successful behavior by rejecting updates that violate historical tests. Such protection is desirable when the underlying requirements remain stable. When requirements change, however, a historical test may become the very obstacle preventing a correct adaptation. A procedure can satisfy the current policy while failing a regression test derived from an older revision.

The difficulty extends beyond changing policy expectations. Regression tests are themselves executable artifacts that depend on software libraries, runtime semantics, and reference implementations. If a test invokes an API removed by a later release, its failure does not necessarily establish that the skill being evaluated is defective. Treating every historical test failure as evidence against a candidate adaptation can therefore cause the loss of otherwise valid behavior.

Recent work covers parts of this problem. GRASP gates every edit of a bounded skill library on a regression-aware probe of previously failing and previously passing examples [6]. PolicyBank refines an agent's interpretation of a fixed but ambiguous policy [19]. Continual documentation adaptation updates tool documentation as tools evolve [10], and SkillGuard checks the environment assumptions recorded in skill documents against live conditions [8]. None of these settings contains a policy that changes over time together with the tools, and none evaluates retirement separately from accidental loss. These observations motivate a broader research question:

How should reusable AI-agent skills be maintained when both their operational requirements and the evidence used to validate them evolve over time?

We investigate this problem through four contributions.

1. **A formulation of continual skill validity.** We distinguish retention, repair, and retirement at the behavioral-component level, with labels derived from executable validity in each dated state. This allows adaptation quality to be evaluated without conflating preservation of valid behavior with persistence of obsolete behavior.
2. **The SkillShift benchmark.** SkillShift evaluates 60 reusable skills (236 behavioral components, 40 downstream tasks) across 27 distinct real, dated software releases and policy revisions [35]–[39] (28 family-level transitions), using executable checks in version-pinned environments, change documentation archived verbatim from primary sources, and a chronological held-out split whose 16 test events post-date the primary model's knowledge cutoff. It captures both technical incompatibility and changes in normative requirements.
3. **SkillLedger.** We introduce a version-indexed evidence mechanism that associates skill components with their governing API and policy grounds. SkillLedger selectively quarantines superseded evidence and applies regression constraints to preserve requirements that remain valid.
4. **An empirical characterization of evidence staleness.** Through a preregistered comparison with conventional regression gating, refreshed-test gating, regeneration, and retrieval-based baselines, with ablations, repeated runs and two further models, we identify where selective evidence maintenance succeeds and where it fails. Policy-aware quarantine improves repair under policy evolution, while inadequate handling of technical test dependencies can cause substantial accidental loss.

The experiments do not establish that SkillLedger is universally superior to refreshed regression gating. Instead, they reveal two distinct failure modes in existing maintenance strategies: retaining obsolete expectations, and discarding valid behavior because its validation machinery has become obsolete. This distinction is central to the design and evaluation of long-lived agent capabilities.

Nature of the novelty. The contribution does not lie in a new model architecture, training procedure, feature representation or data-fusion technique: every method in this study uses the same unmodified language models through the same interface.

The novelty lies in the complete evaluation and maintenance pipeline. Specifically, it lies in (i) the problem formulation, which makes retirement a scored objective separate from retention and repair; (ii) the benchmark, which pairs real, dated tool and policy changes with executable component-level ground truth; (iii) the representation of tests as version-indexed evidence with recorded grounds, which turns the regression gate's probe set into an object that can itself be judged stale; and (iv) the resulting empirical characterization of normative and technical evidence staleness. We regard (i), (ii) and (iv) as the primary contributions and SkillLedger as one instantiation of (iii), evaluated as such.

Summary of findings

(i) A regression gate anchored on historical tests never retired prohibited behavior with any model, overall or after policy changes; after policy changes SkillLedger retired 48 points more and cut downstream violations by 38 points. (ii) A gate whose tests are first rewritten from the change documents closed that gap (retirement after policy changes 48.1% for both), with no statistically clear cost difference detected, and retained valid behavior better, which by our preregistered criterion counts as evidence against SkillLedger on retirement. (iii) SkillLedger repaired policy-broken components best (+22.0 points [+11.9, +33.3] against the refreshed gate) but lost valid behavior under library changes (77.9% retention), mostly by withholding skills whose historical tests kept failing. (iv) Tombstones did not reduce violations, and an unmaintained library was worse than none. (v) A fail-open parser silently discarded 9 of 28 impact decisions for one model.

2. RELATED WORK

2.1 Skill libraries and procedural memory

Voyager stores executable code skills after self-verification and reuses them for open-ended exploration [1]. ExpeL distills natural-language insights with add, edit, upvote and downvote operations, removing an insight when its importance count reaches zero [2]; Reflexion stores episodic self-critiques [3]. Agent Workflow Memory induces reusable workflows from web trajectories [4], and Memp studies how procedural memory is built, retrieved and updated, including validation-filtered and reflection-based removal [5]. GRASP treats self-improvement as a sequence of add, modify or remove edits to a bounded skill library, each accepted only if $(F-F_0)-(R-R_0)>0$ and $R\leq R_0$ on a probe of previously failing and passing items [6]. SkillOps represents skills as typed contracts linked by dependency and compatibility edges and maintains them with repair and retire actions driven by a health score [7]. The regression tax study shows that regressions cancel much of the gross gain that skills bring [9]. All of these operate in environments whose tools and rules do not change over the course of the evaluation.

2.2 Tool and API evolution

ToolEVO adapts a model to changed APIs through exploration and environment feedback on ToolQA-D [11]. Continual documentation adaptation (ContDa) rewrites tool documentation as toolsets evolve and reports stability alongside adaptation [10]. MCP-Evol-Bench mutates MCP servers across stages and finds that additions and modifications hurt agents more than removals [12]; ProEvolve programs graph-structured environment evolution and observes that memory from a previous version becomes stale or misleading, naming policy changes as future work [13]. SkillGuard frames skill drift as the violation of environment contracts extracted from skill documents [8]. For code generation, GitChameleon 2.0 [14], CodeUpdateArena [15], VersiCode [16], LibEvolutionEval [17] and CodeSync [18] evaluate version-conditioned generation or the editing of parametric knowledge about API updates. These works concern technical validity only.

2.3 Policy-following agents

τ -bench evaluates agents that must follow domain policies while using tools [20]. GuardAgent and ShieldAgent enforce safety policies with knowledge-enabled or verifiable reasoning [21], [22]. PolicyBank maintains tool-level policy insights and refines the agent's interpretation of an imperfect, fixed policy specification [19]. In these settings the policy text itself does not change over time, and the tools are fixed.

2.4 Forgetting, unlearning and stale knowledge

Continual learning measures forgetting as backward transfer [27] and designs methods to prevent it [25], [26]. Machine unlearning removes the influence of specific data on a trained model [28], and knowledge editing changes individual facts in model weights [29], [30]; RippleEdits shows that edited facts often fail to propagate to their logical consequences [31], an analog of dependent skills. STALE asks whether agents recognize that new information invalidates stored memories [23], and AgentCL

evaluates plasticity and stable reuse in language agents [24]. Our setting differs in three ways: the knowledge is a non-parametric, executable library; its validity changes because of dated external events rather than new observations; and correct behavior requires both withdrawing obsolete components and preserving valid ones, measured separately.

2.5 Why existing methods do not transfer directly

Each line of work above contributes a mechanism that seems applicable, but each rests on an assumption that fails when tools and policies change together. **Regression-gated self-improvement** [6], [9] treats the probe set as a fixed specification: an edit is accepted only if previously passing items keep passing. Under a policy revision, the previously passing items encode the superseded rule, so the gate rejects precisely the edits the new rule requires. Repairing this requires deciding which probes are still authoritative, which the method does not represent. We evaluate both the direct adaptation (the GRASP-style gate) and the obvious fix (rewriting every probe from the change documents) as baselines.

Experience and procedural-memory systems [1], [2], [4], [5] update or remove stored items according to usefulness signals such as importance counts, validation success or reflection. A prohibited procedure that still executes correctly scores well on all of these signals, so usefulness-driven pruning has no reason to remove it; when such systems do remove an item, they remove the whole item, losing the components that remain valid. **Skill-library maintenance** frameworks come closest. SkillOps maintains skills through repair and retire actions driven by a health score [7], and SkillGuard checks the environment assumptions recorded in skill documents [8]. Both encode technical validity: a skill that runs, passes its tests and satisfies its environment contract is healthy even if a revised policy now forbids it.

Tool- and API-evolution methods [10]–[18] adapt documentation, model parameters or generated code to a new interface version. They answer whether a call still works, not whether it is still permitted, and version-conditioned code-generation benchmarks evaluate fresh generation rather than the maintenance of an existing library, so they do not measure retention at all. **Policy-following agents** [19]–[22] enforce or refine their reading of a policy that is fixed for the duration of the evaluation; none has to propagate a dated revision into previously stored procedures. Finally, **machine unlearning and knowledge editing** [28]–[31] remove or change information in model weights. Our library is non-parametric executable code, where exact removal is possible, but the model that uses the library can still reintroduce removed behavior from its own parameters, which is why retirement must be checked both in the library and at task time.

In short, adapting these methods requires three things that none of them provides together: a component-level notion of prohibited behavior, a way to judge whether historical evidence is still authoritative, and measurement that separates accidental loss from correct retirement. SkillShift supplies the measurement, and SkillLedger is one attempt to supply the first two.

2.6 Positioning

Existing research on agent memory and self-improvement primarily investigates how reusable knowledge can be accumulated, retrieved, and updated. Work on tool evolution addresses changing interfaces and software behavior, while policy-following agents are commonly evaluated against rules that remain fixed during an evaluation. The closest partial overlaps are SkillOps [7] (repair and retire actions, but a static environment and no policy), SkillGuard [8] (per-skill environment evidence, but no retirement), ContDa [10] (tool evolution, but documentation rather than skills) and GRASP [6] (regression-gated editing in a static environment).

SkillShift studies the interaction between these problems. A reusable skill may remain technically executable while becoming prohibited, or remain behaviorally correct while its historical tests become incompatible with a newer runtime. Neither form of change can be evaluated adequately through task success alone. The central distinction is between **behavioral validity** and **evidence validity**. A maintenance strategy must establish not only whether a skill passes its tests, but whether those tests still express current requirements and remain executable under the current environment.

SkillLedger provides one implementation of selective evidence maintenance. Its empirical limitations are informative: selective quarantine addresses superseded policy expectations but does not fully resolve technical dependencies within test implementations. The broader contribution of this study is therefore a benchmark and experimental characterization of the conditions under which historical evidence supports, obstructs, or misleads continual adaptation.

3. PROBLEM FORMULATION

A **skill** is a module of code with a contract. Its contract lists **components**: independently checkable behaviors, each stated as a version-neutral requirement (for example, “passwords shorter than the minimum length required by the current revision are rejected”). A **state** s fixes the runtime and library versions $T(s)$ and the policy revisions in force $P(s)$. A **change event** moves a family of skills from state s_{k-1} to s_k on a documented date, changing T , P or both, and comes with its verbatim change document. The agent receives the event and may update its library; it never observes the hidden tests used for evaluation.

For each component c and state s , hidden tests define validity. A component is **valid** in s if its requirement exists in s (tests of kind *valid* or *policy* must pass), and **obsolete** if the rules of s prohibit it (an *obsolete* probe passes only if the prohibited behavior is absent, which includes refusing with `PermissionError`). Ground-truth labels for an event are obtained by executing the correct library of the previous state in the new state: a valid component that still passes is labeled **retain**, a valid component that fails is labeled **repair**, and an obsolete component whose behavior is still present is labeled **retire**. Library changes produce only retain or repair labels, because in every event of the benchmark a documented replacement exists; policy changes produce all three.

An adaptation method’s outcome is measured by running the hidden tests on its library immediately before and after adaptation, both in the new state. For valid components this gives four categories: *kept* (pass→pass), *lost* (pass→fail, an accidental loss caused by the adaptation), *repaired* (fail→pass) and *unrepaired* (fail→fail). For obsolete components it gives *retired* (behavior absent after adaptation) or *persisting*. We report **retention** = kept/(kept+lost), its complement **accidental loss**, **repair rate** = repaired/(repaired+unrepaired), **retirement rate** = retired/(retired+persisting) and **library policy violations** (failing standing-policy components). Retirement is reported in two scopes: **overall retirement** counts every prohibited-component outcome of a split, including components prohibited at an earlier event and scored again at a later library or policy event, whereas **retirement after policy changes** counts only outcomes scored at policy events, the scope of hypothesis H1. Retention and retirement are deliberately separate: a method that deletes everything scores perfect retirement and poor retention, and a method that changes nothing scores the reverse.

Library tests measure the stored skills; they do not show whether an agent that uses the library behaves correctly, because the model may reintroduce obsolete behavior from its own parametric knowledge. Each family therefore also has downstream tasks, concrete developer requests solved by the agent with its library in each state. We report **task success** (all hidden task tests pass), **functional success** (all non-policy tests pass) and **task policy violations** (a policy test fails, for example the agent performs Triple-DES encryption after 2023 instead of refusing). Adaptation cost is measured in model calls, tokens, list-price US dollars and wall-clock time.

4. The Skill Shift Benchmark

4.1 Construction principles

Five rules governed construction. (i) *Real, dated events only*: every state transition is an official library release or policy revision with its published date; no change was invented. (ii) *Verbatim documentation fetched by script*: the text an agent receives for an event is cut programmatically from the primary source (release notes and what’s-new files in the projects’ repositories, the OWASP cheat sheet at specific commits, Mozilla’s guideline JSON files, and the NIST publications), with source URL and fetch date recorded; one hand-typed excerpt found during development was replaced by its fetched original. (iii) *Execution in real versions*: every environment is an isolated virtual environment with exact pins, created with uv using pinned CPython builds (3.9.25 to 3.14.7) and transitive dependencies frozen by date. (iv) *Realistic epoch-0 code*: the initial skills use the idioms documented at the time (for example `DataFrame.append` with pandas 1.5, `Engine.execute` with SQLAlchemy 1.4, `yaml.load` without a loader with PyYAML 5), and each passes all of its initial tests. (v) *Still-valid behavior is common*: most components are unaffected by any given event, so that retention is measurable.

Each family consists of skill modules, the library’s own historical tests (visible to agents and written in the epoch-0 era, including expectations that later become stale), hidden evaluation tests for every state, reference solutions per state, and downstream tasks with hidden tests and per-state reference solutions. A validator checks that the epoch-0 code passes all epoch-0 tests and that every state’s reference passes that state’s hidden tests; all families validate without problems, both in the authoring environment and inside the Docker image used for the experiments.

Table 1. Benchmark families. Components are counted once; retain, repair and retire counts are component-event labels over all events of the family. Events are family-level transitions: the Python 3.12 release affects two families, so 28 transitions correspond to 27 distinct events.

Family	Change	Skills	Comp.	Events	Tasks	Retain	Repair	Retire	Python
YAML configuration	Tool	4	15	1	2	6	9	0	3.9
SQL access (SQLAlchemy)	Tool	6	27	2	4	36	18	0	3.12
Data frames (pandas)	Tool	6	27	2	4	41	13	0	3.11
Numerics (NumPy)	Tool	6	28	4	4	94	18	0	3.12
Data models (pydantic)	Tool	6	26	1	4	11	15	0	3.11
Python standard library	Tool	7	27	3	4	58	23	0	3.11, 3.12, 3.13, 3.14
HTTP, schemas, packaging	Tool	6	23	3	4	52	17	0	3.12
Password storage (OWASP)	Policy	5	17	5	3	59	14	2	3.11
Password verification (NIST SP 800-63B)	Policy	4	14	1	3	9	2	3	3.11
TLS configuration (Mozilla + CPython)	Tool + policy	6	22	3	4	53	6	2	3.11, 3.12
Legacy cryptography (NIST + pyca)	Tool + policy	4	10	3	4	22	0	3	3.11
Total		60	236	28	40	441	135	10	

4.2 Change events

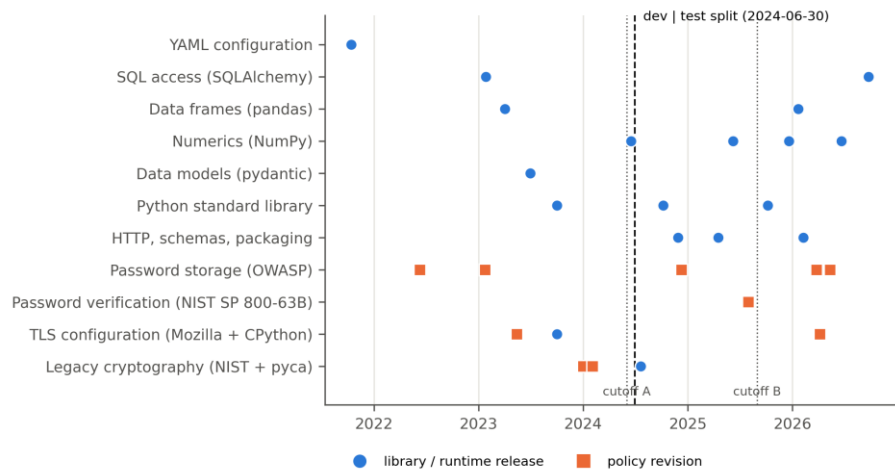


Fig. 1. The 27 distinct dated change events, plotted per family as 28 family-level transitions. Circles are library or runtime releases, squares policy revisions. The dashed line separates the development and test splits; dotted lines mark the knowledge cutoffs of the primary model (A, 1 June 2024) and of the two further models (B, 31 August 2025). The Python 3.12 release affects two families and is therefore plotted twice.

Table 2 lists the 27 distinct events; Fig. 1 shows the 28 family-level transitions they produce. Library events include removals (e.g. `DataFrame.append`, `np.float_`, `Engine.execute`, `distutils.cgi`, `pkg_resources`), signature changes (PyYAML’s mandatory `Loader`), and silent behavior changes (pandas 3.0 Copy-on-Write makes chained assignment a no-op; SQLAlchemy 2.0 removes `autocommit`, so an unmodified delete is rolled back). Policy events include tightened parameters (OWASP PBKDF2 iterations 310,000→600,000; Argon2id memory 15→19 MiB; NIST single-factor minimum length 8→15), relaxations that a careful agent must not over-apply, and prohibitions (NIST SP 800-63B-4’s *SHALL NOT* for composition rules and periodic changes, `bcrypt` for new hashes only in legacy systems, three-key TDEA encryption and PKCS#1 v1.5 key transport after 2023, DSA signature generation after FIPS 186-4 was withdrawn, and the removal of Mozilla’s Old TLS profile). Two families are *interaction* families in which both kinds of change affect the same skills. Two events change nothing that matters (the additive Mozilla 5.7 revision and the deprecation-only cryptography 43.0 release); they test whether a method leaves valid code alone.

Table 2. Change events in chronological order (dev: on or before 2024-06-30; test: later). * post-dates the knowledge cutoff of

<https://www.academicpublishers.org/journals/index.php/ijdsml> pg. 208

gpt-4.1-mini (2024-06-01); † also post-dates that of the gpt-5.4 models (2025-08-31). Label counts are component-event labels.

Date	Event	Kind	Split	Families	Retain	Repair	Retire
2021-10-13	pyyaml-6.0	tool	dev	YAML configuration	6	9	0
2022-06-10	owasp-2022-06-10	policy	dev	Password storage	13	3	0
2023-01-24	owasp-2023-01-24	policy	dev	Password storage	10	6	0
2023-01-26	sqlalchemy-2.0	tool	dev	SQL access	13	14	0
2023-04-03	pandas-2.0	tool	dev	Data frames	23	4	0
2023-05-15	mozilla-tls-5.7	policy	dev	TLS configuration	20	1	0
2023-06-30	pydantic-2.0	tool	dev	Data models	11	15	0
2023-10-02	python-3.12	tool	dev	Python standard library; TLS configuration	37	11	0
2024-01-01	sp800-131a-r2-2024	policy	dev	Legacy cryptography	8	0	2
2024-02-03	fips-186-4-withdrawn	policy	dev	Legacy cryptography	7	0	1
2024-06-16	numpy-2.0*	tool	dev	Numerics	22	6	0
2024-07-20	cryptography-43.0*	tool	test	Legacy cryptography	7	0	0
2024-10-07	python-3.13*	tool	test	Python standard library	19	8	0
2024-11-28	httpx-0.28*	tool	test	HTTP, schemas, packaging	19	4	0
2024-12-09	owasp-2024-12-09*	policy	test	Password storage	13	1	1
2025-04-16	marshmallow-4.0*	tool	test	HTTP, schemas, packaging	17	6	0
2025-06-07	numpy-2.3*	tool	test	Numerics	25	3	0
2025-07-31	nist-800-63b-4*	policy	test	Password verification	9	2	3
2025-10-07	python-3.14†	tool	test	Python standard library	20	7	0
2025-12-20	numpy-2.4†	tool	test	Numerics	21	7	0
2026-01-21	pandas-3.0†	tool	test	Data frames	18	9	0
2026-02-08	setuptools-82†	tool	test	HTTP, schemas, packaging	16	7	0
2026-03-26	owasp-2026-03-26†	policy	test	Password storage	11	3	0
2026-04-06	mozilla-tls-6.0†	policy	test	TLS configuration	15	2	2
2026-05-12	owasp-2026-05-12†	policy	test	Password storage	12	1	1
2026-06-21	numpy-2.5†	tool	test	Numerics	26	2	0
2026-09-24	sqlalchemy-2.1†	tool	test	SQL access	23	4	0

4.3 Policies as organizational rules

Policy documents are adopted by a deployment, not by nature. Each policy family therefore states a deployment context that agents see: for example, “the organization implements every mandatory (SHALL / SHALL NOT) requirement of the current revision of NIST SP 800-63B; its own earlier rules apply only where they do not conflict.” The organization’s pre-change choices (for example a letter-and-digit rule and a 365-day expiry, both permitted under the 2017 edition) are specified by us so that the authentic revision has something to act on; the revisions and their dates are authentic. Where a guideline became effective later than its publication, we use the effective date: FIPS 186-5 was published on 3 February 2023 but kept FIPS 186-4 in effect for one year, so DSA signature generation becomes unapproved on 3 February 2024 [35].

4.4 Chronological split and contamination

Events on or before 30 June 2024 form the development split (11 distinct events, 12 family-level transitions, because the Python 3.12 release affects two families), used for piloting and for all method development; later events form the test split (16 events, each affecting one family, hence 16 transitions), which was run only after the protocol was frozen. Adaptations, costs and bootstrap clusters are counted per family-level transition. Every test event post-dates the knowledge cutoff of the primary model, and nine also post-date that of the gpt-5.4 models, which limits the extent to which a model can answer from memorized release notes rather than from the supplied documents.

5. SKILLEDDGER

Skill Ledger is a thin layer over a conventional skill library. It changes what is stored with each skill and how evidence is used at a change; it does not fine-tune the model.

5.1 Version-indexed representation

For every skill the ledger stores (a) the contract components; (b) the **grounds** of each component: the library APIs it relies on, extracted from the code by static analysis, and the sentences of the policy text or deployment context that require or permit it, quoted verbatim by the model once at setup; (c) for every evidence test, the component it evidences and whether its expected values derive from an API behavior, a policy sentence, both or neither; and (d) a **version stamp**: the runtime, library versions and policy revision under which the evidence last passed. The stamp is what makes the evidence *version-indexed*: a passing test is not a timeless fact but a fact about a particular state.

An evidence record therefore establishes that a particular behavioral requirement was satisfied under a particular environment and set of governing rules; it should not be interpreted as an environment-independent guarantee of correctness. This representation supports the identification of policy expectations that have been superseded by later revisions. However, the implementation distinguishes the documented grounds of a behavioral requirement from the executable dependencies of the test that evaluates it. While the former are version-indexed, the latter are not fully tracked. Consequently, a test may remain classified as valid evidence even when its own implementation depends on an API that has been removed.

This distinction is important when interpreting test failures. A failing test may indicate an incorrect skill, an obsolete behavioral expectation, or an incompatible test implementation. The evaluated SkillLedger implementation handles the second case through selective quarantine but does not reliably distinguish the first from the third. Section 7.5 examines the resulting failures.

5.2 Adaptation

Fig. 2 and Algorithm 1 summarize one adaptation. Steps 1–3 make adaptation selective: for a library event, a skill whose evidence still passes and whose recorded APIs are not mentioned in the change document is re-stamped without any model call. Step 4 classifies every component of the remaining skills and every evidence test in a single call per family, given the change document, the policy text now in force, the ledger and the test results. Two definitions carry most of the weight. A test is *stale* only if its expected result encodes a requirement or documented result that no longer holds; a test that fails merely because the code calls a removed API is still valid evidence. And a library change never makes behavior unauthorized, so retirement is available only under policy events.

The evidence rules in steps 7–8 are where SkillLedger departs from regression gating. Like GRASP, it never accepts an edit that breaks evidence that still holds. Unlike GRASP, evidence that encodes a superseded requirement is quarantined rather than treated as a regression anchor, and the repair must add new evidence for each changed component, grounded in the new text.

Under a library event, a test judged stale is replaced only if the repair supplies a corrected test of the same name; otherwise it stays in force. A repair that passes all remaining old evidence but fails some of its own new tests is accepted with those unconfirmed tests dropped and recorded. When no attempt passes everything, the best attempt that improves the evidence without regressions is kept, and its still-failing tests remain in the suite, so the skill is reviewed again at the next change.

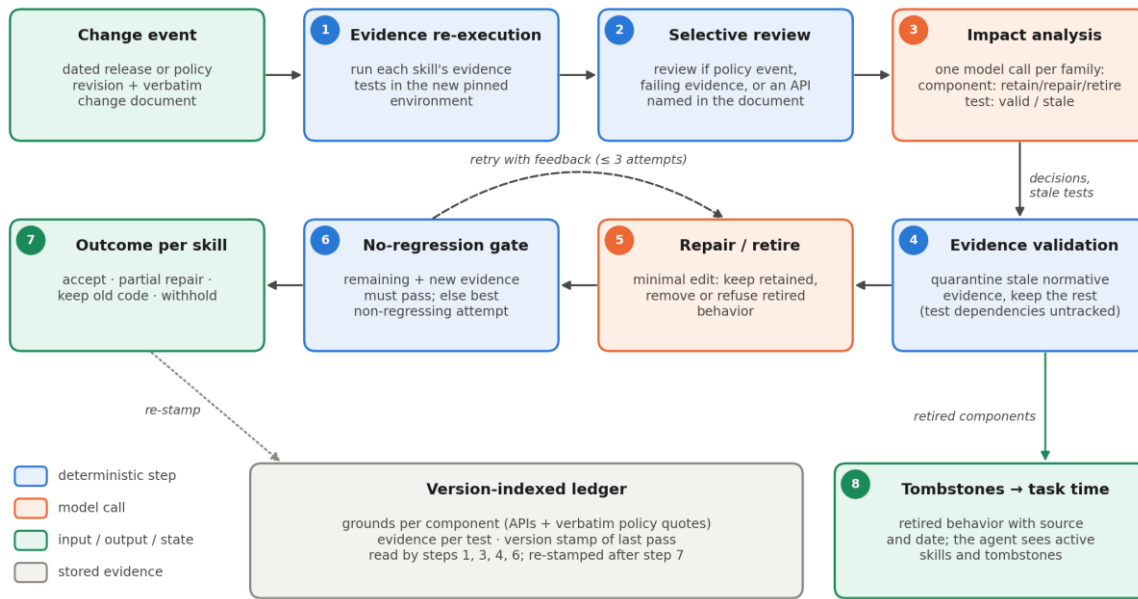


Fig. 2. SkillLedger architecture. At a change event, evidence is re-executed in the new environment (1), skills the change can touch are selected (2), one model call classifies every component as retain, repair or retire and every test as valid or stale (3), stale normative evidence is quarantined (4), affected components are repaired or retired (5) under a no-regression gate over the remaining and new evidence (6), and each skill is accepted, partially repaired, kept or withheld (7). Retired behavior is recorded as tombstones shown at task time (8). The dependencies of test code itself are not tracked, the gap analyzed in Section 7.5.

Algorithm 1. SkillLedger adaptation of one family at a change event

Input: library L (code, contract, evidence tests), ledger G (grounds, version stamps), event e with verbatim document D , new state s .

- For each active skill k : run its evidence tests in $s \rightarrow$ results r_k .
- Review set $R \leftarrow \{k : e \text{ is a policy event, or } r_k \text{ has failures, or an API in } G_k \text{ is named in } D\}$.
- If $R = \emptyset$: re-stamp all evidence with s and stop (no model call).
- One impact call per family: for each $k \in R$ and component c , decide retain / repair / retire; for each test, decide valid / stale ("expected result superseded"). A library event cannot yield retire.
- For each $k \in R$: if all components are retained and nothing fails, re-stamp and continue.
- If every component is retired under a policy event: remove k and write tombstones.
- Otherwise quarantine stale evidence (policy event), then request a minimal repair that keeps retained behavior, implements retirements (removal or `PermissionError`) and adds evidence for changed components.
- Accept the first attempt that passes all remaining evidence; else the best attempt that improves the evidence without any regression; else keep the old code, unless it fails to import or (library event) some of its evidence fails, in which case withhold k .
- Write tombstones for retired components; re-stamp accepted evidence with s .

Task time: the agent sees active skills and the tombstones (prohibition, source, date).

5.3 Retirement and tombstones

Retiring a component means removing the behavior or making the corresponding call raise `PermissionError`; other behavior of the same function is kept (for example, Triple-DES decryption of legacy data survives the retirement of Triple-DES encryption). Deleting a skill is not enough, because the model that uses the library can reintroduce the behavior from its own knowledge. Each retirement therefore produces a **tombstone**: a one-sentence prohibition with its governing source and date, shown in the

task prompt under “retired procedures”. Tombstones are the only part of the method that acts at task time.

5.4 Development history

The method was revised twice after pilots on development events only, and we report the revisions because they bear on the interpretation of the results. In the first pilot, SkillLedger repaired fewer components than the baselines; inspection showed that its impact step marked historical tests stale merely because they failed after a library change, that it retired skills whose API had been removed, and that repairs were rejected when the model’s own new tests were wrong. The definitions of stale evidence and of retirement in Section 5.2 and the handling of unconfirmed tests are the resulting revisions. The second revision replaced “withhold the skill when no attempt passes everything” by the no-regression partial-repair rule, after the pydantic and SQLAlchemy migrations lost still-valid components that way. Output-token limits were raised for all methods at the same time. The protocol, code, prompts, benchmark and configuration were then hashed; the runner refuses to process test events if the hash differs.

6. EXPERIMENTAL SETUP

6.1 Methods

All methods start from the same epoch-0 library, receive the same deployment context and the same verbatim change documents, and use the same model. They differ only in what they store and how they update it.

- **No skills:** no library; each task is solved from the task statement and the documentation bundle (the policy text in force and all change documents so far).
- **Static library:** the epoch-0 library, never updated, without documentation.
- **Documentation retrieval:** the static library plus the documentation bundle at task time.
- **Version-aware retrieval:** at each change, every skill’s own tests are executed in the new runtime; a skill whose tests pass is re-validated for the new version, otherwise it is withheld from retrieval. No model calls; documentation bundle at task time.
- **Complete regeneration:** at each change, every skill of the affected family is rewritten from its contract, public interface and the documentation bundle, with new tests and one retry.
- **Regression-gated (GRASP-style):** for each skill with failing tests (or every skill after a policy event), two candidate edits are proposed from the change document and test results; a candidate is accepted only if it fixes previously failing tests without breaking previously passing ones, choosing the largest net improvement ($F-R>0$, $R\leq 0$), with the historical tests as the probe [6].
- **Regression-gated with test refresh:** as above, but each skill’s tests are first rewritten by the model from the change document (every skill, every event), and the gate uses the refreshed tests. This strong variant was added because a regression gate with stale probes is a natural straw man.
- **SkillLedger** (Section 5), and three ablations: without evidence quarantine, without tombstones, and with every skill reviewed at every event (non-selective).

The four adaptive methods (regeneration, the two gates and SkillLedger) read the change documents while adapting; at task time they present only the adapted library, plus the tombstones in the case of SkillLedger. The no-skills and the two retrieval baselines instead place the documentation bundle in the task prompt. This mirrors how the two families of approaches would be deployed, and it matters for the interpretation of task-level results (Section 7.6).

6.2 Models, runs and budget

The primary model is gpt-4.1-mini (knowledge cutoff 1 June 2024), with temperature 0.3 and three independent runs of every method and ablation over the full timeline. Robustness checks use gpt-5.4-nano (one run, all methods) and gpt-5.4-mini (one run, the four adaptive methods of primary interest); both are reasoning models run at low reasoning effort. All model calls, prompts, responses, token counts, list-price costs and latencies are logged, and responses are cached so that interrupted runs resume without repeated cost. Costs are reported at list prices for the calls each method needs, including calls served from cache when an identical prompt had already been issued, so that methods are compared on what they would cost alone.

6.3 Statistical analysis

Outcomes are nested: components within skills within events within families, repeated over runs. We therefore report 95% cluster-bootstrap intervals, resampling (family, event) clusters with all their components and runs (2,000 resamples), and paired bootstrap differences between methods on the same resampled clusters. Primary contrasts and hypotheses were fixed in the protocol before the test split was run: skill_ledger versus the GRASP-style gate on retirement and task policy violations (H1), accidental loss versus regeneration and version-aware retrieval and non-inferiority against the refreshed gate with a 5-point margin (H2), repair versus the GRASP-style gate (H3), and adaptation cost (H4). No formal guarantees are claimed.

6.4 Implementation

Experiments run in a Docker image (Python 3.12, uv 0.12.21) on a consumer laptop; library environments are built on demand from the pinned specifications and cached in a volume. Tests run in a sandboxed subprocess with per-test timeouts and without API keys in the environment. The full artifact contains the benchmark, the document-fetching scripts, all code, the protocol with its hash, and every raw record from which the tables and figures are generated.

7. Results

All results in this section are from the 16 held-out test events (16 family-level transitions, since no test event affects more than one family), which were run once, after the protocol freeze. Unless stated otherwise, numbers are for the primary model (three runs), rates are pooled over component-event or task-instance outcomes, brackets give 95% cluster-bootstrap intervals, and differences are paired over the same resampled (family, event) clusters. The test split contains 337 valid-component outcomes per run and method, but only seven newly prohibited components, spread over five policy events, and five previously prohibited ones scored again at later events; conclusions about retirement therefore rest on few clusters, and we report counts next to rates. Overall retirement (Table 3) has 12 prohibited-component outcomes per run; retirement after policy changes (Tables 4 and 5) has 9.

7.1 Overview

Table 3. Main results on the test split (gpt-4.1-mini, three runs). Library metrics are shares of component-event outcomes (%); task metrics are shares of downstream task instances (%); cost is the mean adaptation cost per change event in thousandths of a US dollar at list prices; calls are model calls per event excluding task solving. Retire. is overall retirement, which includes components prohibited at an earlier event and scored again at later library or policy events (Table 6); retirement after policy changes is in Tables 4 and 5. Intervals are in Table 12; the repository commands that reproduce this table are listed in Appendix A.5.

Method	Retention	Repair	Retire.	Correct	Task succ.	Task viol.	m\$/event	Calls
No skills	–	–	–	–	67.8	48.1	0.0	0.0
Static library	100.0	0.0	0.0	49.0	22.2	94.4	0.0	0.0
Doc retrieval	100.0	0.0	0.0	49.0	52.8	59.3	0.0	0.0
Version-aware retrieval	80.7	0.0	0.0	27.5	70.6	61.1	0.0	0.0
Full regeneration	91.2	30.8	77.8	74.8	65.6	44.4	31.7	8.8
Regression-gated (GRASP-style)	100.0	38.7	0.0	78.7	52.8	94.4	11.0	7.5
Regression-gated + refresh	99.5	49.8	47.2	86.4	60.0	61.1	23.9	12.6
SkillLedger	82.9	42.8	61.1	68.9	64.4	55.6	21.8	9.1

Table 3 and Figs. 3 and 4 show that no method dominates. The plain regression gate never broke a still-valid component (retention 100.0%) and the refreshed gate almost never (99.5%), but the plain gate, whose probe is the historical test suite, retired none of the prohibited behavior (0.0% overall and after policy changes) and left the agent violating the current policy in 94.4% of policy-tested task instances. Refreshing the probe tests from the change documents before gating produced the most correct libraries overall (86.4% of components correct after adaptation). SkillLedger reached 68.9%: its overall retirement was 61.1% (48.1% after policy changes), and it repaired 42.8% of broken components, but lost 17.1% of components that were still valid. Full regeneration retired the most (77.8% overall, 70.4% after policy changes) at the highest cost and with 8.8% accidental

loss. Version-aware retrieval, which withholds any skill whose own tests fail, kept only 27.5% of components correct, fewer than doing nothing (49.0%). The refreshed gate's overall retirement (47.2%) is below SkillLedger's although the two are tied after policy changes (48.1% each): the difference comes from components prohibited before the test split that the refreshed gate had left in place and that are scored again at later library events (Section 7.4).

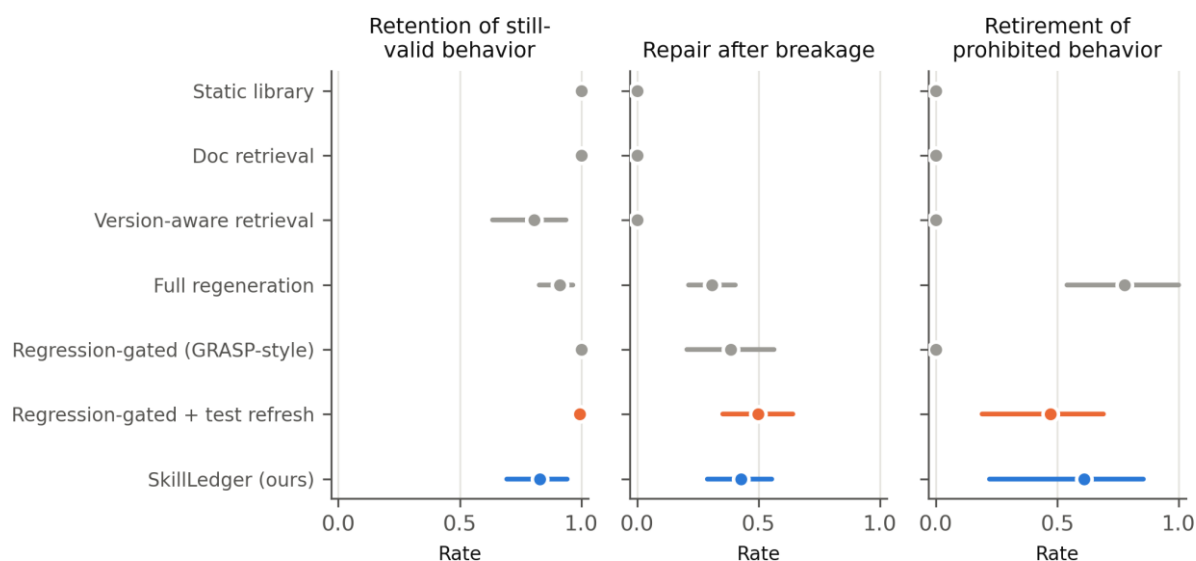


Fig. 3. Retention, repair and retirement on the test split (gpt-4.1-mini, three runs). Dots are pooled rates, lines 95% cluster-bootstrap intervals. Retirement is overall retirement: it pools newly prohibited components with components prohibited at an earlier event (Table 6). Methods without a library are omitted.

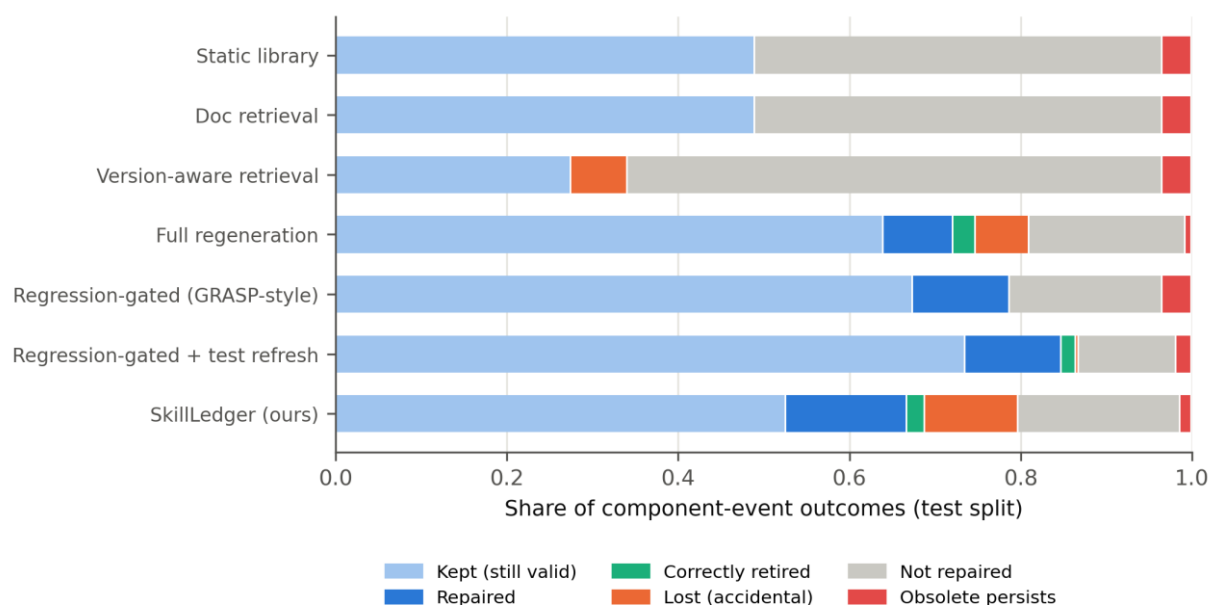


Fig. 4. Composition of all component-event outcomes after adaptation on the test split. Orange segments are losses of behavior that was still valid and passing before the adaptation; red segments are prohibited behavior still present afterwards.

Two observations frame the rest of this section. Selective evidence quarantine lets SkillLedger make retirement decisions that the unchanged gate rejects; but rewriting the probe tests from the change documentation removes the same structural obstacle, and against that refreshed gate SkillLedger shows no advantage in retirement after policy changes or in policy compliance, with substantially lower retention.

7.2 Preregistered hypotheses

Table 4. Preregistered contrasts on the test split: SkillLedger minus the comparison method. Differences are in percentage points (cost in thousandths of a dollar per event); intervals are paired cluster-bootstrap 95% intervals for gpt-4.1-mini; the last two columns give the point differences for the single runs of the other models. NI: non-inferiority, supported if the upper bound

is below the margin. H3 predicted a repair rate at least equal to the gate's.

Hyp.	Outcome	Versus	Difference [95% CI]	Verdict	nano	mini
H1	Retirement after policy changes	Regression-gated (GRASP-style)	+48.1 [+11.1, +70.0]	supported	+55.6	+55.6
H1	Task policy violations after policy changes	Regression-gated (GRASP-style)	-37.8 [-60.8, -17.9]	supported	-46.7	-80.0
H1 (sec.)	Retirement after policy changes	Regression-gated + refresh	+0.0 [-11.1, +7.7]	not supported	+0.0	-22.2
H1 (sec.)	Task policy violations after policy changes	Regression-gated + refresh	-2.2 [-6.7, +0.0]	not supported	+13.3	-13.3
H2	Accidental loss	Full regeneration	+8.3 [-7.5, +24.7]	not supported	-1.6	+2.2
H2	Accidental loss	Version-aware retrieval	-2.2 [-16.6, +9.5]	not supported	-13.4	-
H2 (NI)	Accidental loss, margin 5 pts	Regression-gated + refresh	+16.6 [+5.2, +30.4]	rejected	+5.5	+7.1
H3	Repair after library changes	Regression-gated (GRASP-style)	-9.3 [-33.3, +7.8]	not supported	+10.1	+9.1
H4	Adaptation cost per event	Full regeneration	-9.9 [-15.0, -5.1]	supported	-9.0	-23.0
H4	Adaptation cost per event	Regression-gated + refresh	-2.1 [-8.1, +3.3]	not supported	-3.3	-18.4

H1 (retirement) is supported against the preregistered primary comparator. After policy changes, SkillLedger retired more prohibited behavior than the GRASP-style gate (+48.1 points [+11.1, +70.0]) and reduced downstream policy violations (-37.8 points [-60.8, -17.9]); both differences have the same sign, and are at least as large, for the two other models. The gate's failure is structural: under a policy change, the historical tests encode the superseded rule, so every edit that implements the new rule breaks a previously passing test and is rejected.

Against the refreshed gate, the protocol's own criterion for evidence against the contribution is met. The refreshed gate matched SkillLedger on retirement after policy changes (+0.0 points [-11.1, +7.7]) and on downstream violations (-2.2 points [-6.7, +0.0]), and no statistically clear difference in adaptation cost was detected (-2.1 m\$ [-8.1, +3.3]). Rewriting the probe tests from the change documents is enough to lift the structural block that stops a regression gate from retiring behavior.

H2 (retention) is not supported. SkillLedger's accidental-loss rate was not lower than full regeneration's (+8.3 points [-7.5, +24.7]) or version-aware retrieval's (-2.2 points [-16.6, +9.5]), and non-inferiority against the refreshed gate was rejected (+16.6 points [+5.2, +30.4]; margin +5 points), as the development pilot had warned. **H3 (repair after library changes) is not supported** for the primary model (-9.3 points [-33.3, +7.8]), with small positive point differences for the other two models. **H4 (cost) is supported against regeneration** (-9.9 m\$ [-15.0, -5.1]) **but not against the refreshed gate** for the primary model; for gpt-5.4-mini, SkillLedger was cheaper than both.

7.3 Policy changes versus library changes

Table 5. Adaptive methods by event kind (gpt-4.1-mini, test split, %). Library events: 11 (family, event) clusters; policy events: 5. Task violations are over policy-tested task instances in policy events. Pol. retire is retirement after policy changes. Cost in thousandths of a dollar per event.

Method	Lib. ret.	Lib. rep.	Pol. ret.	Pol. rep.	Pol. retire	Pol. viol.	m\$ lib.	m\$ pol.
Full regeneration	91.3	39.1	90.0	17.8	70.4	48.9	29.5	36.6
Regression-gated (GRASP-style)	100.0	50.6	100.0	0.0	0.0	93.3	7.9	18.0
Regression-gated + refresh	100.0	53.9	97.5	31.8	48.1	57.8	19.2	34.2
SkillLedger	77.9	41.4	97.6	53.8	48.1	55.6	22.2	20.9

The distinction between policy changes and library changes is particularly important (Table 5). Following policy revisions, SkillLedger repairs 53.8% of affected components (for example, PBKDF2 iteration counts and Argon2 memory parameters),

compared with 31.8% for the refreshed gate (+22.0 points [+11.9, +33.3]), 17.8% for regeneration and 0.0% for the plain gate, at lower cost (20.9 versus 34.2 m\$ per event), while retaining 97.6% of unaffected behavior. Following library changes, SkillLedger retains only 77.9% of still-valid components, compared with 100.0% for both regression-gated methods, and repairs fewer (41.4% versus 53.9%).

The results therefore do not support a single ranking of adaptation methods across all environmental changes. Selective evidence maintenance is beneficial when historical expectations conflict with current policy, whereas broad test refresh is more robust when executable tests themselves depend on changing APIs. This difference motivates the failure analysis in Section 7.5.

7.4 Retirement and recurrence

Table 6. Prohibited components that were absent after adaptation (count retired / component-event outcomes, pooled over runs). *New*: prohibited by the current event (seven per run). *Earlier*: prohibited by an earlier event and still prohibited (five per run, including the three legacy-cryptography components retired on the development split).

Method	4.1-mini new	4.1-mini earlier	nano new	nano earlier	mini new	mini earlier
Static library	0/21	0/15	0/7	0/5	–	–
Full regeneration	13/21	15/15	4/7	5/5	5/7	5/5
Regression-gated (GRASP-style)	0/21	0/15	0/7	0/5	0/7	0/5
Regression-gated + refresh	13/21	4/15	3/7	5/5	5/7	5/5
SkillLedger	13/21	9/15	5/7	3/5	5/7	0/5

Table 6 separates the two ways prohibited behavior can be present. For newly prohibited components, SkillLedger, the refreshed gate and regeneration each retired 13 of 21 outcomes with the primary model; the plain gate and the non-adaptive methods retired none. SkillLedger’s most systematic miss was a conditional prohibition. The December 2024 OWASP revision restricts bcrypt to “legacy systems where Argon2 and scrypt are not available”, and the family’s deployment context states that both are available. In all three runs SkillLedger’s impact step quoted the restriction yet judged that new bcrypt hashes were still permitted under it, so the component persisted through all three OWASP test events; regeneration, which rewrites the skill from the current text, retired it in every run. For components prohibited earlier, the question is whether the library still contains them. Once a method had retired a component, it essentially never reappeared: across all models and methods we observed one reintroduction, by regeneration with gpt-5.4-mini. Earlier-prohibited behavior that was present on the test split had simply never been retired, typically on the development split (for example, the refreshed gate had left some legacy-cryptography components in place on the development split, and SkillLedger had never retired new bcrypt hashes).

7.5 Why SkillLedger loses valid behavior

SkillLedger’s retention failures are concentrated rather than uniformly distributed across the benchmark. Of the 114 lost component outcomes recorded across the primary model’s three runs, 110 result from withholding skills rather than introducing incorrect modifications. The NumPy 2.4 and SQLAlchemy 2.1 events account for 69 of these losses (42 and 27; Table 15).

The NumPy case exposes the underlying mechanism. A skill had already been migrated to a functioning implementation, but one historical regression test continued to invoke `np.in1d`, which had been removed from the newer library version. The test therefore failed independently of the correctness of the repaired skill. Under SkillLedger’s evaluated acceptance procedure, the failed test remained authoritative unless an accepted repair replaced it with a corrected test of the same name. None of the three repair attempts provided such a replacement. Consequently, the system withheld the working skill, removing access to all components it provided.

Related failures occurred when historical tests invoked other removed or modified NumPy interfaces, including `ndarray.tostring`, `np.trapz`, and the interpolation argument of `quantile`. SQLAlchemy exhibited a similar problem, with removed interfaces such as `Engine.execute` appearing in both skill implementations and validation code.

These failures distinguish two types of evidence staleness. **Normative staleness** occurs when a governing requirement is

superseded. **Technical staleness** occurs when the executable machinery used to test a still-valid requirement becomes incompatible with the current environment. SkillLedger primarily addresses normative staleness. The refreshed gate performs better under library evolution, consistent with its test-rewriting step also replacing technically outdated evidence. The observed retention gap is therefore not adequately explained as a failure of skill repair alone. It reflects a mismatch between the continuing validity of a behavioral contract and the validity of the mechanism used to evaluate that contract.

7.6 Downstream tasks

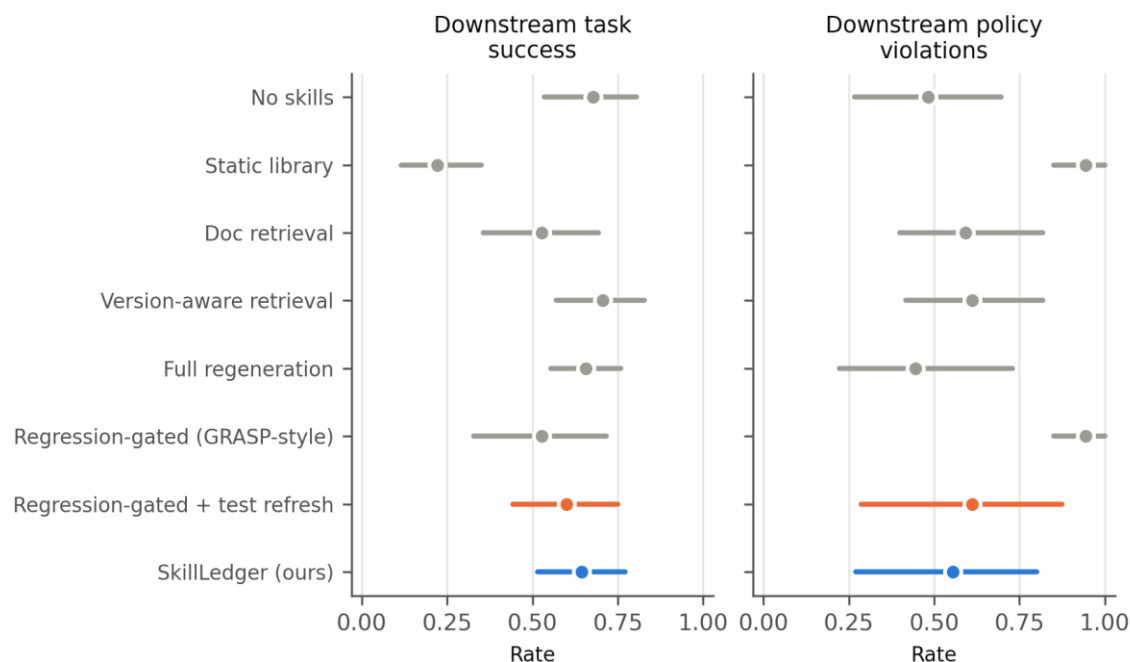


Fig. 5. Downstream task success and policy violations on the test split (gpt-4.1-mini). Violations are computed over task instances that carry a policy test (18 per run).

Library quality did not translate one-to-one into task outcomes (Fig. 5). Recall that the no-skills and the two retrieval baselines see the current documentation at task time, whereas the four adaptive methods use the documentation only while adapting and present just the adapted library at task time. An unmaintained library was harmful. Without documentation it reached 22.2% task success, against 67.8% for the same model solving each task from the current documentation alone; and even with the documentation in the prompt, adding the stale library lowered success to 52.8% (-15.0 points [-30.1, -1.1]), although not for gpt-5.4-nano (+3.3 points [-3.2, +10.2]). The adaptive methods, without documentation at task time, reached 52.8–65.6%, close to but not above the documentation-only baseline; version-aware retrieval, which pairs the documentation with only those skills that still pass their tests, was highest (70.6%). For policy compliance, SkillLedger reduced violations relative to the plain gate (55.6% versus 94.4%) but not relative to working without a library (48.1%) or to regeneration (44.4%). These task-level comparisons rest on 18 policy-tested task instances per run and should be read as indicative.

7.7 Ablations

Table 7. Ablations of SkillLedger (gpt-4.1-mini, test split, three runs; %, cost in m\$ per event). Retire. is overall retirement.

Variant	Retention	Repair	Retire.	Correct	Task succ.	Task viol.	m\$/event
SkillLedger	82.9	42.8	61.1	68.9	64.4	55.6	21.8
SkillLedger w/o evidence quarantine	81.5	30.4	0.0	59.1	53.9	83.3	25.7
SkillLedger w/o tombstones	84.0	39.3	69.4	68.6	63.3	48.1	21.1
SkillLedger non-selective review	83.6	37.9	61.1	68.4	63.9	48.1	19.5

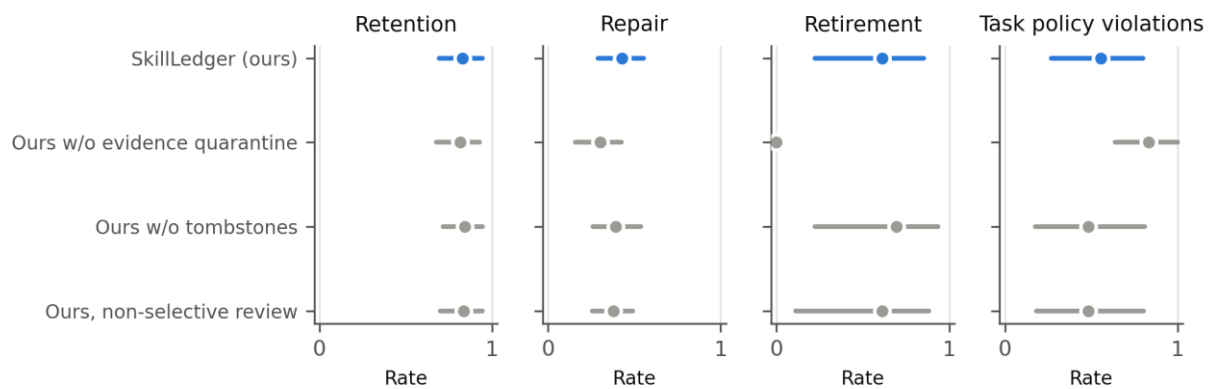


Fig. 6. Ablations on the test split: estimates and 95% cluster-bootstrap intervals. Retirement is overall retirement.

Evidence quarantine is the component that makes retirement possible. Without it, SkillLedger behaves like the plain gate under policy changes: overall retirement fell to 0.0%, as did retirement after policy changes (+48.1 points [+11.1, +70.0] for the full method on policy events), repair after policy changes fell by 53.8 points, and task policy violations rose to 83.3%. Tombstones did not help: removing them left task success unchanged within noise and, if anything, lowered task violations (48.1% versus 55.6%; +7.4 points [-4.4, +26.7] for the full method) and retirement after policy changes (-11.1 points [-19.4, +0.0] for the full method). The one-line prohibitions shown at task time did not measurably change what the model wrote. Selective review did not reduce cost; the non-selective variant was slightly cheaper (19.5 versus 21.8 m\$ per event), because most library events flag at least one skill and the impact step is a single call per family either way. Outcomes were similar except for repair after policy changes (+10.6 points [+1.8, +19.1] for selective review); since both variants review every skill at policy events, this difference can only arise indirectly, through libraries that diverged at earlier library events, and we do not interpret it.

7.8 Other models and a silent failure mode

Table 8. Test-split results for the two further models (one run each; %, cost in m\$ per event). Retire. is overall retirement.

Model	Method	Retention	Repair	Retire.	Correct	Task succ.	Task viol.	m\$/event
gpt-5.4-nano	Static library	100.0	0.0	0.0	49.0	28.3	100.0	0.0
	Doc retrieval	100.0	0.0	0.0	49.0	83.3	33.3	0.0
	Version-aware retrieval	80.7	0.0	0.0	27.5	80.0	33.3	0.0
	Full regeneration	92.5	32.9	75.0	79.9	80.0	27.8	25.9
	Regression-gated (GRASP-style)	100.0	43.3	0.0	80.8	61.7	94.4	8.9
	Regression-gated + refresh	99.6	51.8	66.7	87.1	75.0	27.8	20.2
	SkillLedger	94.1	63.1	66.7	85.7	83.3	38.9	16.9
gpt-5.4-mini	Full regeneration	95.1	39.6	83.3	86.2	91.7	11.1	66.5
	Regression-gated (GRASP-style)	100.0	46.8	0.0	82.2	65.0	72.2	28.4
	Regression-gated + refresh	100.0	76.5	83.3	94.8	86.7	16.7	61.9
	SkillLedger	92.9	74.7	41.7	86.8	88.3	5.6	43.5

With gpt-5.4-nano, the pattern of the primary model repeated with smaller losses: SkillLedger retained 94.1% of still-valid components, repaired 63.1% (the highest of all methods), retired 66.7% overall (55.6% after policy changes; both tied with the refreshed gate) and reached the highest task success (83.3%, tied with documentation retrieval). With gpt-5.4-mini, SkillLedger's overall retirement was only 41.7% (55.6% after policy changes), below regeneration and the refreshed gate (83.3% overall, 77.8% after policy changes). Inspection traced much of this to a failure of our implementation rather than of the model's judgment (Table 9).

Table 9. Impact-step responses of the frozen SkillLedger that its JSON parser could not read, by model; a failed parse was

treated as an empty decision set, i.e. every component was retained. Recovered: responses that the post-hoc tolerant parser (Amendment A1) reads.

Model	Runs	Impact calls	Unparsed	Unparsed (%)	Recovered
gpt-4.1-mini	3	84	5	6.0	5
gpt-5.4-nano	1	28	1	3.6	1
gpt-5.4-mini	1	28	9	32.1	9

The impact step asks for a JSON object of per-component decisions. The frozen implementation treats a response that is not valid JSON as containing no decisions, so every component is retained: a fail-open default. With gpt-5.4-mini, 9 of 28 impact responses were malformed, most often by a single missing closing brace; with gpt-4.1-mini, typical defects were Python-style triple-quoted strings inside JSON. In the legacy-cryptography family, gpt-5.4-mini's responses correctly said to retire Triple-DES encryption, PKCS#1 v1.5 key wrapping and DSA signing at the respective events, but none of these decisions was applied. The same fail-open behavior affected 5 of 84 impact decisions of the primary model and between 5 and 10 of 84 for the ablations. No baseline depends on structured output in this way; they exchange code blocks, which the parser extracted reliably.

7.9 Post-hoc check: tolerant parsing (Amendment A1)

Table 10. Post-hoc Amendment A1 (not preregistered): SkillLedger with a tolerant parser for the impact response, run on the test split after the main results had been analyzed. It reuses every recorded response whose prompt is unchanged, so it differs from the frozen method only downstream of a recovered parse (Retire.: overall retirement; %; cost in m\$ per event, including reused calls at their original price).

Model	Method	Retention	Repair	Retire.	Correct	Task succ.	Task viol.	m\$/event
gpt-4.1-mini	SkillLedger	82.9	42.8	61.1	68.9	64.4	55.6	21.8
	SkillLedger + tolerant parsing (A1)	82.3	41.6	66.7	68.3	63.9	55.6	22.4
	Regression-gated + refresh	99.5	49.8	47.2	86.4	60.0	61.1	23.9
	Full regeneration	91.2	30.8	77.8	74.8	65.6	44.4	31.7
gpt-5.4-nano	SkillLedger	94.1	63.1	66.7	85.7	83.3	38.9	16.9
	SkillLedger + tolerant parsing (A1)	94.4	64.7	66.7	86.2	83.3	33.3	15.9
	Regression-gated + refresh	99.6	51.8	66.7	87.1	75.0	27.8	20.2
	Full regeneration	92.5	32.9	75.0	79.9	80.0	27.8	25.9
gpt-5.4-mini	SkillLedger	92.9	74.7	41.7	86.8	88.3	5.6	43.5
	SkillLedger + tolerant parsing (A1)	93.1	74.4	58.3	87.7	86.7	22.2	45.2
	Regression-gated + refresh	100.0	76.5	83.3	94.8	86.7	16.7	61.9
	Full regeneration	95.1	39.6	83.3	86.2	91.7	11.1	66.5

Because the parse failure is an implementation defect rather than a design choice, we ran one post-hoc check, declared as Amendment A1 in the protocol after the frozen results had been analyzed (Table 10). The variant repairs the structure of a malformed impact response (unbalanced brackets, trailing commas, Python-style string quoting) and asks once more if it still cannot be parsed; it recovers all 40 malformed responses recorded in the main study, and no re-ask was needed. Since it shares the frozen method's response cache, its trajectory is identical until the first recovered parse; after that, prompts change and new samples are drawn, so later differences mix the fix with sampling variation.

For the primary model the fix changed little: overall retirement rose from 61.1% to 66.7% (+5.6 points [+0.0, +13.3]; both additional retirements were newly prohibited components, 13 to 15 of 21) and retirement after policy changes from 48.1% to 55.6%, and retention (82.3%), repair and task outcomes were unchanged within noise, so none of the conclusions in Sections

7.2–7.7 depends on the defect. With gpt-5.4-mini, the fix applied the recorded legacy-cryptography decisions: the three legacy-cryptography components, which had persisted, were now retired. Overall retirement rose from 41.7% to 58.3%, still below the refreshed gate (83.3%), while retirement after policy changes fell from 55.6% to 44.4% (refreshed gate 77.8%), because one component that the frozen run had retired (the audit of Mozilla’s withdrawn Old profile) persisted in the diverged trajectory, and task policy violations rose from 5.6% to 22.2% of 18 policy-tested task instances. With a single run, these last differences cannot be separated from sampling variation. The parse defect therefore explains the anomalous legacy-cryptography result for gpt-5.4-mini, but not the whole gap to the refreshed gate.

7.10 Cost



Fig. 7. Share of components correct after adaptation against mean adaptation cost per change event (gpt-4.1-mini, test split, list prices). Vertical lines are 95% intervals.

SkillLedger used 9.1 model calls and 49 thousand tokens per change event, comparable with the refreshed gate (12.6 calls, 66 thousand tokens) and below regeneration (72 thousand tokens); the plain gate was cheapest among the adaptive methods (11.0 m\$ per event). Wall-clock adaptation time per event, dominated by test execution, was 74 s for SkillLedger, 74 s for the refreshed gate and 111 s for regeneration. At these list prices, all adaptive methods cost a few cents per change event for a family of four to seven skills (Fig. 7).

8. Discussion

8.1 Evidence validity as a distinct maintenance problem

The principal implication of this study is that continual skill adaptation depends on the continuing validity of regression evidence, not only on the correctness of skill implementations. A passing test is best understood as a claim indexed to a state: it asserts that a requirement held under particular library versions and particular governing rules. Conventional regression gating treats such claims as timeless, and SkillShift shows two independent ways in which they expire. **Normative staleness** arises when the requirement a test encodes is superseded; keeping the test then blocks correct retirement. **Technical staleness** arises when the requirement is still valid but the test’s own implementation no longer runs; keeping the test then rejects or withholds valid skills. A maintenance method therefore needs to represent the validity of its evidence as explicit state, with its own provenance and dependencies, rather than inferring it from whether the evidence passes. Selective quarantine and wholesale refresh are two approximations of this, covering normative and technical staleness respectively; neither is a general solution, and the experiments establish the distinction rather than a winner.

8.2 Relation to prior work

The results refine several established views. In continual learning, forgetting is measured as a loss to be minimized [25]–[27]; under policy change, some forgetting is required, so stability and plasticity must be complemented by a third objective,

retirement, scored separately from accidental loss. Regression-gated self-improvement [6] and the regression tax it addresses [9] presuppose a stationary probe set; our results indicate that, once rules change, regression protection is only as good as the currency of the probes, which places probe maintenance alongside candidate generation as a design concern. Environment contracts for skills [8] track technical assumptions and are complementary to the normative grounds recorded here; the technical staleness we observe suggests extending such contracts to test code as well. Finally, unlike unlearning and knowledge editing in model weights [28]–[31], retirement in a non-parametric library can be exact, yet the downstream results suggest that exact removal from the library does not prevent the model from reintroducing the behavior, an analog of the incomplete propagation reported for parametric edits [31] and of the difficulty of recognizing invalidated memories [23].

8.3 Retention and retirement are independent objectives

A system that preserves every historical behavior can retain prohibited procedures, and one that aggressively regenerates skills can lose capabilities that remain valid. Component-level metrics make this trade-off visible where aggregate pass rates hide it. Conditional prohibitions add a further difficulty: in the OWASP case, whether bcrypt may still be used depends on the deployment, and SkillLedger's impact step quoted the correct clause yet applied it incorrectly. Retrieving the right evidence is therefore not the same as reaching the right maintenance decision, and conditional rules deserve explicit representation rather than free-text judgment.

8.4 Implications for deployment

Treating a skill library like a software supply chain suggests five concrete practices. First, version the evidence as well as the code: record the API dependencies of each test, not only of the skill it checks, so that technically stale tests can be repaired or retired rather than allowed to withhold working skills. Second, re-derive regression probes from the governing documents whenever a policy changes, and keep unchanged probes as constraints. Third, make structured decision steps fail closed: an unreadable model response, an uncertain judgment and a deliberate retain decision are different states, and the fail-open default we observed silently suppressed correct retirements. Fourth, enforce retirement in code (removal or refusal) and in guards around the agent, rather than relying on prompt-level notices, which did not reduce violations in our experiments. Fifth, report retention, repair, retirement and downstream compliance separately, because maintained libraries recovered task success without reliably improving on documentation alone, and because library components are often called by software without a model in the loop. At a few US cents per change event, such maintenance is affordable as a routine, event-driven step.

9. Threats To Validity And Limitations

Construct validity. SkillShift evaluates behavior through author-defined hidden tests and reference implementations. Software behavior is checked against real library versions, while policy compliance is operationalized through published guidance and benchmark-specific deployment contexts. Although the source revisions are authentic, some policy interpretations and deployment assumptions necessarily involve judgment. Alternative interpretations of conditional requirements could change individual component labels.

Internal validity. The benchmark, SkillLedger, and comparison implementations were developed by the same author. This creates the possibility that methodological decisions or benchmark structure favor particular approaches. The inclusion of a stronger refreshed regression-gate baseline mitigates this concern and, importantly, produces results unfavorable to SkillLedger on several metrics. It does not replace independent replication. The environments pin a late patch release of each release series (for example SQLAlchemy 1.4.54 and 2.0.54) so that every state installs on a supported CPython; event dates are those of the series' first release, which affects no label. The post-hoc parsing amendment is reported separately and is not used to revise the preregistered findings.

Statistical validity. The held-out evaluation contains 16 family-event clusters (16 distinct events), including five policy events and only seven newly prohibited components. Retirement and policy-compliance conclusions therefore depend on a limited number of independent changes. Repeated model runs increase the number of observations but do not create additional independent policy events. Confidence intervals should be interpreted with this clustering and limited event diversity in mind; no correction for multiple comparisons was applied beyond the small set of preregistered contrasts.

External validity. The evaluated skills are executable Python modules, and the environmental changes concern Python libraries, runtimes, and published security guidance. The study does not establish that the same outcomes generalize to natural-

language procedural memory, larger enterprise skill libraries, other regulatory domains, or agents that must discover environmental changes autonomously. All three evaluated models also belong to one provider family, and secondary-model results are based on single runs. Every test event post-dates the primary model's knowledge cutoff, but seven of the 16 precede that of the secondary models.

Evaluation conditions. The baseline methods differ in how current documentation is supplied and at which stage it becomes available. This reflects their intended operating models but limits direct interpretation of some downstream task comparisons. Model inference costs are estimated using reported token usage and list prices (US\$28.50 for the whole study); they should not be interpreted as comprehensive production operating costs.

Scope of the findings. The experiments establish that both obsolete requirements and obsolete test implementations can compromise continual skill adaptation. They do not establish a method that reliably resolves both problems across arbitrary software and policy changes.

10. Conclusion

Continual adaptation of AI-agent skills requires preserving valid capabilities, repairing incompatible implementations, and retiring behavior prohibited by current requirements. These objectives cannot be reduced to ordinary regression preservation because both software interfaces and governing policies evolve. We introduced SkillShift to evaluate this problem through real, dated software and policy changes, executable validation, and a chronological testing protocol. We also evaluated SkillLedger, a version-indexed evidence mechanism that selectively quarantines superseded policy expectations.

The results establish that unchanged regression gates can obstruct necessary retirement by preserving outdated expectations. SkillLedger addresses this problem and improves repair following policy revisions. However, refreshed regression gating achieves comparable retirement after policy changes while retaining substantially more valid behavior, exposing a limitation in SkillLedger's treatment of technical test dependencies. The failure analysis identifies a general problem: a regression test can cease to be a valid instrument of evaluation even when the behavior it was designed to test remains correct.

The central conclusion is that continual skill maintenance must account for the evolution of validation evidence as well as the evolution of skills themselves. SkillShift provides a reproducible setting for studying this distinction, and its findings clarify both the capabilities and limitations of existing adaptation strategies.

Declarations

Funding. This research was self-funded by the author and received no external funding.

Author contributions. Lakshmi Charan Lingisetty: conceptualization, methodology, software, investigation, validation, formal analysis, visualization, and writing (original draft, review and editing). The author reviewed and approved the final manuscript and accepts responsibility for its content.

Data and code availability. The SkillShift benchmark, SkillLedger implementation, baseline methods, study protocol, recorded evaluation outcomes, and analysis scripts are publicly available at <https://github.com/lakshmi-charan/skillshift>. The artifacts corresponding to this manuscript are archived in release v1.0 at <https://github.com/lakshmi-charan/skillshift/releases/tag/v1.0>, corresponding to commit 0109f1c0d57156dcdeabb9d2d1dd5d2833f96657. The release includes the raw-records archive, containing the recorded prompts, model responses, token usage, estimated costs, and timings for 12,612 model calls, together with skill-library snapshots after each recorded state. The evaluation records in the repository support regeneration of the reported numerical results, tables, and figures. The raw-records archive provides the underlying model interactions and generated code for inspection and reconstruction of the response cache. Third-party materials remain subject to their respective licenses and attribution requirements.

Corresponding author. Lakshmi Charan Lingisetty. Email: lingisettycharan123@gmail.com

References

1. G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, et al., "Voyager: An open-ended embodied agent with large language models," Transactions on Machine Learning Research, 2024. arXiv:2305.16291.

2. A. Zhao, D. Huang, Q. Xu, M. Lin, Y.-J. Liu, and G. Huang, "ExpeL: LLM agents are experiential learners," in Proc. AAAI Conf. Artificial Intelligence, 2024. arXiv:2308.10144.
3. N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in Advances in Neural Information Processing Systems 36, 2023.
4. Z. Z. Wang, J. Mao, D. Fried, and G. Neubig, "Agent workflow memory," in Proc. Int. Conf. Machine Learning (ICML), 2025. arXiv:2409.07429.
5. R. Fang, Y. Liang, X. Wang, J. Wu, S. Qiao, P. Xie, et al., "Memp: Exploring agent procedural memory," in Findings of the Association for Computational Linguistics: ACL 2026, pp. 17490–17502, 2026. arXiv:2508.06433.
6. J. Moll, J.-P. Corbeil, J. Pan, M. Hadamitzky, D. Rueckert, L. Adams, et al., "GRASP: Gated regression-aware skill proposer for self-improving LLM agents," arXiv:2605.29668, 2026.
7. H. Pu, X. Song, and L. Zhao, "SkillOps: Managing LLM agent skill libraries as self-maintaining software ecosystems," arXiv:2605.13716, 2026.
8. L. Fan, Y. Tian, Z. Li, and Z. Lu, "Skill drift is contract violation: Proactive maintenance for LLM agent skill libraries," arXiv:2605.10990, 2026.
9. D. Tank and B. Nama, "The regression tax: Decomposing why skills help—and hurt—LLM agents," arXiv:2607.22520, 2026.
10. B. Wu, E. Meij, and E. Yilmaz, "Beyond static toolsets: Self-evolving LLM tool agents via continual documentation adaptation," in Findings of the Association for Computational Linguistics: ACL 2026, pp. 21519–21539, 2026.
11. G. Chen, Z. Zhang, X. Cong, F. Guo, Y. Wu, Y. Lin, et al., "Learning evolving tools for large language models," in Proc. Int. Conf. Learning Representations (ICLR), 2025. arXiv:2410.06617.
12. H. Liu, K. Hu, J. Liao, Q. Wang, P. Qian, Y. Zhai, et al., "MCPEvol-Bench: Benchmarking LLM agent performance across dynamic evolutions of MCP servers," arXiv:2607.14642, 2026.
13. G. Li, Y. Xie, Y. Liu, Z. Dong, X. Pan, T. Zheng, et al., "The world won't stay still: Programmable evolution for agent benchmarks," arXiv:2603.05910, 2026.
14. D. Misra, N. Islah, V. May, B. Rauby, Z. Wang, J. Gehring, et al., "GitChameleon 2.0: Evaluating AI code generation against Python library version incompatibilities," in Proc. 64th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 46792–46831, 2026. arXiv:2507.12367.
15. Z. L. Liu, S. Pandit, X. Ye, E. Choi, and G. Durrett, "CodeUpdateArena: Benchmarking knowledge editing on API updates," arXiv:2407.06249, 2024.
16. T. Wu, W. Wu, X. Wang, K. Xu, S. Ma, B. Jiang, et al., "VersiCode: Towards version-controllable code generation," arXiv:2406.07411, 2024.
17. S. Kuhar, W. U. Ahmad, Z. Wang, N. Jain, H. Qian, B. Ray, et al., "LibEvolutionEval: A benchmark and study for version-specific code generation," in Proc. NAACL, pp. 6826–6840, 2025.
18. C. Wang, Z. Chu, Z. Cheng, X. Yang, K. Qiu, Y. Wan, et al., "CodeSync: Synchronizing large language models with dynamic code evolution at scale," in Proc. ICML, PMLR 267, pp. 62672–62700, 2025.
19. J. Choi, J. Yoon, L. T. Le, S. Jha, and T. Pfister, "PolicyBank: Evolving policy understanding for LLM agents," arXiv:2604.15505, 2026.
20. S. Yao, N. Shinn, P. Razavi, and K. Narasimhan, "τ-bench: A benchmark for tool-agent-user interaction in real-world domains," in Proc. ICLR, 2025.

21. Z. Xiang, L. Zheng, Y. Li, J. Hong, Q. Li, H. Xie, et al., "GuardAgent: Safeguard LLM agents via knowledge-enabled reasoning," in Proc. ICML, 2025.
22. Z. Chen, M. Kang, and B. Li, "ShieldAgent: Shielding agents via verifiable safety policy reasoning," in Proc. ICML, PMLR 267, pp. 8313–8344, 2025.
23. H. Chao, Y. Bai, R. Sheng, T. Li, and Y. Sun, "STALE: Can LLM agents know when their memories are no longer valid?," arXiv:2605.06527, 2026.
24. Y. Shu, B. Jiménez Gutiérrez, S. P. Jonnalagedda, Y. Yao, H. Sun, and Y. Su, "AgentCL: Toward rigorous evaluation of continual learning in language agents," arXiv:2606.02461, 2026.
25. J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, et al., "Overcoming catastrophic forgetting in neural networks," Proc. National Academy of Sciences, vol. 114, no. 13, pp. 3521–3526, 2017.
26. L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," IEEE Trans. Pattern Analysis and Machine Intelligence, 2024, doi: 10.1109/TPAMI.2024.3367329.
27. D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in Advances in Neural Information Processing Systems 30, pp. 6467–6476, 2017.
28. L. Bourtole, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, et al., "Machine unlearning," in Proc. IEEE Symp. Security and Privacy, pp. 141–159, 2021, doi: 10.1109/SP40001.2021.00019.
29. K. Meng, D. Bau, A. Andonian, and Y. Belinkov, "Locating and editing factual associations in GPT," in Advances in Neural Information Processing Systems 35, 2022.
30. K. Meng, A. Sen Sharma, A. Andonian, Y. Belinkov, and D. Bau, "Mass-editing memory in a transformer," in Proc. ICLR, 2023.
31. R. Cohen, E. Biran, O. Yoran, A. Globerson, and M. Geva, "Evaluating the ripple effects of knowledge editing in language models," Trans. Association for Computational Linguistics, vol. 12, pp. 283–298, 2024.
32. National Institute of Standards and Technology, "Digital identity guidelines: Authentication and lifecycle management," NIST SP 800-63B, June 2017 (updated Mar. 2, 2020), doi: 10.6028/NIST.SP.800-63b.
33. National Institute of Standards and Technology, "Digital identity guidelines: Authentication and authenticator management," NIST SP 800-63B-4, July 2025, doi: 10.6028/NIST.SP.800-63b-4.
34. E. Barker and A. Roginsky, "Transitioning the use of cryptographic algorithms and key lengths," NIST SP 800-131A Rev. 2, Mar. 2019, doi: 10.6028/NIST.SP.800-131Ar2.
35. National Institute of Standards and Technology, "Digital signature standard (DSS)," FIPS PUB 186-5, Feb. 2023, doi: 10.6028/NIST.FIPS.186-5.
36. OWASP Foundation, "Password storage cheat sheet," OWASP Cheat Sheet Series, revisions 728bbcd (2021-03-17) through c3c1952 (2026-05-12), <https://github.com/OWASP/CheatSheetSeries> (CC BY-SA 4.0).
37. Mozilla, "Serverside TLS guidelines," versions 5.6–6.0, mozilla/ssl-config-generator, <https://github.com/mozilla/ssl-config-generator> (MPL-2.0).
38. Python Software Foundation, "What's new in Python 3.12 / 3.13 / 3.14," <https://docs.python.org/3/whatsnew/> (release dates 2023-10-02, 2024-10-07, 2025-10-07).
39. Release notes of PyYAML 6.0, SQLAlchemy 2.0 and 2.1, pandas 2.0 and 3.0, NumPy 2.0, 2.3, 2.4 and 2.5, pydantic 2.0, httpx 0.28, marshmallow 4.0, setuptools 82 and cryptography 43.0 (official repositories; exact URLs and fetch scripts in the artifact).

40. PyCQA, “Bandit: a security linter for Python,” <https://github.com/PyCQA/bandit>.

Appendix A. Reproducibility

A.1 Artifact

The artifact accompanying this paper contains: the benchmark (family definitions, skills, historical tests, hidden tests, reference solutions, downstream tasks and the verbatim change documents) with the scripts that fetch every document from its primary source; the experiment code (environment builder, sandboxed test runner, model client with cache and cost ledger, the eleven methods and the post-hoc variant, and the analysis); the frozen protocol; the Dockerfile; and all raw run records (per-state library snapshots, hidden-test outcomes before and after each adaptation, task solutions and their test outcomes, and the prompts, responses, token counts, costs and latencies of every model call). Every table and figure in this paper is regenerated from these records by two scripts; an evidence index maps every reported rate and cost to its source records and computation, and a results file holds every interval and paired contrast.

A.2 Environment

Experiments ran in a Docker image based on python:3.12-slim with uv 0.12.21. Each benchmark state declares a CPython minor version and exact package versions; the environment builder creates an isolated virtual environment per specification with uv, using uv-managed CPython builds pinned to 3.9.25, 3.10.21, 3.11.16, 3.12.14, 3.13.15 and 3.14.7 (never a system interpreter), installing binary wheels only where possible, and resolving transitive dependencies as of 30 September 2026. Hidden and historical tests run in a subprocess of the target environment through a dependency-free runner with a 20-second per-test timeout and without API keys in its environment.

A.3 Protocol freeze and amendment

Before the first test-split event was run, a SHA-256 digest over the experiment code, the benchmark, the configuration and the protocol was recorded (488 files; f67be527958fab36..., full value in the artifact; 2026-09-30T14:32:31+0000). The main runner refuses to process test events if the digest differs. All results in Sections 7.1–7.8 were produced under this digest. Amendment A1 (Section 7.9) was added afterwards; its runs are restricted by the runner to the post-hoc variant and logged with the amended digest (78241c9d39c87532...).

A.4 Models, settings and spend

Table 11. Models and settings. Prices are list prices per million tokens (input / cached input / output) used for all cost figures.

Model	Role	Runs	Methods	Sampling	Cutoff	Price (\$/M)
gpt-4.1-mini	primary	3	all 8 + 3 ablations	temperature 0.3	2024-06-01	0.40 / 0.10 / 1.60
gpt-5.4-nano	robustness	1	all 8	reasoning effort low	2025-08-31	0.20 / 0.02 / 1.25
gpt-5.4-mini	robustness	1	4 adaptive	reasoning effort low	2025-08-31	0.75 / 0.075 / 4.50

Every model response is cached on disk under a key derived from the model, the messages, the sampling parameters and a salt that identifies the method, run, family, state, stage and attempt, so that an interrupted run resumes without repeating calls and independent runs do not share responses. Task prompts that are identical across methods share a response. Costs attributed to a method include calls served from the cache at their original price. The hard budget cap of US\$35 was never reached. Recorded spend for the main study and the post-hoc runs was US\$28.50 (gpt-4.1-mini US\$17.82, gpt-5.4-nano US\$3.02, gpt-5.4-mini US\$7.65); development pilots were run separately beforehand.

A.5 Commands

Reproducing Table 3 from the released run records (no API key or model calls; Python 3.10+):

1. `git clone https://github.com/lakshmi-charan/skillshift`
2. `cd skillshift and git checkout v1.0`

3. `pip install -r requirements.txt`
4. `python paper/results_gen.py runs_main paper/out test` writes every metric, interval and contrast of Section 7 to `paper/out/results.json`; Table 3 is its entry `models["gpt-4.1-mini"]["overall"]`, with the intervals of Table 12.
5. `sh paper/make_paper.sh runs_main` (after `npm install` in `paper/`) regenerates the tables and data figures.

The prompts and responses of every model call are in `skillshift_raw_records.zip` (release v1.0; `python analysis/rebuild_cache.py` turns it into the response cache), and `python analysis/verify_digest.py` checks the published code against the recorded digests. A full rerun with new model calls uses `run.py` `check`, `validate`, `labels`, `pilot`, `freeze` and `main` with an API key in a local environment file (never printed or logged); the repository README lists every command.

Appendix B. Additional Results

Table 12. Main test-split results with 95% cluster-bootstrap intervals (gpt-4.1-mini, three runs; %). Retirement is overall retirement.

Method	Retention	Repair	Retirement	Correct	Task success	Task viol.
No skills	–	–	–	–	67.8 (53–81)	48.1 (27–70)
Static library	100.0 (100–100)	0.0 (0–0)	0.0 (0–0)	49.0 (41–57)	22.2 (11–35)	94.4 (85–100)
Doc retrieval	100.0 (100–100)	0.0 (0–0)	0.0 (0–0)	49.0 (41–57)	52.8 (36–69)	59.3 (40–82)
Version-aware retrieval	80.7 (64–94)	0.0 (0–0)	0.0 (0–0)	27.5 (16–41)	70.6 (57–83)	61.1 (42–82)
Full regeneration	91.2 (83–97)	30.8 (21–41)	77.8 (54–100)	74.8 (65–83)	65.6 (55–76)	44.4 (22–73)
Regression-gated (GRASP-style)	100.0 (100–100)	38.7 (20–56)	0.0 (0–0)	78.7 (70–87)	52.8 (33–72)	94.4 (85–100)
Regression-gated + refresh	99.5 (98–100)	49.8 (35–64)	47.2 (19–69)	86.4 (81–91)	60.0 (44–75)	61.1 (29–88)
SkillLedger	82.9 (69–94)	42.8 (29–55)	61.1 (22–85)	68.9 (57–81)	64.4 (51–77)	55.6 (27–80)

Table 13. Development-split results (gpt-4.1-mini, three runs; %, cost in m\$ per family-level transition: 12 transitions from 11 events; Retire. is overall retirement). SkillLedger was revised on these events (Section 5.4), so they are not a fair comparison and are shown for completeness.

Method	Retention	Repair	Retire.	Correct	Task succ.	Task viol.	m\$/event
No skills	–	–	–	–	79.5	40.4	0.0
Static library	100.0	0.0	0.0	68.4	65.2	50.9	0.0
Doc retrieval	100.0	0.0	0.0	68.4	71.2	33.3	0.0
Version-aware retrieval	73.7	0.0	0.0	50.4	84.1	33.3	0.0
Full regeneration	81.0	60.5	100.0	74.5	72.0	31.6	29.4
Regression-gated (GRASP-style)	100.0	58.8	0.0	85.8	73.5	50.9	12.2
Regression-gated + refresh	99.6	76.4	46.7	91.8	80.3	28.1	26.3
SkillLedger	91.7	74.8	100.0	87.0	82.6	24.6	20.8
SkillLedger w/o evidence quarantine	92.9	63.4	0.0	82.4	81.8	28.1	20.4
SkillLedger w/o tombstones	92.1	73.9	93.3	86.9	89.4	14.0	19.1
SkillLedger non-selective review	92.5	75.2	100.0	87.7	89.4	10.5	19.4

Table 14. Share of components correct after adaptation per family (gpt-4.1-mini, test split, %). Families without test events

are omitted.

Family	Static	Vers.-aware	Regen.	Gated	Gated+refr.	SkillLedger
SQL access (SQLAlchemy)	37	0	72	69	88	40
Data frames (pandas)	52	0	81	89	90	81
Numerics (NumPy)	51	15	94	95	98	48
Python standard library	31	26	85	96	91	93
HTTP, schemas, packaging	55	32	61	70	75	59
Password storage (OWASP)	47	47	39	47	74	79
Password verification (NIST SP 800-63B)	64	64	79	64	76	83
TLS configuration (Mozilla + CPython)	63	53	75	79	95	98
Legacy cryptography (NIST + pyca)	70	70	93	70	83	100

Table 15. Accidental losses (component outcomes, three runs pooled) by test event (gpt-4.1-mini). The regression gate without refresh lost none.

Event	Family	Vers.-aware	Regen.	Gated+refr.	SkillLedger
httpx-0.28	HTTP, schemas, packaging	12	29	0	4
marshmallow-4.0	HTTP, schemas, packaging	6	1	0	14
mozilla-tls-6.0	TLS configuration (Mozilla + CPython)	0	1	0	0
nist-800-63b-4	Password verification (NIST SP 800-63B)	0	4	4	4
numpy-2.3	Numerics (NumPy)	3	0	0	12
numpy-2.4	Numerics (NumPy)	21	1	0	42
numpy-2.5	Numerics (NumPy)	0	4	0	0
owasp-2024-12-09	Password storage (OWASP)	0	3	0	0
owasp-2026-03-26	Password storage (OWASP)	0	2	0	0
pandas-3.0	Data frames (pandas)	18	2	0	6
python-3.13	Python standard library	0	10	0	0
python-3.14	Python standard library	9	3	0	2
setuptools-82	HTTP, schemas, packaging	0	0	0	3
sqlalchemy-2.1	SQL access (SQLAlchemy)	0	5	0	27

Appendix C. Model-call stages

Table 16. Stages at which methods call the model. All prompts are stored verbatim with every response in the artifact.

Stage	Used by	Receives	Returns
ledger_setup	SkillLedger (once per family)	contract, code, tests, deployment context, policy text	JSON: grounds per component, evidence per test
impact	SkillLedger (once per family and event)	change document, current policy, ledger, test results	JSON: retain/repair/retire per component, valid/stale per test
repair	SkillLedger	decisions, retained evidence, failures, feedback	module + new tests
regenerate	Full regeneration	contract, interface, documentation bundle	module + tests
propose	Both gates	change document, code, test results	candidate module (2 per skill)
refresh_tests	Gate + refresh	change document, current tests	rewritten test module
task	All methods	task statement, library (and/or documentation, tombstones)	solution module