Agent Skills Under Change: A Lifecycle Benchmark and Coordinate-Guided Updates

Anonymous authors
Paper under double-blind review

Abstract

As agents accumulate reusable Skills, their capabilities depend on artifacts that must evolve with changing facts, tools, and policies. Yet an update that fixes today's failure can silently break working behavior and propagate errors into future tasks. We introduce LifecycleSkillBench, a lifecycle evaluation protocol and benchmark that pairs reproducible environmental changes with affected and protected tasks and tracks recovery, retention, and error propagation across updates. We also propose CoordinateClosure, a coordinate-guided update framework spanning four dimensions: facts, interfaces, execution boundaries, and policies. It localizes edits, checks dependencies, and combines public repair feedback with execution replay to decide whether to adopt an update. In controlled development experiments with matched public inputs, model-call schedules, and output caps, coordinate-structured patches improve first-pass state-check acceptance from 68.8% to 98.4% for Qwen3.7-Plus and from 76.6% to 85.9% for GLM-5.2. In a separate study of four 16-step maintenance streams, the complete controller, including repair and checkpoint restoration, passes the evaluated state and safety checks in 64/64 states, compared with 56/64 for whole-state rewriting. Recorded input and output tokens fall from 958,833 to 42,225, a 95.6% reduction. These experiments show how lifecycle evaluation exposes error persistence across updates, while coordinate-structured patches improve first-pass acceptance and the full CoordinateClosure controller reduces recorded token use relative to whole-state rewriting in the tested maintenance settings.

1 Introduction

Agent Skills store successful procedures, tool-use rules, and domain knowledge for reuse across tasks. Facts expire, APIs migrate, and execution requirements and policies change. Updating this shared state requires repairing affected behavior while preserving the capabilities that future tasks rely on.

Agents can accumulate reusable skills and maintain evolving strategies (Wang et al., 2023; Zhang et al., 2026a), while lifelong benchmarks examine learning and skill evolution across sequential tasks (Zheng et al., 2025b; Zhang et al., 2026b). We focus on updates triggered by environmental changes: can a revised Skill recover affected tasks while preserving capabilities that remain valid, and how do its effects carry into later versions? Answering this question requires linking each change to affected and protected tasks across update histories. We examine three update failures: edits to the wrong state region, inconsistent dependencies, and constraint compliance without goal completion.

Our two main contributions are a lifecycle benchmark and a coordinate-guided update method:

1. LifecycleSkillBench, which pairs reproducible environmental changes with affected and protected tasks, tracking recovery, retention, error propagation, and execution cost across successive updates.
2. CoordinateClosure, which localizes Skill edits along four change dimensions, checks dependencies, and combines public repair feedback with execution replay to decide whether to adopt an update.

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

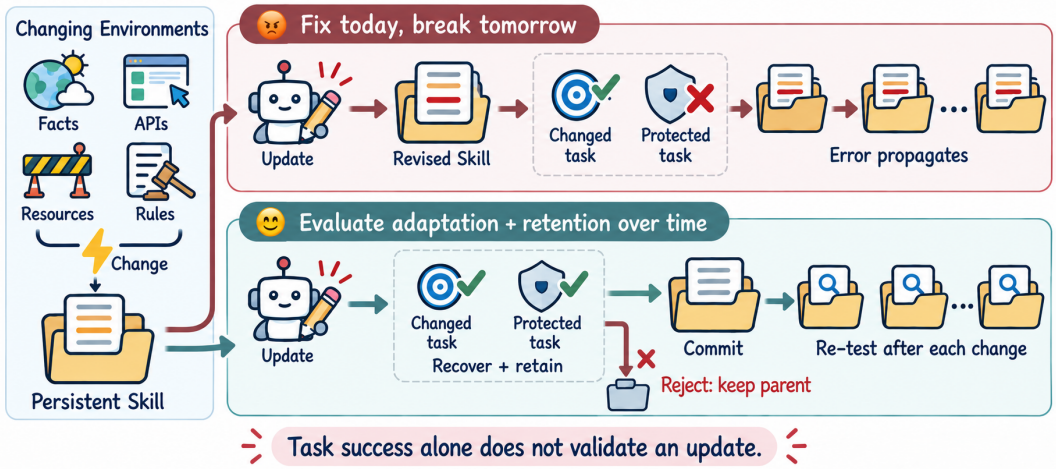


Figure 1: Risks across successive Skill updates. An edit can repair the current task while breaking existing capabilities, with errors persisting into later tasks. Checking recovery and retention after each change informs whether to adopt the new version.

Under matched public inputs, model-call schedules, output caps, and repair procedures, coordinate patches improve first-pass acceptance over generic patches by 29.7 percentage points for Qwen3.7-Plus and 9.4 for GLM-5.2, with similar generation token use. In a separate four-stream maintenance study, the complete controller, including scoped input selection, rollback, and deterministic repair, raises the fraction of states passing the evaluated state and safety checks from 87.5% to 100% relative to whole-state rewriting, with 95.6% fewer recorded input and output tokens.

2 Problem Definition

Let S_t denote the persistent Skill artifact before event e_t, and let E_t contain the evidence available to the updater. The candidate is $\hat{S}_{t+1} = U(S_t, e_t, E_t)$. The evaluator holds affected tasks T_t, which require adaptation, and protected tasks P_t, whose requirements remain unchanged.

We call an update successful when affected tasks recover, protected capabilities are retained, the new behavioral constraints hold, and required output artifacts are produced. A failure-to-change occurs when one or more of these obligations remain unsatisfied despite an attempted update. Rejection preserves the previous state but does not count as successful adaptation when the affected task remains unsolved.

Let $R_t, K_t, B_t, X_t \in \{0, 1\}$ denote affected-task recovery, protected-task retention, behavioral compliance, and executable delivery. R_t and K_t equal one only when every task in T_t and P_t, respectively, passes its declared completion checks; B_t and X_t require all applicable behavioral obligations and required outputs across these tasks to pass. An unresolved check prevents its indicator from being one. Over T evaluated updates, define

$$J_t = R_t K_t B_t X_t, \qquad \text{JointSafeCoverage} = \frac{1}{T} \sum_{t=1}^{T} J_t. \tag{1}$$

Unknowns, timeouts, and parse failures contribute zero to this conservative success rate and remain separately labelled in the denominator. Component scores and exact-state agreement are reported separately; correlated stream states are not independent events.

3 LifecycleSkillBench

LifecycleSkillBench evaluates whether reusable Skills remain effective as their operating environment changes (Figure 2). Each event specifies the before/after environment and pairs affected tasks

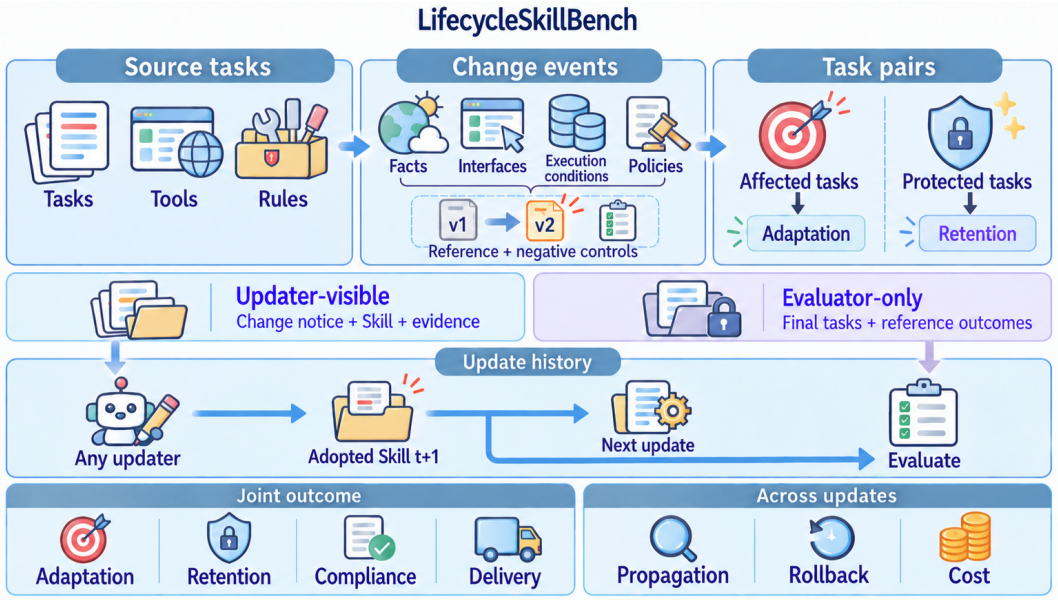


Figure 2: LifecycleSkillBench construction and evaluation. Public tasks, tools, and rules support reproducible environmental changes, paired affected and capability-retention tasks, and persistent update histories. The evaluated method receives the change description, pre-update Skill, and available evidence. The evaluator separately stores test tasks, reference outcomes, and recoverable historical versions, and reports recovery, retention, error propagation, and execution cost.

with tasks whose requirements remain unchanged, testing both recovery and regression. Successive events reuse the adopted Skill to track persistent errors. Appendix B explains the design and coverage audit.

3.1 Evaluation objectives

We ask whether disrupted tasks recover, unaffected tasks still succeed, and an edit's errors persist into later tasks. These define task recovery, capability retention, and error propagation. For example, adapting a broken tool call to changed parameters and completing the same user request constitutes recovery.

3.2 Event construction and splits

Each event records before/after environment configurations, a change description, and available update evidence. Affected tasks require adaptation; protected tasks retain valid goals and check existing capabilities. The evaluator stores reference outcomes and scoring rules. Execution records retain tool calls, final states, errors, and costs; reference outcomes and final-test feedback are withheld from proposal and repair.

The benchmark specifies source-grouped development, calibration, and final-evaluation splits, keeping related variants together. Development data support method design, calibration data determine check configurations, and final-evaluation data compare frozen methods. These are protocol requirements; the reported batches' actual partitions and uses are summarized in Appendix D.1. Tasks used for debugging remain development data.

Event validity. A valid event binds its source version, scoring and unknown-handling rules, and reconstruction procedure. Both a working reference and discriminative negative controls are required. If a changed policy makes the goal infeasible, compliant refusal is a stress-test outcome, not adaptation. Event validity is independent of method rankings.

3.3 Data sources and event construction

We organize 385 event-construction specifications across ToolSandbox (300), TimeWarp (36), AgentDojo (27), SkillsBench (16), and cross-source combinations (6). Figure 3 summarizes construction, executable coverage, and protection checks; source rules appear in Appendix D, Table 3.

Executable development coverage. The reliability bundle contains 19 development conditions (12 ToolSandbox, six historical IANA rule changes, and one synthetic retail-policy condition), with 201 control records including 26 unknown controls. A separate 12-step native-tool stream checks continuity. These are reference/control checks, not learned-method successes or independent test events (Appendix A).

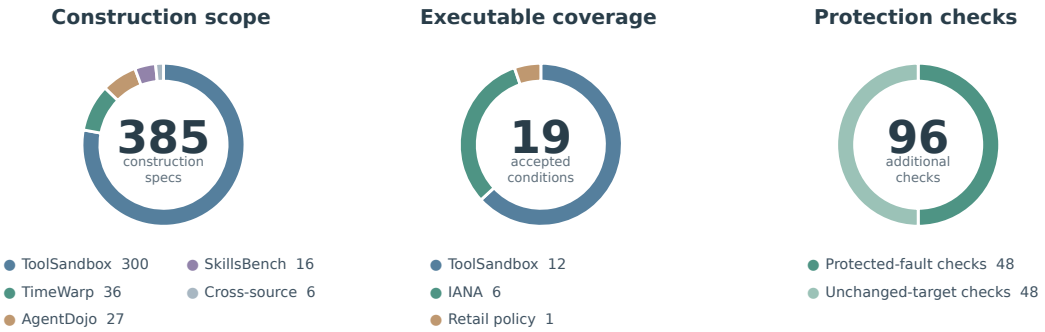


Figure 3: Construction scope and validation depth. Specifications, accepted conditions, control records, and correlated update steps are distinct units, not a pooled sample count.

Beyond the 201 baseline control records, 96 additional checks test protected-behavior discrimination: 48 deliberately faulty protected executions and 48 unchanged-target executions on the same artifacts. These checks deepen validation of existing conditions rather than add independent events. The separate stream checks error carryover and rollback; local protection checks do not establish full-dialogue retention.

3.4 Successive updates

Each adopted Skill becomes the next update's starting point, exposing accumulated errors. The protocol can also test unnecessary edits without environmental change (no-change), readaptation when an earlier rule returns (reversion), and reversal of a failed update (rollback). These name test conditions, not completed sample counts. Independent source events, trajectories, correlated update steps, and repeated trials are counted separately. Appendix D summarizes construction and source-grouping rules; the supplementary manifests bind reconstruction and scoring records.

4 CoordinateClosure

4.1 Coordinates connect edit scope to verification

We organize requirements into facts and provenance (world), tool signatures (interface), runtime and delivery contracts (boundary), and behavioral obligations (policy). An event can affect several domains. A structured (typed) patch records the edit location, replacement, and protected scope; domain labels link edits to checks, and dependency expansion identifies related requirements. Models propose and repair candidates; deterministic checks govern admission (Figure 4).

4.2 From a change notice to an adopted update

The updater receives the current Skill and a public change notice. It first identifies which facts, interfaces, execution conditions, or policies are affected and proposes edits to the corresponding locations. A coordinate names the kind of requirement; a path identifies its location in the Skill; the

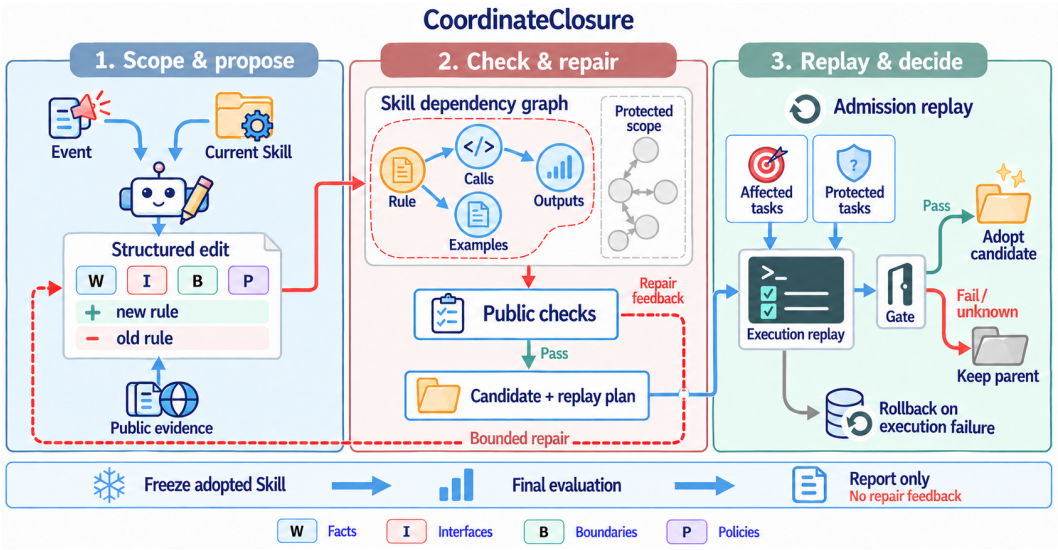


Figure 4: CoordinateClosure update workflow. Changes guide edits and dependency checks, followed by affected-task and capability-retention replay. Public-check failures may trigger repair; failed or unknown execution checks retain the old version. The lower track evaluates frozen updates without repair feedback. W/I/B/P denote facts, interfaces, execution boundaries, and behavioral policies.

replacement specifies the proposed change. Protected locations define what must remain valid. This separates deciding where a change belongs from regenerating the whole artifact.

For example, suppose a payment API changes both its argument name and its amount unit. The interface coordinate identifies the affected call binding, while the update must also adjust the amount conversion and any examples using the old convention. If the return contract changes as well, result interpretation joins the checking scope. Unrelated payment rules remain protected. A single event can therefore induce several linked edits: coordinates organize their responsibilities, and declared dependencies connect their checks. The current checker uses explicit contract rules; it does not infer arbitrary semantic dependencies.

The controller then follows declared dependencies to check related rules, calls, and examples. A failed public check produces repair feedback containing the error location and violated requirement. Within the repair budget, the updater revises the candidate. Replay executes affected and protected tasks with the resulting Skill; adoption requires recovery, retention, compliance, and delivery to pass together. A failed or unknown check retains the parent. If execution requires rollback, the controller restores a verified checkpoint. Final evaluation tests the frozen adopted Skill without returning feedback to the updater. Appendix B describes version binding and decision records.

Dependency expansion. An edit can affect related content: changing a refund rule may require checking call arguments, examples, and output checks. Starting from directly edited items C_j, we follow declared dependencies in D_j. An edge $u \to v$ means that editing u requires checking v. The resulting dependency closure is

$$\text{Closure}_j(C_j) = C_j \cup \{v : \exists u \in C_j,\ u \rightsquigarrow v \text{ in } D_j\}. \tag{2}$$

Items are rewritten only when needed. Closure terminates on the finite graph and is idempotent. Replay then tests affected and protected tasks and can expose execution errors missed by the declared dependencies.

5 Experiments

We first validate event execution and fault detection, then compare first-pass edits, continual-maintenance cost, and admission efficiency. Table 1 summarizes the controlled studies; Appendix C gives configurations and cost accounting.

Research questions. **RQ1 (Benchmark reliability):** Are events reproducible, and do checks detect intended failures and regressions? **RQ2 (First-pass edits):** Do coordinate-structured patches improve initial acceptance under matched generation settings? **RQ3 (Continual maintenance):** Does the complete controller reduce persistent errors and token use? **RQ4 (Admission efficiency):** Can dependency filtering reduce replay executions while preserving admission decisions? Supplementary diagnostics examine repair and distinguish compliance from task completion (Appendix G).

5.1 Experimental setup

Units and comparisons. An event is an environmental transition; a task is a user goal; a trajectory records its dialogue and tool calls. The 385 construction specifications are separate from evaluated Skills. Controlled comparisons use four 16-step development streams per arm and backbone. Typed and generic patches share public inputs, a 2,048-token output cap, and repair procedures. Appendix F gives model settings, scoring procedures, and a worked example; Appendix G reports policy diagnostics.

Implemented comparisons. We implement three update strategies in a common harness: whole-state rewriting generates a complete replacement; generic local patches emit changes without coordinate labels; coordinate patches add change-domain and scope constraints. Full CoordinateClosure additionally combines dependency checks, repair feedback, execution replay, and checkpoint restoration. Table 1 groups these comparisons and the replay-only versus dependency-filtered admission ablation.

Table 1: Completed controlled development comparisons. A and B each use four 16-state streams; C reports the identical pattern on one retail and two calculation fixtures, each with 17 constructed candidates. Different baselines and units are not pooled into an overall ranking.

Metric	Baseline strategy	Evaluated variant
A. First-pass updates	Generic patch	Coordinate patch
Qwen initial state checks	44/64	**63/64**
GLM initial state checks	49/64	**55/64**
Qwen repair-required updates	20	**1**
GLM repair-required updates	15	**9**
Qwen final, same deterministic repair	64/64	64/64
GLM final, same deterministic repair	64/64	64/64
B. Continual maintenance	Whole-state rewrite	CoordinateClosure
CoordinateClosure includes repair and checkpoint restoration.		
Stream exact-safe states	56/64	**64/64**
Observed stream total tokens	958,833	**42,225**
C. Admission checks (per fixture)	Replay only	Dependency-filtered replay
Admission workflow executions	32	**8**
Invalid candidates rejected	15/15	15/15
Joint successes	1/17	1/17

Admission versus evaluation. Admission tasks decide whether to adopt an update; final tasks assess recovery and retention after freezing, without feeding back into repair. This is a formal protocol requirement, not a property established for all development runs. Database state, required actions, compliance, and goal completion are checked separately. All planned runs remain in the denominator. A joint outcome is unknown only when no required check has failed and at least one remains unresolved.

Statistical reporting. Specifications, candidates, trajectories, and scored outcomes are counted separately. With y successes and u unknowns among n runs, $[y/n, (y+u)/n]$ gives bounds under alternative resolutions of unknown outcomes. These bounds do not quantify sampling uncertainty. Source-clustered bootstrap intervals are reported for the paired policy-state diagnostic in Appendix G; the main controlled comparisons are descriptive results over correlated streams.

5.2 RQ1: Benchmark execution and fault detection

Separate isolated installations reproduce 216 ToolSandbox controls and nine synthetic-retail outcomes, including two expected unknowns (Appendix I).

Each event binds source versions, before/after requirements, affected and protected tasks, and scoring rules. Working references test feasibility, obsolete or deliberately incorrect solutions test fault detection, and injected operational failures test unknown handling. The 19-condition bundle contains 201 control records, including 26 unknown controls; a separate 12-step native-tool stream tests state carryover (Appendix A).

Across the 12 ToolSandbox conditions, the scorer detects 48/48 injected protected-branch errors, and all 48 paired unchanged-target controls pass. Together, reconstruction and these 96 additional controls support benchmark reliability on the covered conditions: execution outcomes are reproducible, and scoring distinguishes injected protected-behavior faults from unchanged behavior.

5.3 RQ2: First-pass acceptance and repair demand

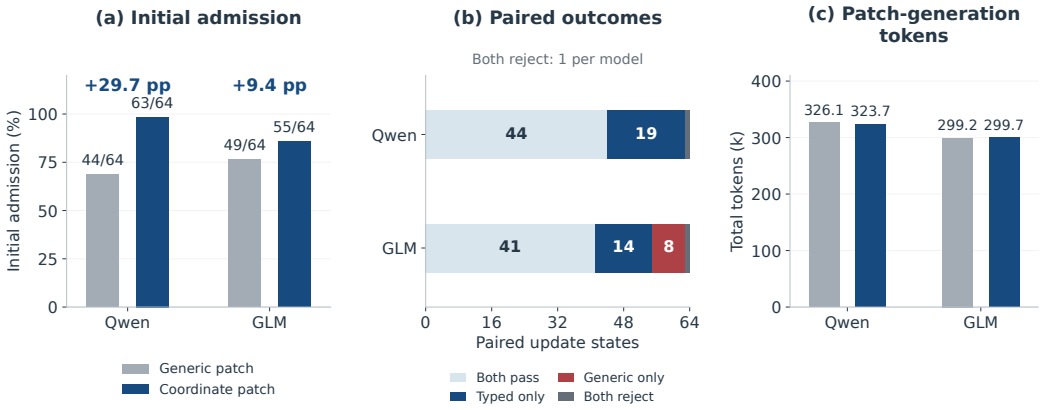


Figure 5: Coordinate patches pass more first checks at similar recorded generation cost. (a) Acceptance before repair. (b) Blue: only coordinate patches pass; red: only generic patches pass. GLM gains 14 states and loses eight, a net gain of six. (c) Generation tokens are similar. Each arm contains four 16-state development streams (52 model-generated updates and 12 rollbacks); shared repair brings both formats to 64/64.

Coordinate patches require fewer repairs while using similar numbers of generation tokens: repair-required states fall from 20 to one for Qwen and from 15 to nine for GLM (Figure 5). Evidence, call schedules, 2,048-token output caps, retries, checks, and repair procedures are matched. Coordinate patches specify the edit's dimension, original content, and replacement; generic patches lack coordinate-type constraints. The comparison therefore measures representation together with its parser and scope constraints.

Before repair, acceptance rises from 44/64 to 63/64 for Qwen and from 49/64 to 55/64 for GLM. Qwen has 19 coordinate-only passes and no generic-only passes; GLM has 14 and eight, respectively. Thus GLM improves in aggregate, although generic patches pass some states that coordinate patches fail. Each arm contains four correlated 16-state development streams: 52 model-generated updates and 12 deterministic rollbacks. On generated updates alone, counts are 32/52 versus 51/52 for Qwen and 37/52 versus 43/52 for GLM. Shared deterministic repair brings both formats to 64/64; the measured benefit is fewer states requiring repair, with tied final acceptance.

Across 52 generated updates per arm, parse failures fall from 11 to one for Qwen and from ten to four for GLM. Fewer parsing failures contribute to the first-pass improvement; acceptance also depends on state and scope checks.

5.4 RQ3: Error persistence and maintenance cost

The complete controller reduces persistent state errors and recorded token use relative to whole-state rewriting.

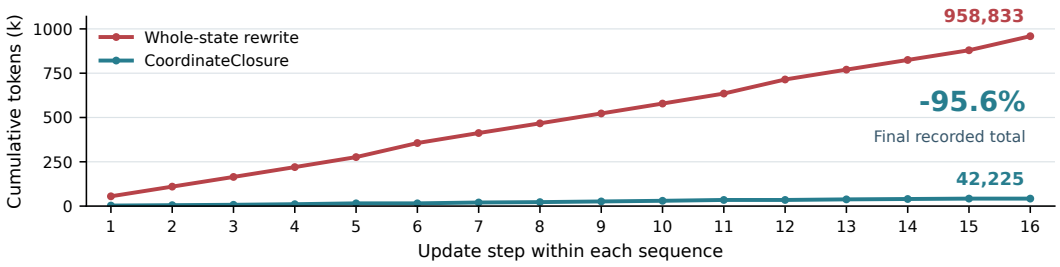


Figure 6: Full controller vs. whole-state rewriting: cumulative recorded input and output tokens across four 16-step Qwen streams. The controller uses 42,225 tokens versus 958,833 (95.6% less). State-by-state outcomes appear in Appendix Figure 7.

In a separate Qwen development study with four 16-step streams, the coordinate controller reaches 64/64 exact-safe states, compared with 56/64 for whole-state rewriting. One whole-state generation error persists through steps 8–15 and contaminates a rollback checkpoint. The benchmark thus detects damage that would be missed by checking only the latest edit's target. The controller passes 63/64 initial state checks, including 12 deterministic rollbacks, and repairs one rejected proposal using deterministic rules derived from the public requirements. Retention here measures exact preservation of unchanged state fields; retention on complete tasks is evaluated separately.

The controller records 42,225 input and output tokens, compared with 958,833 for whole-state rewriting (95.6% less). Rewriting reads and emits the full state and calls the model on all 64 steps; the controller supplies only the current event, generates scoped edits on 52 steps, restores checkpoints directly on 12 rollbacks, and applies public checks and deterministic repair. One additional failed rewrite request has no usage record; totals therefore represent observed usage. Appendix C details accounting and configurations.

These savings arise in large-state, small-change maintenance. For example, an interface event changes `years` from integer to string, requiring `20` to become `"20"`. A scoped patch updates the call without regenerating unrelated tool definitions, templates, and facts. Together with direct checkpoint restoration, this reduces repeated generation across updates.

Task-level audits complement these state checks: one protected-task dialogue reports incorrect savings despite correct database actions (Appendix G.5).

5.5 RQ4: Admission efficiency

Dependency filtering reduces admission replays while preserving decisions within the tested finite contracts. We compare four checking configurations on the same 17 constructed candidates for a retail interface migration: all 16 combinations of endpoint version, two argument-name versions, and protected-operation behavior, plus one missing candidate. The dependency checker tests the public endpoint–argument contract before execution. Rules and candidate hashes are frozen before this run; two admission workflows are separate from two final workflows. This is a development component ablation.

The dependency checker detects all 12 endpoint–argument inconsistencies. Adding it before replay preserves every admission decision while reducing admission workflow executions from 32 to 8 (75%). Replay rejects all 15 candidates that fail final joint requirements; the one jointly valid candidate is accepted. Static checks therefore remove known inconsistencies cheaply, while replay checks

task behavior beyond the declared dependency contract. All four configurations have joint success of 1/17: filtering protects against invalid updates but does not repair them.

Two additional ToolSandbox-derived calculation fixtures reproduce the same component pattern: dependency filtering reduces admission workflow slots from 32 to 8 per fixture while every arm remains at 1/17 joint success. Appendix H reports all outcomes and execution-cost accounting.

6 Related Work

Skill acquisition and evolution. Voyager builds a reusable skill library, SkillWeaver discovers and refines skills as APIs, and ACE incrementally maintains strategy playbooks (Wang et al., 2023; Zheng et al., 2025a; Zhang et al., 2026a). Reflexion and Self-Refine use feedback to improve behavior or outputs (Shinn et al., 2023; Madaan et al., 2023). CoordinateClosure connects scoped edits to dependency checks and affected/protected-task replay, making version adoption an explicit decision.

Lifelong agent evaluation. LifelongAgentBench evaluates learning across interdependent interactive tasks; SkillFlow studies sequential skill discovery, patching, and reuse (Zheng et al., 2025b; Zhang et al., 2026b). LifecycleSkillBench focuses on environmental transitions, pairing affected and protected tasks to track recovery, retention, and error propagation across adopted versions. This adaptation–retention objective also connects to continual learning and model editing; our update targets are external Skill artifacts (Appendix E).

Execution-based evaluation and update validation. BFCL, τ^2-Bench, ToolSandbox, and AgentDojo evaluate tool calls, user goals, stateful interaction, and adversarial behavior (Patil et al., 2025; Barres et al., 2025; Lu et al., 2025; Debenedetti et al., 2024). These environments support execution-grounded assessment. Our protocol links selected sources to reproducible changes, paired task checks, and persistent version histories; the controller uses admission replay to decide whether to adopt a candidate.

7 Discussion and Future Work

The controlled studies show three complementary benefits: coordinate patches improve first-pass acceptance at similar generation cost, the complete controller reduces recorded token use relative to whole-state rewriting, and dependency filtering reduces admission replays without changing decisions. The maintenance savings are particularly relevant when a large Skill state undergoes small changes: scoped edits and checkpoint restoration avoid regenerating unchanged content. Together, these results show how structured updates and execution checks can reduce the work required to maintain reusable Skills across changes.

These findings concern controlled, partly development-visible settings. Future work should evaluate the complete controller on source-disjoint tasks, assess full-dialogue capability retention, and measure end-to-end cost and latency with complete usage records.

8 Conclusion

Reusable Skills must evolve with their environments while preserving behavior that remains valid. This work contributes both an evaluation protocol for that process and a coordinate-guided update framework. LifecycleSkillBench pairs reproducible changes with affected and protected tasks and preserves update histories, making recovery, regression, and error propagation observable across versions. CoordinateClosure connects scoped edits and dependency checks to public repair feedback and execution-based adoption decisions.

The controlled studies reveal complementary benefits. Coordinate patches improve first-pass acceptance at similar generation cost, reducing repair demand. Across four maintenance streams, the complete controller passes the evaluated state and safety checks in 64/64 states versus 56/64 for whole-state rewriting, with 95.6% fewer recorded tokens. Dependency filtering reduces admission replays by 75% while preserving decisions. Together, these findings show how evaluating successive updates and matching edit scope to change requirements can make Skill maintenance more reliable and economical in the tested settings.

Reproducibility Statement

The supplementary package provides core update code, frozen records for the main controlled comparisons, and an offline verification script. The script checks file integrity, recorded outcomes, and token accounting, and exercises the core update operations. Appendix C details the main-result accounting. Separate native reconstruction experiments are reported in Appendix I.

Ethics Statement

All lifecycle events use public sources or simulated environments. No new human-subject data are collected. Policy and transaction tests run in sandboxes and should not be interpreted as production safety certification.

AI Use Statement

Generative AI assisted with literature research, experimental design and implementation, result analysis, translation, and illustrations. Experimental model use is described in the relevant sections. The authors take responsibility for all content and evidence.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment, 2025. URL https://arxiv.org/abs/2506.07982.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K. Dokania, Philip H. S. Torr, and Marc'Aurelio Ranzato. On Tiny Episodic Memories in Continual Learning, 2019. URL https://arxiv.org/abs/1902.10486.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 82895–82920. Curran Associates, Inc., 2024. doi: 10.52202/079017-2636. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a Generalist Agent for the Web. In *Advances in Neural Information Processing Systems*, volume 36, pp. 28091–28114. Curran Associates, Inc., 2023. doi: 10.52202/075280-1220. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/5950bf290a1570ea401bf98882128160-Abstract-Datasets_and_Benchmarks.html. Datasets and Benchmarks Track.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The Hitchhiker's Guide to Testing Statistical Significance in Natural Language Processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1383–1392. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1128. URL https://aclanthology.org/P18-1128/.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How Capable are Web Agents at Solving Common Knowledge Work Tasks? In *Proceedings of the 41st International*

Conference on Machine Learning, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 2024. URL https://proceedings.mlr.press/v235/drouin24a.html.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving Factuality and Reasoning in Language Models through Multiagent Debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé, III, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64 (12):86–92, 2021. doi: 10.1145/3458723. URL https://doi.org/10.1145/3458723.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/6507b115562bb0a305f1958ccc87355a-Abstract-Conference.html.

Internet Assigned Numbers Authority. Time Zone Database: tzdata2022a, 2022. URL https://data.iana.org/time-zones/releases/tzdata2022a.tar.gz. Versioned data archive.

Internet Assigned Numbers Authority. Time Zone Database: tzdata2023c, 2023. URL https://data.iana.org/time-zones/releases/tzdata2023c.tar.gz. Versioned data archive.

Internet Assigned Numbers Authority. Time Zone Database: tzdata2025a, 2025. URL https://data.iana.org/time-zones/releases/tzdata2025a.tar.gz. Versioned data archive.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-world Github Issues? In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling Declarative Language Model Calls into State-of-the-Art Pipelines. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/f1cf02ce09757f57c3b93c0db83181e0-Abstract-Conference.html.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114. URL https://doi.org/10.1073/pnas.1611835114.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating Multimodal Agents on Realistic Visual Web Tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.50. URL https://aclanthology.org/2024.acl-long.50/.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Zhizhong Li and Derek Hoiem. Learning Without Forgetting. In *Computer Vision – ECCV 2016*, volume 9908 of *Lecture Notes in Computer Science*, pp. 614–629. Springer, 2016. doi: 10.1007/978-3-319-46493-0_37. URL https://doi.org/10.1007/978-3-319-46493-0_37.

Adam Liska, Tomas Kocisky, Elena Gribovskaya, Tayfun Terzi, Eren Sezener, Devang Agrawal, Cyprien De Masson D'Autume, Tim Scholtes, Manzil Zaheer, Susannah Young, Ellen Gilsenan-Mcmahon, Sophia Austin, Phil Blunsom, and Angeliki Lazaridou. StreamingQA: A Benchmark for Adaptation to New Knowledge over Time in Question Answering Models. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 13604–13622. PMLR, 2022. URL https://proceedings.mlr.press/v162/liska22a.html.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as Agents. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/e9df36b21ff4ee211a8b71ee8b7e9f57-Abstract-Conference.html.

David Lopez-Paz and Marc' Aurelio Ranzato. Gradient Episodic Memory for Continual Learning. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/f87522788a2be2d171666752f97ddebb-Abstract.html.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and Editing Factual Associations in GPT. In *Advances in Neural Information Processing Systems*, volume 35, pp. 17359–17372. Curran Associates, Inc., 2022. doi: 10.52202/068431-1262. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/6f1d43d5a82a37e89b0665b33bf3a182-Abstract-Conference.html.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-Editing Memory in a Transformer. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.07229.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for General AI Assistants. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/25ae35b5b1738d80f1f03a8713e405ec-Abstract-Conference.html.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Fast Model Editing at Scale. In *International Conference on Learning Representations*, 2022a. URL https://arxiv.org/abs/2110.11309.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. Memory-Based Model Editing at Scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 15817–15831. PMLR, 2022b. URL https://proceedings.mlr.press/v162/mitchell22a.html.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model Cards for Model Reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pp. 220–229, 2019. doi: 10.1145/3287560.3287596. URL https://doi.org/10.1145/3287560.3287596.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems, 2023. URL https://arxiv.org/abs/2310.08560.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 126544–126565. Curran Associates, Inc., 2024. doi: 10.52202/079017-4020. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/28e50ee5b72e90b50e7196fde8ea260e-Abstract-Conference.html.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4902–4912. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.442. URL https://aclanthology.org/2020.acl-main.442/.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris Maddison, and Tatsunori Hashimoto. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/7274ed909a312d4d869cc328ad1c5f04-Abstract-Conference.html.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.03768.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully Test-time Adaptation by Entropy Minimization. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2006.10726.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2308.08155.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying LLM-Based Software Engineering Agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754. FSE 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems*, volume 37, pp. 52040–52094. Curran Associates, Inc., 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers. In *International Conference on Learning Representations*, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/3339f19c5fcee3ad74502947a32be9e6-Abstract-Conference.html.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024b. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757. Curran Associates, Inc., 2022. doi: 10.52202/068431-1508. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/82ad13ec01f9fe44c01cb91814fd7b8c-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b126cc38b8638e07bef37e7b2bb72bf-Abstract-Conference.html.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639(8055):609–616, 2025. doi: 10.1038/s41586-025-08661-4. URL https://doi.org/10.1038/s41586-025-08661-4.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. In *International Conference on Learning Representations*, 2026a. URL https://arxiv.org/abs/2510.04618.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. SkillFlow: Benchmarking Lifelong Skill Discovery and Evolution for Autonomous Agents. *arXiv preprint arXiv:2604.17308*, 2026b. URL https://arxiv.org/abs/2604.17308.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web Agents can Self-Improve by Discovering and Honing Skills. *arXiv preprint arXiv:2504.07079*, 2025a. URL https://arxiv.org/abs/2504.07079.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. LifelongAgentBench: Evaluating LLM Agents as Lifelong Learners. *arXiv preprint arXiv:2505.11942*, 2025b. URL https://arxiv.org/abs/2505.11942.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large Language Models Are Human-Level Prompt Engineers. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2211.01910.

A Updated benchmark reliability and artifact-level comparison

Pinned factual-rule sources. The six core IANA conditions execute archived releases 2022a and 2023c; the four additional groups compare 2023c with 2025a (Internet Assigned Numbers Authority, 2022; 2023; 2025). Reconstruction uses the archive hashes in each frozen manifest, not the latest database. Release labels attached to individual changes identify when a rule or historical correction was recorded; they need not be the two archives executed in a comparison. In particular, the Vostok event tests a correction to the database's historical representation, not a claim that the old database offset describes the actual historical transition.

Supplementary controls retain 12 existing conditions: all 48 injected protected-branch errors are detected and all 48 paired unchanged-target checks pass. A further 120/120 historical positive/negative controls are reproduced. Only the first 96 checks are new; no conditions or model calls are added.

The development reliability bundle contains 19 conditions across 14 primary provenance groups: 12 ToolSandbox conditions from eight groups, six historical IANA rule changes from five jurisdiction groups, and one synthetic retail-policy condition. Shared protected tasks and variants preclude treating every condition as independent. Of 201 reference/control records, 26 deliberately test unknown handling: 12 ToolSandbox, 12 IANA, and two synthetic-retail controls. These counts exclude a separate 12-step tool-execution stream and all method-generation trials. The IANA workload executes historical time-zone rules rather than judging free-text factual answers. A separate Linux worker-isolation replay completes 60 conversions and 36 denied attempts to access evaluator material; this validates that runner, not the isolation of earlier experiments. These are development conditions, not a source-disjoint final test.

Two-model interface-update diagnostic. One further retail-tool migration uses separate admission and final tasks, two models (`qwen3.7-plus-2026-05-26`, `glm-5.2`), requested seeds 11 and 29, temperature zero, a 2,048-token output cap, and no repair. Twelve generation requests compare whole rewriting, untyped patches, and coordinate patches; the fourth arm reuses each coordinate candidate with admission replay. All candidates and decisions are frozen before final replay. The evaluator supplies reference workflows, so this tests generated call-binding artifacts rather than complete model-planned conversations.

For Qwen, affected-task passes are 2/2, 0/2, 2/2, and 2/2 across the four arms; all protected checks pass. The untyped proposals rename the endpoint but omit required argument renames. For GLM, the corresponding counts are 1/2, 2/2, 2/2, and 2/2; the remaining whole-rewrite candidate has an

invalid schema, leaving its affected and protected outcomes unknown. Overall, 32 planned final slots contain 28 passes, two failures, and two unknowns, with 30 actual execution receipts. All eight admission checks pass. Independent raw-response parsing and native replay reproduce the records. Generation consumes 23,067 tokens; coordinate generation is slightly more expensive in this batch. This single-mechanism diagnostic supports an observed benefit of complete structured edits, not broad superiority, full-task retention, or an incremental gate advantage. Operationally failed earlier attempts remain archived separately.

Cross-source rule-version execution check. On the six accepted IANA development conditions, 36 generation requests compare the same three formats with two models and seed 11; the gated arm reuses typed candidates. Admission and final date/zone inputs are distinct. This experiment generates version-selection artifacts, not new time-zone rule implementations. All 24 admission conversions pass. Of 144 final slots, 126 execute and pass and 18 remain unknown: all six GLM whole-rewrite responses select the correct rule versions but also emit metadata rejected by the fixed schema. This is format nonconformance, not evidence of inferior factual adaptation. All valid methods tie, and the gate accepts every typed candidate. Independent recompilation of the official rules and replay reproduce all records. Total generation usage is 10,647 tokens. The result checks cross-source executability, not independent-test superiority or full conversational retention.

Separating format compliance from execution correctness. To test whether the unknown outcomes arise solely from extra descriptive fields, we conduct a post-hoc format-compatibility analysis. The same deterministic rule is applied to all 36 original responses: remove only fields outside the declared schema, without filling missing fields, changing required values, repairing invalid JSON, or consulting reference answers to modify candidates. The normalization policy and original-response hashes are fixed before rescoring. Six GLM whole-rewrite responses become executable; all 36 candidates are valid, and all 24 admission and 144 final conversions are actually executed and pass. The original strict result of 126 passes and 18 unknowns is retained separately. No additional model requests are made, so generation usage remains 10,647 tokens. This analysis shows that all methods select the correct rule versions on this workload and that the original difference concerns format compatibility, not factual adaptation. Because the rule was developed after inspecting the original results, this is reported as a sensitivity analysis rather than a replacement for the frozen scoring protocol.

B Design rationale and evidence for the two contributions

We evaluate an update through its execution outcomes: it should recover affected tasks, satisfy the new behavioral constraints, and retain unaffected capabilities. Joint update checks assess whether these requirements hold together. To describe the scope of a change, we represent an event as

$$\Delta_t = (\Delta_t^w, \Delta_t^i, \Delta_t^b, \Delta_t^p), \tag{3}$$

where the superscripts denote world facts, tool interfaces, execution boundaries, and behavioral policies. The world coordinate covers changing facts and their provenance, such as dates, entity attributes, and external knowledge; it excludes tool signatures, runtime resources, and behavioral rules. Coordinates localize edits and verification requirements; an event can activate several coordinates.

For example, changing a policy from five passengers per order to one makes a three-passenger booking invalid. If separate orders are permitted and authorized, the updated Skill must complete three valid bookings to satisfy the goal. Operational errors, parse failures, timeouts, and unknown states remain in the denominator; unknown is never silently counted as safe.

Why evaluate updates? A reusable Skill carries decisions from one task to the next. A task-level success can therefore hide two different costs: damage to another valid task and an error inherited by later versions. This motivates three connected design choices. Before/after environments establish what adaptation is required; affected/protected task pairs expose the trade-off within an update; persistent histories expose consequences across updates. These choices define a method-independent evaluation problem. Any updater can be evaluated through the same task and history interface.

Validity before scale. A valid event specifies the old environment, intervention, and new requirements; provides a working reference and appropriate negative controls; includes protected tasks; and

binds scoring, unknown handling, provenance, versions, split, and reconstruction. Negative controls include stale solutions, no-ops, and damaged protected states. Infeasible goals are labelled policy stress tests. Together, these checks establish feasibility and scorer discrimination.

Coverage is an auditable profile. Coverage is recorded by change mechanism, source group, task obligation, and temporal condition, with evidence labelled specification-only, contract-level, or tool-executed. The earlier ToolSandbox-plus-policy audit covered 13 conditions; the expanded 19-condition accounting appears in Appendix A. Related variants share provenance, and the 385-specification inventory remains separate from executable validation.

Coordinates organize a larger update process. Coordinates label the responsibilities of edits and checks. Structured patches make the changed locations explicit; dependency expansion identifies related items to inspect; public checks identify concrete repair obligations; replay supplies the adoption evidence. The implemented workflow checker classifies failures including unavailable endpoints, argument names and types, unit conversion, resource prerequisites, result selection, export type, and edits to protected scope. This is a diagnostic classification, not an empirically validated learned classifier. Its dependency graph covers a declared finite contract language, not all semantic dependencies.

Repair information and decision records. The implemented repair packet contains the public contract, current candidate, raw proposal, errors with paths and explanations, dependency obligations, and a bounded repair instruction. Its purpose is to make the next edit answer a specific check failure. It excludes final reference answers and final-test feedback. Candidate/state hashes and evaluator versions link proposal, execution, and acceptance records. These records support decision tracing. The reported workflow study permits at most one learned repair with the same public feedback for each proposal format. Utility checks cover a narrower schema contract and should not be described as the full dependency compiler.

Version binding and proposal roles. Candidate bytes, the parent-state hash, the public evidence packet, and compiler/evaluator versions are bound by a deterministic identifier. Unknown operators, unsupported fields, parent drift, or expected-hash mismatch fail closed. Rejection leaves the parent unchanged; rollback returns to a hash-verified checkpoint. Where multiple agents propose or review typed claims, agreement cannot replace execution checks, and no new patch may be synthesized after admission without revalidation. This role separation is an implementation contract; the reported comparisons do not isolate a multi-agent benefit.

B.1 From design choices to measurable benefits

The design follows the failure path of an update: an incorrect edit scope, an omitted dependency, an unsuccessful repair, or an incorrect adoption decision. Table 2 makes each component's output and evaluation target explicit. These components serve complementary roles within the update procedure.

C Historical development denominators and cost accounting

Why local maintenance saves tokens. The state contains reusable tool definitions, call templates, and factual records, while an individual event changes only a small part. For example, one interface event changes the type of `years` from integer to string: the corresponding call must change from `20` to `"20"`, without regenerating unrelated entries. Scoped edits avoid repeatedly reading and emitting the full state; checkpoint restoration similarly avoids model-generated rollback. Recorded input tokens decrease from 517,999 to 30,414, and output tokens from 440,834 to 11,811. This identifies large-state, small-change maintenance as the setting in which the complete pipeline is particularly economical. It does not imply the same saving over another local-patch method; the matched representation comparison has similar token use.

Both studies use four 16-step development streams. States share histories and cannot be treated as 64 independent samples for significance tests. The eight incorrect rewrite states reflect one erroneous

Table 2: Implemented mechanisms and the evidence needed to assess them.

Mechanism	Operational output	Evaluation target and current evidence
Responsibility labels and structured edits	Edit locations, owners, and replacements	First-pass contract acceptance; supported by the matched representation-and-validation comparison.
Dependency checks	Related requirements and identified contract violations	Missed obligations and protected-scope changes; evaluated within the declared finite contracts.
Diagnostic repair packet	Candidate, public contract, error locations, and repair requirements	Repair success and additional calls; shared-feedback workflow diagnostics, no exclusive method advantage established.
Replay admission	Adopt or retain the previous version, with execution receipts	False admissions and unnecessary rejections within the reported development fixtures.
Version-bound records	Candidate/state hashes and evaluator version	Decision traceability and reproducibility; these properties do not imply higher task success.

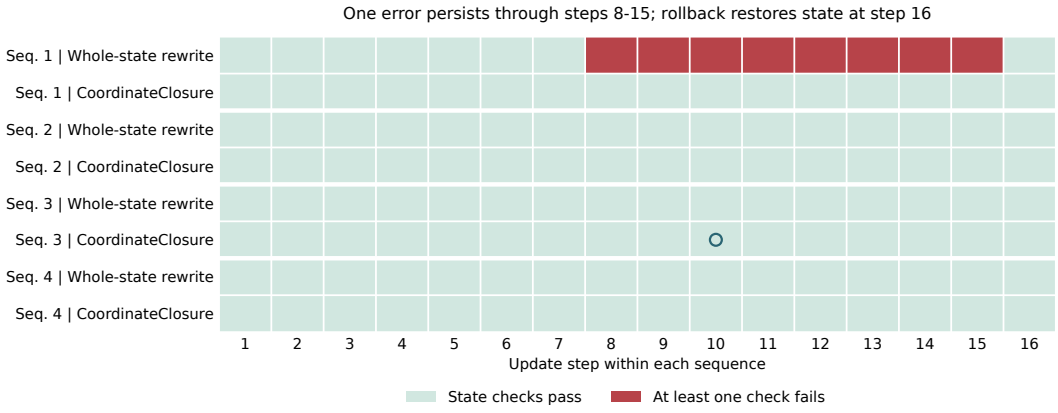


Figure 7: State checks across four 16-step Qwen streams. One rewrite error persists through steps 8–15 of sequence 1; step 16 restores the initial state. The open circle marks deterministic repair at sequence 3, step 10. CoordinateClosure passes 64/64 states after repair, compared with 56/64 for rewriting; the eight failures share one error.

field persisting through steps 8–15. Controlled scores check changed fields, unchanged fields, and complete expected-state equality, rather than conversational success or general behavioral safety.

The matched-format study uses `qwen3.7-plus-2026-05-26` and `glm-5.2`, temperature 0, a 2,048-token output cap, 240-second timeout, at most two operational retries, and concurrency 8; no provider seed is controlled. Visible-input hashes match for all 64 paired states, with 52 successful model responses per arm. Generic/typed parse failures are 11/1 for Qwen and 10/4 for GLM, so initial acceptance includes serialization and constraint-check effects. Recorded total tokens are 326,059/323,747 for Qwen and 299,202/299,748 for GLM. Each generic arm has one additional failed request without usage; missing usage is not interpreted as zero cost.

The rewrite comparison uses the same Qwen version and an 8,192-token output cap, but differs in input scope, output representation, rollback scheduling, and repair. Rewrite input/output tokens are 517,999/440,834 versus 30,414/11,811 for the controller, with 64 versus 52 successful calls. Summed successful-request durations are 7,576.884 versus 248.370 seconds, not concurrent wall-clock latency; CPU checking, materialization, and repair are not separately timed.

The public event language fully specifies state transitions: a deterministic public interpreter reproduces all 64 expected states without a model. Perfect repaired results validate execution reliability within that language; first-pass contract adherence, repair demand, and state-copying cost remain

separately observable. This does not show that a model can infer semantic obligations absent from the public evidence.

D Construction inventory and release checks

Five manifests contain 385 specifications: 300 ToolSandbox, 36 TimeWarp, 27 AgentDojo, 16 SkillsBench, and six cross-source combinations. A specification describes construction and evaluation, not a completed model execution. Counts follow source-selection and variant-generation rules, not balanced sampling: ToolSandbox contributes 75 scenarios with four interface variants each. Related variants are not independent samples. Table 3 summarizes the construction mechanisms.

This inventory is reported separately from controlled studies, τ^2 policy interactions, and SkillFlow execution diagnostics; their counts cannot simply be added. The experiments identify executed tasks, models, and trajectories per batch.

Table 3: Source composition of the 385 construction specifications, grouped by the selected construction mechanisms.

Source	Construction mechanism	Specs.
ToolSandbox	75 tool-use scenarios, each with four interface variants	300
TimeWarp	Tasks paired across environment versions; factual changes require separate verification	36
AgentDojo	Versioned injection-task additions or updates	27
SkillsBench	Four source tasks, each with four execution-condition changes	16
Cross-source	Combinations reusing components from existing sources	6

The five manifests use the following construction rules:

- ToolSandbox selects 75 source scenarios and creates four perturbations per scenario: tool description, argument type, argument description, and tool name. The 300 variants stay grouped in three splits of 25 source scenarios (100 variants) each.
- TimeWarp binds 36 source tasks to environment versions 1 and 5. The manifest holds intent and backend content constant; a version change alone therefore does not demonstrate changed facts. Classification requires inspecting the actual change.
- AgentDojo contributes 27 versioned injection-task updates or additions from 23 source groups. Not every record has a paired predecessor, and injection-task changes cannot all be interpreted as business-policy changes.
- SkillsBench combines four source tasks with four execution-condition changes each, including resources, output contracts, and dependency versions, yielding 16 specifications.
- Six cross-source specifications cover three pairwise combinations. Their components reuse existing sources rather than adding six independent source tasks.

Each experiment reports the executed subset of this inventory.

Submitted supplementary package. The package contains core update code, frozen records for the main controlled comparisons, and offline verification tools. It supports recomputation of the packaged results and testing of the core update operations. Historical archive checks and separate native reconstruction experiments are reported in Appendix I.

Earlier scoped reconstruction checkpoints. A 26-file workflow module supplies contracts, mutation recipes, a source-family-grouped development manifest, separate evaluation fixtures, a dependency checker, and a model-free execution entry point for four tool families and eight conditions. It pins the ToolSandbox revision and source hashes while excluding third-party raw data. A fresh macOS arm64 installation with Python 3.11.13 passes all 40 native reference checks and dependency checks. A 37-file extension adds two numerical-tool fixtures, a provider-free adapter, reconstruction instructions, and a license ledger. Extraction verifies all hashes and 60 reference checks (40 workflow and 20 numerical). The archive excludes credentials, provider runners, and third-party raw data.

Earlier source-module acceptance and exclusions. The scoped ToolSandbox module accepts 12 development conditions from eight source groups and five modules. Independent replay reproduces 120 native reference checks and 12 unknown-handling injections. Other candidates remain outside native acceptance: 18 of 20 BFCL cases pass call-contract checks without executing tools, while 24 StreamingQA cases support annotation reconstruction but lack verified factual transitions.

D.1 Partitions and uses of the reported batches

The table distinguishes the benchmark's split protocol from the actual uses of the reported batches. Admission/final-task separation alone does not establish source-disjoint final evaluation.

Batch	Actual partition and evidence use
19-condition reliability checks	Development reference, fault, and unknown-handling controls; checks execution and scoring.
Matched patches and continual maintenance	Four 16-step development streams; public checks and repair are visible; descriptive comparisons of correlated states.
Finite interface-dependency ablations	17 constructed candidates per fixture; two admission and two final workflows are separate; development component tests.
Retail and airline workflow pilots	Specified admission and final task/user groups are disjoint; development diagnostics, not source-disjoint full-controller evaluation.
Dialogue and retention diagnostics	Tasks, models, and judgment scope recorded per batch; post-hoc audits and original outcomes retained separately.

E Additional related work

This section expands the positioning in the main text: continual learning and editing motivate adaptation and retention, skill improvement informs candidate generation, and agent benchmarks supply execution settings. Further connections concern memory, repair, and behavioral testing. The literature scope and the implemented experimental comparisons are reported separately.

Continual learning, editing, and temporal knowledge. Learning without Forgetting, EWC, GEM, and episodic replay study acquisition of new capabilities while retaining old ones (Li & Hoiem, 2016; Kirkpatrick et al., 2017; Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2019). MEND, ROME, MEMIT, and SERAC investigate targeted edits and their scope (Mitchell et al., 2022a; Meng et al., 2022; 2023; Mitchell et al., 2022b). TENT adapts models at test time (Wang et al., 2021); retrieval-augmented generation and StreamingQA address external knowledge and temporal adaptation (Lewis et al., 2020; Liska et al., 2022). Our setting shares their adaptation–retention objective, but updates externally stored Skill artifacts and evaluates tool behavior, execution state, and commit decisions. Our contribution is a joint protocol for evaluating and controlling updates to external Skill artifacts.

Skill acquisition and candidate improvement. ReAct interleaves reasoning and action, Reflexion uses verbal feedback, and Voyager accumulates reusable skills (Yao et al., 2023; Shinn et al., 2023; Wang et al., 2023). SkillFlow directly studies lifelong skill discovery and evolution (Zhang et al., 2026b); therefore skill evolution is an existing research problem. Toolformer, Gorilla, and ToolLLM develop tool-use capabilities (Schick et al., 2023; Patil et al., 2024; Qin et al., 2024). Self-Refine revises outputs using self-feedback (Madaan et al., 2023); APE, OPRO, DSPy, and TextGrad optimize instructions or compound language-model systems (Zhou et al., 2023; Yang et al., 2024a; Khattab et al., 2024; Yuksekgonul et al., 2025). These works motivate stronger updater baselines. Our comparison asks whether candidate improvements survive changed-task execution and protected-task replay under matched information and budgets; citing these methods does not imply that all have already been implemented or evaluated here.

Agent and tool-use benchmarks. BFCL, τ-bench, τ^2-Bench, ToolSandbox, and AgentDojo provide tool-call, conversational, stateful, or adversarial evaluation (Patil et al., 2025; Yao et al., 2025; Barres et al., 2025; Lu et al., 2025; Debenedetti et al., 2024). AgentBench and GAIA broaden task

coverage (Liu et al., 2024; Mialon et al., 2024). ALFWorld and WebShop supply interactive environments (Shridhar et al., 2021; Yao et al., 2022); Mind2Web, WebArena, VisualWebArena, and WorkArena evaluate web interaction (Deng et al., 2023; Zhou et al., 2024; Koh et al., 2024; Drouin et al., 2024). OSWorld and SWE-bench emphasize computer and software execution (Xie et al., 2024; Jimenez et al., 2024). These are related evaluation settings, not all sources used in our experiments. We extend selected sources with paired updates, protected tasks, and commit/rollback records; scores remain lifecycle-derived rather than official leaderboard results.

Persistent memory, repair, and risk evaluation. MemGPT manages hierarchical memory for long contexts and multi-session interaction (Packer et al., 2023); we additionally ask whether modified persistent artifacts still satisfy task requirements. SWE-agent studies agent-computer interfaces, while Agentless separates software repair into localization, repair, and patch validation (Yang et al., 2024b; Xia et al., 2025), offering related designs for scoped edits and execution checks. ToolEmu emulates tool execution and evaluates risks with language models (Ruan et al., 2024). These settings also motivate distinguishing simulated judgments, call-contract checks, and actual execution when reporting evidence.

Multi-agent collaboration. AutoGen supports conversational coordination, MetaGPT structures role-based workflows, and multiagent debate exchanges competing responses (Wu et al., 2024; Hong et al., 2024; Du et al., 2024). These mechanisms can supply candidate diversity. CoordinateClosure separates proposal and repair from executable admission: agreement among agents does not substitute for a successful replay. Matched call budgets help distinguish update mechanisms from extra inference.

Behavioral testing and trustworthy reporting. CheckList motivates evaluating specified behaviors beyond aggregate accuracy (Ribeiro et al., 2020). Datasheets and Model Cards motivate explicit data provenance and intended-use boundaries (Gebru et al., 2021; Mitchell et al., 2019). Recommendations on significance testing and uncertainty in few-run evaluation motivate paired comparisons and uncertainty reporting (Dror et al., 2018; Agarwal et al., 2021). For lifecycle evaluation, independent source events should be distinguished from correlated steps, and operational failures kept visible. A statistical non-inferiority claim requires a prespecified margin; a nonsignificant difference alone does not establish preservation. We do not claim such a test here.

F Execution settings and a worked development case

Execution and provenance. The following settings apply to the saved τ^2 retail exchange batch. The actor's requested model identifier is `qwen-plus`, with trial seed 11, temperature 0.2, at most 1,024 output tokens per call, and 40 steps per trajectory. API timeout is 90 seconds, configured API retries are zero, task timeout is 600 seconds, and concurrency is four. The simulator uses lifecycle-v5. Frozen records bind tasks, candidates, and executed policies by hash. Four source tasks across three generated policies and one stale control yield 16 trajectories. A trial seed does not guarantee deterministic hosted API responses, and a requested alias is not an immutable server-side weight version.

The frozen v1b configurations request `qwen3.7-plus-2026-05-26` and `glm-5.2`. Both use temperature zero, a 2,048-token output cap, a 240-second timeout, a retry setting of two, one-second backoff, and concurrency eight. Each non-rollback step makes one model call per arm; rollback steps make none. Provider seeds are uncontrolled, so these are not multi-seed stability results. This budget generates edits, unlike the native actor's 1,024-token task-execution budget.

Input and transition. Source task 0 requests two exchanges from a delivered order: a full-size keyboard with clicky switches, preferring RGB but accepting no backlight if that combination is unavailable, and a thermostat compatible with Google Home. The bulletin prohibits gift-card settlement of exchange price differences. Because this task uses a credit card, it is a protected task expected to remain valid, not a changed task specifically testing a newly forbidden action. The updater receives public policy evidence; the original user goal and reference actions are preserved. The actual patch replaces the following settlement clause while retaining the same-product restriction:

> Before: The user must provide a payment method to pay or receive refund of the price difference. If the user provides a gift card, it must have enough balance to cover the price difference.
>
> After: The user must provide a non-gift-card payment method to pay or receive refund of the price difference. Gift-card settlement is no longer permitted for exchanges.

The candidate assigns this edit to `policy`. This case illustrates policy-edit localization.

Recorded execution. Indices below are zero-based saved-message indices. These are selected evidence excerpts, not a regenerated demonstration.

1. Messages 2–9 identify the account by name and ZIP, retrieve the order, and inspect product variants.
2. Message 10 identifies the permitted no-backlight fallback, presents the thermostat replacement, computes refunds of $3.17 and $13.46 (total $16.63), and requests confirmation.
3. Message 11 confirms only these two exchanges and the $16.63 refund to the original credit card.
4. Message 12 calls `exchange_delivered_order_items`, mapping item IDs 1151293680 and 4983901480 to 7706410293 and 7747408585. The payment method is `credit_card_9513926`.
5. Messages 13–14 contain the returned order record and completion report. Database and necessary-action checks pass; no gift-card exchange call is observed.

Scoring and limits. The database scorer checks final state; the action guard checks required writes and arguments; the violation monitor checks exchange payment parameters. A separate post-hoc audit gives the same method-blind trajectory to `qwen-plus` and `glm-5.2`, with temperature zero, an 8,192-token output limit, thinking disabled, and concurrency two. Packets contain the user scenario, updated policy, original natural-language assertions, and two common added requirements: fulfill applicable requests and obey policy. Both judges mark this case's requirements and simulator fidelity as passing, producing a joint diagnostic pass. This is not human-calibrated accuracy or an exhaustive factual verification of every assistant statement.

This worked case covers input, patching, execution, and scoring. It has no separate admission tasks or complete gated arm; independent final evaluation therefore requires a different batch with frozen admission decisions.

G Execution and measurement diagnostics

Public-task diagnostics motivate the joint endpoint. In the updated-policy arm of the τ^2 development cohort, 28/44 observable trajectories contain no explicit violation, while only 10/44 complete the native task. Eighteen therefore avoid observed violations without recovering the task. In SkillFlow, both compared program pipelines pass 8/9 native cases; the unresolved case produces no required output files despite a zero return code. These diagnostics motivate separate behavioral and delivery checks.

Repair of rejected first-pass candidates. On 45 rejected development candidates frozen after the first-pass runs, obligation-grounded learned repair recovers 24/45 (53.3%); deterministic repair using known public transition rules recovers 45/45. The latter is a rule-based upper bound. The selected candidates are correlated and are counted separately from independent events.

A subsequent end-to-end development run separates two admission tasks from two final tasks and records candidate, executed-policy, and gate hashes. All ten final trajectories contain the required exchange execution evidence; seven pass the joint diagnostic checks and three remain unresolved. The gate retains the parent policy. This establishes an executable audit path, while method effectiveness still requires broader matched comparisons (Appendix G.5).

G.1 Adapter readiness and treatment scope

ToolSandbox's 300/300 constructor/evaluator audit checks construction without model calls. Adapter-level checks are reported separately from generated-update effectiveness.

TimeWarp WikiTest produced 24 Qwen/AgentLab run records with a 40-step cap: all 24 were truncated and none completed the task within budget. These are unsuccessful generic-agent baseline runs, not a CoordinateClosure comparison. Truncation alone does not distinguish inadequate budget from navigation, tool, or reasoning failures. AgentDojo currently has 20 native-bound events and seven excluded events among 27 checks; SkillsBench's citation-check smoke verifies adapter readiness only.

G.2 Public checks and repair for small workflows

We construct small executable workflows from four ToolSandbox function families: reminders, contacts, messaging, and network settings. Each family has a single-change and a coupled-change condition, yielding eight conditions involving arguments and return structures, time units, or resource prerequisites. These are synthetic controlled changes, not historical upstream migrations. Each condition is proposed by `qwen-plus` and `glm-5.2` (temperature zero, requested seed 11, 2,048 output-token limit), giving 16 condition–model units per format. All families and variants are development data. Admission and final replay use distinct generated task entities, not source-family-held-out data. Execution invokes native tool functions rather than a full conversational agent.

Initial format	First-pass target	After common checks/repair/gate	Total tokens
Whole rewrite	16/16	16/16	11,008
Untyped patch	7/16	16/16	19,189
Coordinate patch	6/16	16/16	20,977

Table 4: Small-workflow development diagnostic. All groups use the same public checker and at most one generated repair after public rejection, followed by admission and final replay. Tokens include reported input/output usage for initial generation and repair.

The public checker verifies tool arguments, unit conversions, output references, resource prerequisites, and protected scope. It uses hand-declared rules and a fixed dependency structure specialized to these families, rather than discovering complete semantic dependencies for new tasks; exchanging two same-type inputs can still pass static checks. Repair receives only the public contract, candidate, and public-check errors, never final-task outcomes. With the common checker, at most one repair, and replay gate, all groups reach 16/16 on both target and protected tasks. Protection covers only an unchanged contact query, not broad capability retention. Ten repair requests were rejected before generation because of JSON-output configuration and each was retried once in a separately recorded recovery; original requests and gates remain preserved. The ledger contains 77 requests, 67 generated responses, and 51,174 reported tokens. Ten rejected requests have no usage records and are not treated as known zero-cost requests.

Common public checks and repair remove the final-success difference in these small workflows. Coordinate patches offer no success or token advantage here. The checker was refined during development; this diagnostic covers smaller artifacts and different cost components from the main maintenance study.

G.3 Additional numerical-tool migrations

Two previously unused source-scenario-derived numerical subtasks test timestamp differences and geographic distances. Synthetic API migrations change endpoint names, argument units or nesting, and return structures; upstream native functions perform the computation. Holiday and location retrieval are replaced by explicit public inputs. These are two events from one ToolSandbox utilities module, not complete conversations or independent source families. Contracts, scoring, and source hashes are frozen before generation. Twenty reference checks establish pre-migration validity, post-migration reference success, stale-target failure, and unchanged unit-conversion behavior.

Two backbones (`qwen-plus`, `glm-5.2`) generate three formats for both events: 12 candidates with temperature zero, requested seed 11, and a 2,048-token output limit. A fourth arm reuses the identical coordinate candidate with replay admission and no repair. Admission and final tasks use distinct numerical inputs. All eight admission and 32 final execution receipts are present, with no binding errors or unknown final outcomes. Whole rewriting, untyped patches, coordinate patches, and coordinate patches with admission pass respectively 3/4, 2/4, 3/4, and 3/4 target tasks; all four pass 4/4 protected unit-conversion tasks. Each denominator is two events times two models. Generated tokens total 2,270, 1,914, and 2,009 for the three proposal formats; admission reuses its candidate. An incorrect return-value selector causes all three GLM distance candidates to fail; rejection retains a stale endpoint and does not restore the target. Coordinate edits match rewriting in this small development comparison, with retention limited to the protected unit-conversion task.

Public-schema repair extension. A subsequent development version reuses the 12 candidates above and adds 12 fresh candidates for two date/time conversions from the same utilities module. All three formats receive the same public-schema checker and at most one repair; only the fourth arm additionally gates execution of the identical typed candidate. Forty reference checks pass before generation. The 16 admission slots comprise 12 executions and four public-check rejections; all 64 final slots are executed, yielding 46 passes and 18 failures. On the two reused events, target/protected passes are respectively 3/4 and 4/4 for rewriting, 2/4 and 3/4 for untyped edits, 3/4 and 4/4 for typed edits, and 3/4 and 4/4 with admission. On the two fresh events these counts are 2/4 and 2/4, 3/4 and 3/4, 2/4 and 2/4, and 2/4 and 4/4. New generation and repair use 16,635 reported tokens; reused generation adds 6,193 historical tokens, counted once. Some responses return a public-packet wrapper instead of the required artifact and remain failures under the frozen parser. Admission preserves the parent for invalid candidates, protecting the unchanged task without restoring the changed target. One reused untyped candidate loses executability after repair. This version was designed after inspecting earlier development results; the schema checker is not a complete semantic dependency compiler, and the fresh events are not independent source-family tests.

G.4 Policy diagnostics

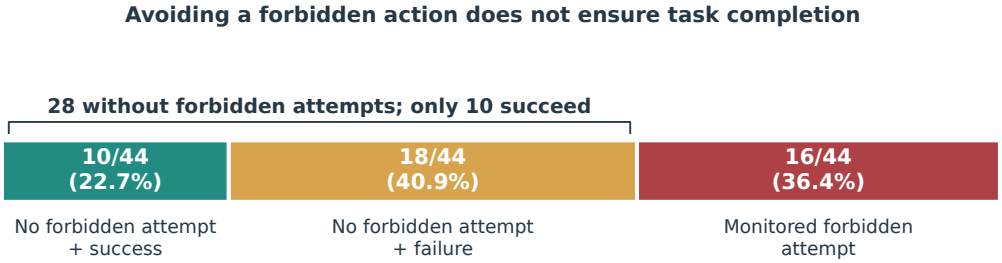


Figure 8: Behavior and task completion in the frozen policy-state diagnostic. Among 44 observable updated changed-task trajectories, 28 contain no monitored forbidden attempt, but only 10 also complete the task. Four of 48 pairs are unobservable and excluded from this diagnostic breakdown. This is not a CoordinateClosure-versus-baseline comparison; the monitor covers specified forbidden actions only.

In the frozen τ^2 v4b development cohort, 240 cells span four coupled events, 20 source tasks, two policy states, three trial seeds, and two backbones. Five cells remain operationally unknown. Of 48 changed-state pairs, 44 are observable on both sides. The paired effect on explicit violation attempts is -0.545, with 27 wins, three losses, and 14 ties; the source-cluster bootstrap interval is $[-0.813, -0.198]$. Changed-task native success has a paired effect of $+0.205$ with interval $[0, 0.438]$, and protected retention has effect -0.014 with interval $[-0.097, 0.069]$. This stale-versus-updated-policy comparison is development evidence, not a four-method update comparison.

An independent 32-cell accepted-policy diagnostic yields 31 completed trajectories and one operational unknown. The scorer omits required natural-language assertions and three changed events have no expected database actions. Thus component passes cannot establish full task recovery.

Qwen3.8-Max and GLM-5.2 independently review 52 assertions in 25 packets, agreeing on 50 (31 passes and 19 failures), with two disputes. The 96.15% agreement is not calibrated accuracy. An eight-trajectory exchange smoke remains affected by premature simulator termination: a database pass can coexist with no requested action. These runs are excluded from method-effectiveness claims.

G.5 Failure analysis and adaptive verification

Protected-task expansion

We select seven original-payment return tasks whose source users do not overlap with the admission or final tasks used in the preceding return and exchange diagnostics. Original goals and reference actions remain unchanged. Selection follows source-task ID order after schema and reference-replay validation; one reference-replay failure is excluded and recorded before method execution. Two actors execute three generated-policy arms and a stale-policy control with one trial seed, yielding 56 planned cells and 55 saved trajectories. One GLM untyped-patch execution remains operationally unknown. Typed-versus-stale pairs yield three wins, zero losses, and four ties for Qwen, and seven ties for GLM. Typed and untyped generators produce identical policy text on this event for both actors, so differences between their executions cannot identify a coordinate-representation effect. The small, single-event development sample supports a limited database-retention observation; broader event coverage and statistical non-inferiority remain unverified.

Table 5: Protected return tasks: native database-check passes over seven planned tasks per cell. U denotes an operationally unknown result, retained in the denominator. These are component-level development results, not full-task success rates.

Actor	Stale	Whole rewrite	Untyped patch	Typed patch
Qwen-plus	3/7	5/7	6/7	6/7
GLM-5.2	7/7	7/7	6/7 + 1 U	7/7

Post-hoc semantic audit. Two judges assessed all 55 saved trajectories; one planned cell had no trajectory. Of 110 initial judgments, three had invalid structure and received one retry each, without retrying valid disagreements or unfavorable judgments. Combining the effective judgments with database and necessary-action checks yields 39 pass, 6 fail, and 11 unknown outcomes over 56 planned cells. Typed-versus-stale pairs give Qwen one win, zero losses, four ties, and two unknown pairs; GLM has zero wins, one loss, four ties, and two unknown pairs. In the GLM loss, the requested savings were reported as $38.32 instead of $41.64 despite a successful database action. Thus, database retention alone does not establish semantic retention. These model-assisted, post-hoc diagnostics do not establish representation effects or calibrated evaluation accuracy.

Airline shadow evaluation. A separate fixed-format evaluation produced 46 structurally valid judgments for 23 saved trajectories from 24 planned cells, without rerunning agents or changing the original admission decisions. The 20 final-evaluation cells yield 7 pass, 9 fail, and 4 unknown joint outcomes; one missing admission trajectory remains unknown in the full ledger. Both original gates retain their parent policy. Structural validity measures judgment-format completeness, not correctness; judge disagreement and uncertain simulator fidelity remain unresolved. This single-event development diagnostic is not a held-out method comparison.

Additional protected tasks and environment correction. Three additional source tasks, disjoint from the seven above, yield 24 complete trajectories across four policy states and two actors: 12 pass, four fail, and eight remain unknown under joint checks. One task exposes an upstream multi-item tool defect, so these original-environment results are not clean method-effectiveness evidence. A versioned correction is applied uniformly to all eight actor–policy combinations for that task; seven produce complete trajectories, with four passes, two failures, and two unknowns over the eight planned runs. Required-action checks accept equivalent item orderings while preserving item pairing, payment parameters, and multiplicity. Original receipts and outcomes are retained. The two task sets cover ten protected tasks for one policy-change event; the repaired batch repeats one task and

is reported separately, not pooled as additional independent evidence. Source-database verification also confirms the earlier GLM savings regression: the cheapest valid replacements save $14.18 and $27.46, totaling $41.64, rather than the reported $38.32.

Retention diagnostic on additional source tasks. For the same policy event, source-ID selection adds six previously unused protected tasks (20, 33, 36, 40, 41, 43), preserving complete user goals and reusing fixed policy candidates. Two actors and four policy states run in the version-corrected environment, yielding 48 planned cells. Joint database, necessary-action, and two-judge semantic checks yield 23 pass, 6 fail, and 19 unknown outcomes; disagreements and uncertain user-simulator fidelity remain unknown. This denominator pools stale, whole-rewrite, untyped-local, and coordinate-update states and is not any method's success rate. The additional tasks broaden protected-task coverage for one event without adding independent change events. This development diagnostic is reported separately from the preceding batches and does not establish non-degradation.

Operational recovery and scoring review. The original 48-cell result remains unchanged. Six GLM trajectories interrupted by empty-assistant-message validation were each retried once under a fixed selection rule, retaining tasks, candidates, seeds, and scoring protocol. All six completed, with two-judge outcomes of three passes, two failures, and one unknown. Combining them with the 42 unchanged outcomes gives a secondary analysis of 26 passes, eight failures, and 14 unknowns. Recovery execution records an additional 739,916 input/output tokens; interrupted original prefixes and subsequent judgments incur separate costs. This secondary view does not replace the original 23/6/19 record.

A uniform offline evidence audit attributes the 14 historical unknown labels to 12 judge disagreements and two assertion-coverage errors. Twelve of these trajectories already fail database or necessary-action checks. For the conjunctive endpoint, any confirmed required-component failure takes precedence over unresolved judgments; unknown applies only when no component is confirmed to fail and at least one remains unresolved. The existing component-dominates reaggregation therefore yields 26 pass, 20 fail, and two unknown outcomes on the same 48 recovery-sensitivity trajectories. The historical judge-first counts (26/8/14) and all original judgments remain archived; no trajectory is rerun and the success numerator is unchanged. Localized judge errors include misreading available products and missing simulator departures from the source goal; these diagnoses do not justify selective reassignment to pass.

Synthetic recoverable policy event. A separately frozen, self-authored refund event executes a legal original-payment refund using actual τ^2 RetailTools and its database. It is counted separately from the 12 ToolSandbox development conditions and does not alter the original unsatisfiable gift-card task. Admission and final checks use distinct synthetic entities, with reference controls verified before generation. Two models (Qwen-plus and GLM-5.2), two repeats (requested seeds 11 and 29), and three generation formats produce 12 candidates; the fourth arm reuses the typed candidate with replay admission and no repair. All 12 outputs materialize to the same updated artifact. All eight admission and 32 final executions pass; each method–model group has target and protected scores of 2/2. Offline replay exactly matches all 40 execution receipts. Generation uses 800, 828, and 884 tokens for the three formats, totaling 2,512. This single-mechanism diagnostic verifies a learned-update-to-execution path, with identical artifacts and tied outcomes.

Task-disjoint end-to-end admission diagnostic

We connect candidate generation, native execution, independent admission replay, and final evaluation on one retail policy event. Qwen-plus executes tasks; Qwen-plus and GLM-5.2 independently assess natural-language assertions and user-simulator fidelity. Admission source tasks (58, 94) are disjoint from final source tasks (45, 75). This is a single-trial development diagnostic. Task45 permits a legal settlement alternative when a gift card cannot be used, and its original reference action already uses PayPal. The stale policy can therefore succeed; this construction is not assumed to force a stale-policy failure.

Each method executes one affected and one protected task. Joint checks require native reward components, natural-language assertions, user fidelity, and required-action evidence. Action evidence joins an assistant tool call to a subsequent successful tool result by call ID and checks the action

name and full arguments; promises cannot substitute for execution. This is a necessary condition, not a replacement for semantic or permission checks.

Table 6: End-to-end development diagnostic on recoverable tasks. Each cell contains one trajectory; unresolved outcomes stay in the denominator and do not rank methods.

Method	Affected task	Protected task
Stale policy	Pass	Pass
Whole-policy rewrite	Pass	Pass
Untyped local patch	Pass	Unknown
Typed coordinate patch	Disputed	Unknown
Typed patch + replay admission	Pass (parent)	Pass (parent)

All ten final trajectories contain required-action evidence: seven pass joint checks, one has an assertion disagreement, and two have incomplete scoring schemas. Both admission trajectories lack required execution and have incomplete judgments, so the gate retains the parent. Final passes in the gated arm therefore concern the parent. Typed and untyped candidates contain identical policy text; their dialogue differences cannot identify a representation effect.

Four synthetic known-label scorer controls were also assessed by both models. Of eight calls, seven returned schema-valid opinions and six correctly classified both task outcome and user fidelity. Qwen falsely reported success when the user stopped before execution; GLM returned an incomplete response on that case. This motivated the programmatic action check. These small controls do not estimate scoring accuracy on natural trajectories, and the diagnostics are excluded from the main effectiveness table.

Fixed-budget probe diagnostic

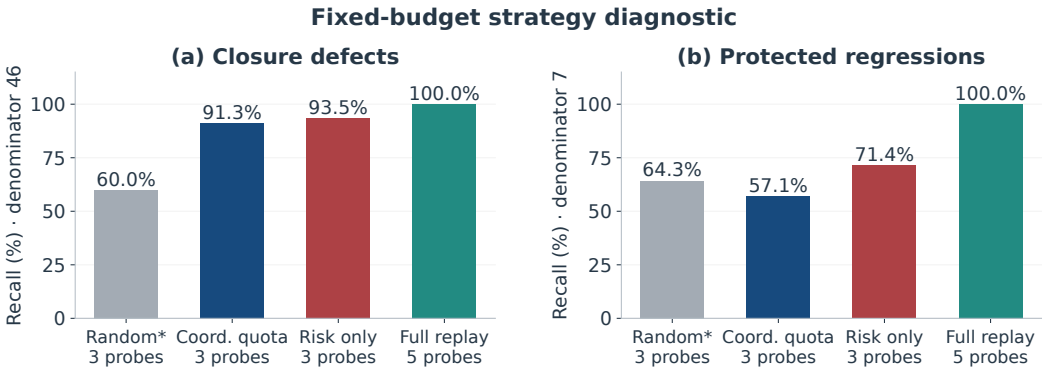


Figure 9: Fixed-budget probe-selection diagnostics on 24 defective candidates. Three-probe strategies are compared with full five-probe replay. Defect recall uses 46 defects; protected-regression recall uses seven candidates. Random selection is an exact expectation. With no beneficial candidates, the pool cannot measure beneficial-update admission. This is not a budget sensitivity sweep.

On a post-hoc pool of 24 defective candidates, each with two target and three protected probes, uniform selection of three probes gives 60.0% expected defect recall. Risk-only selection achieves 93.5%, compared with 91.3% for a fixed two-target/one-protected coordinate quota. Protected-regression candidate recall is 71.4% versus 57.1%. This exposes a limitation of fixed quotas and motivates a risk–coverage selector, whose prospective benefit remains unverified. Since every candidate is defective, this pool cannot measure acceptance of beneficial updates.

In StreamingQA-derived case-disjoint development confirmation, whole/scoped annotation-state checks pass in 23/24 versus 24/24 cases for Qwen and 23/24 versus 23/24 for GLM; scoped updates use 26.6% and 20.7% fewer tokens in the respective reported comparisons. These checks are

not native QA execution or verified factual evolution, and the small number of discordant outcomes does not establish an accuracy improvement. SkillFlow's 8/9 versus 8/9 outcome similarly provides a delivery-failure case, not evidence of improved recovery.

H Finite interface-dependency component ablation

The same four-arm ablation was executed on two existing ToolSandbox calculation fixtures, days until a holiday and geographic distance, with 17 candidates each. In each fixture, schema / dependency / replay / dependency-plus-replay yield target passes 2/2/1/1, protected passes 9/15/17/17, and joint passes 1/1/1/1, all out of 17. Jointly invalid candidates rejected number 0/12/15/15, with no rejection of the single beneficial candidate. Replay admission slots fall from 32 to 8; available-tool call attempts fall from 24 to 6, and successful calls from 20 to 6. Across both fixtures, 80 admission and 272 final workflow slots were actually executed with no model calls. These repeat constructed finite-contract tests on development fixtures, not independent natural events. Rejection retains the old artifact and therefore need not restore changed-task performance.

The checker verifies the public endpoint and two argument-name bindings. Each configuration retains the parent for a missing candidate. All 17 candidates remain in the denominator, and counts are compared within this versioned run. Dependency filtering reduces candidate admission tool dispatches from 65 to 17; separately recorded reference dispatches decrease from 144 to 36. Final candidate dispatches remain 69 and final reference dispatches remain 85 in both replay configurations. The 75% reduction concerns admission workflows, not total cost, latency, or tokens. Replay rejects one candidate that passes the target but fails protection, explaining the target-pass decrease from two to one. The injected suite does not estimate natural model-error frequencies; zero false rejections on one valid candidate is not a population estimate.

Table 7: Exhaustive candidates for one development interface event. Rejections are out of 15 jointly invalid candidates; target, protection and joint passes are out of 17. The sole valid candidate is never wrongly rejected.

Checks	Reject	Target	Protect	Joint	Replays
Schema	0	2	9	1	0
Schema + dependency	12	2	15	1	0
Schema + replay	15	1	17	1	32
Schema + dependency + replay	15	1	17	1	8

I Fresh change groups and dialogue-format diagnostics

New groups within the same source. The core release snapshot contains 19 development conditions; four additional IANA change groups are reported separately, not pooled into 23 independent source events. Two models and three generation formats produce 24 requests; the replay arm reuses coordinate candidates. Under a uniformly frozen metadata-compatibility rule, all 24 admission and 128 final conversions pass, with ties across methods. Strict-format results remain separate: 116 passes and 12 unknowns. A post-hoc semantic-diff audit finds exactly one affected zone-selector change in each of the 24 candidates and no unrelated edits. Generation costs vary by model and format, without a universal cost advantage.

Complete dialogues and observability revision. Local goal audits yield eight passes and four failures in the original 12-dialogue batch, versus seven passes, four failures and one unknown in the separate 12-dialogue error-budget sensitivity batch. These are not pooled into one success rate. The original latest-order task lacks an observable timestamp or ordering contract. We therefore construct a distinct revision with publicly queryable synthetic timestamps rather than rescore the old task. Both models complete authentication, timestamp and order lookup, and the correct payment answer before and after the policy change, yielding four goal-complete dialogues. This supports goal completion on one revised development task, not broad capability non-inferiority.

Execution-message normalization. All four revised-task traces contain raw messages mixing text with tool calls, contrary to the literal output protocol. The runner routes those messages to the tool environment without also delivering their text to the user. A new adapter clears only that text in the execution copy, preserving raw responses and model history, tool arguments and order; unsupported multiple calls are rejected rather than split. Offline native replay of all four traces, with 24 message projections, reproduces tool outputs, user-visible dialogue and final database states exactly. Normalized-protocol checks pass for all four traces and nine tests pass. Original literal failures remain reported. This is not a new online model run or a full-policy success rate.

Shared-artifact retail pilot. A separately frozen retail interface event uses admission tasks 60/62 and final tasks 79/90 from a different user group. Local-record exposure screening found no prior execution of the final group before this pilot; this is not a new upstream source. Two models generate three formats, with a fourth arm reusing the coordinate candidate for admission. Four provider errors among six initial requests are retained; a separately recorded connection-only recovery obtains the four missing responses without changing prompts or scoring. All six candidates materialize to the same artifact, and all 16 target/protected reference-workflow checks pass. Four complete Qwen protected dialogues subsequently complete the authorized cancellation; one misstates the price difference, so goal completion does not imply error-free communication. Four additional affected-task dialogues yield two original-goal passes and two failures. In the two failures, the simulated user invents a color inconsistent with the source task and database. These failures remain recorded and are flagged for simulator-fidelity failure, not attributed to updater differences. All four arms use identical artifacts. These evidence-indexed, non-blind assistant audits are diagnostic rather than calibrated ground truth; raw-message and normalized-transport compliance remain separate.

Airline interface pilot. One additional controlled baggage-interface event uses admission tasks 17/6 and final tasks 12/5 from two different final user groups, with local-record exposure checks and native reference/stale preflight tests. Three Qwen generation requests compare rewriting, untyped patches and coordinate patches; admission reuses the coordinate candidate. The latter two formats and the gated arm pass both final reference workflows, yielding six passes. Rewriting returns a normal model response but reverses the argument mapping and is rejected by the frozen materializer; its two execution outcomes remain unknown, not provider failures eligible for infrastructure recovery. This single-event format-conformance difference is not a reliable method ranking. Natural-language goal constraints are not evaluated by these reference workflows. Neither pilot increases the 19-condition core release count.

Repeated generation and contract sensitivity. On the already observed airline event, two models, four additional seeds, and three generation formats produce 24 requests; the gated arm reuses coordinate candidates. All eight generic-patch and eight coordinate-patch candidates materialize to the same artifact, whereas all eight whole-rewrite responses reverse the argument mapping. Of 64 final workflow slots, 48 pass and 16 remain unknown. These are correlated repetitions of one event, not additional independent events. A subsequent post-hoc sensitivity protocol, frozen before new calls, uniformly clarifies for all formats that mapping keys are old parameter names and values are new names. Six requests across two models, three formats, and another seed all materialize successfully; all 16 final four-arm slots pass, with identical candidates. Independent reconstruction without model calls matches candidates and final receipts in both batches. The clarified-contract result prevents interpreting the earlier difference as a coordinate-specific advantage. The batches use different seeds and are not a paired causal estimate of the prompt change. Generation/materialization errors must be distinguished from execution outcomes; this diagnostic shows no coordinate-patch token advantage.

Task-group-disjoint native workflow supplement. Four additional endpoint migrations use six final airline task/user groups, disjoint from five admission groups; local execution records contain no prior runs of these final groups. Twelve initial generations and eight public-feedback repairs consume 41,376 tokens. Under the frozen materialization contract, rewriting passes all four joint event checks; generic and coordinate patches each yield eight unknown final workflow outcomes, while the admission arm retains the parent, passing four protected workflows and failing four affected workflows. Both patch formats assign an entire argument-mapping object, whereas the materializer accepts only leaf assignments, a restriction omitted from the public format instructions. Independent

reconstruction matches all original results. A separate post-hoc analysis expands object assignments into equivalent leaf assignments without changing any values or calling a model; all four arms then pass all four joint event checks (32/32 final workflow cells). The apparent ranking therefore depends on an underspecified representation contract. The fourth arm uses public binding checks and admission, not the full controller; reference-workflow replay does not evaluate complete conversations. All raw failures, unknowns and repair costs are retained.

Additional reproduction checks. Fresh extraction of the historical evidence archive verified all 1,727 payload hashes. An empty-cache IANA rebuild compiles the six core and four additional conditions. Existing-environment reruns reproduce 216 ToolSandbox controls and nine synthetic-retail outcomes, including two expected unknowns. These reproduce existing conditions; clean-environment results follow below.

Isolated native retail reconstruction. A further reconstruction downloads the fixed upstream commit and installs 74 version-pinned dependencies in a fresh Python 3.12 virtual environment without copying the previous environment. Seven native controls and two expected unknowns for the same synthetic policy event reproduce the original full receipt hashes. Runtime networking is disabled and no model is called. The entrypoint requires Python 3.12; verification covers local reconstruction of this existing policy fixture.

Isolated native ToolSandbox reconstruction. The existing 12-condition subset is also reconstructed from the fixed official source in a fresh Python 3.11 environment with 102 version-pinned dependencies. All 216 native controls match the frozen expectations: 120 reference/stale controls, 48 injected protection faults detected, and 48 corresponding unchanged-target passes, with no operational unknowns. A separate audit verifies all new receipt hashes and the complete historical control population. Execution networking is disabled and no model is called. These repeat existing controls and do not add events or method-comparison trajectories.