GENERALITY ON DEMAND: PROGRESSIVE BEHAVIORAL CONSOLIDATION FOR WEB AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Web agents often invoke a model again for routines they have already solved, paying for fresh action selection on every repetition. We study whether verified experience can instead become executable behavior that still recovers from deployment drift. PBC (Progressive Behavioral Consolidation) compiles repeated successes into parameterized workflows. At a live failure it returns control to an agent, then persists a recovery only if the resulting patch passes replay, regression, and held-out validation. On Gmail, PBC achieves 0.998 success, comparable to the strongest baselines (0.986–0.992), while reducing agent dependence from 1.000 to 0.306. Its final artifact executes 0.80/0.80 of isolated and cumulative drift states without a model. On GitLab, dependence falls to 0.111 and tokens to 0.34M per stream, with 0.948 success versus 1.000 for the full-solver baselines. Main-table results average three independent draws. Among the gate's rejections, what blocks a recovery is preserving behavior the system has already verified, not the novelty of the fix. Verified recovery thus changes how later tasks execute: the agent handles unresolved cases, while the workflow serves behavior the system has learned to verify.

1 INTRODUCTION

A web agent that has completed a routine a hundred times still re-reads the page, re-interprets its state, and re-selects every action on the next instance. The behavior has become familiar. The execution has not. On a diagnostic of 300 mastered-task trials (Appendix D.6), the compiled workflow solved every trial at *zero* model tokens, while the full agent solved 53.3% of them and spent 88.8% of its prompt tokens re-serializing page state rather than producing new reasoning: generality stays active even when the behavior itself no longer needs it. Existing self-evolution research makes future execution better informed: past interaction is stored as memory recalled at inference (Zhao et al., 2024), synthesized into demonstrations used as in-context exemplars (Murty et al., 2024), abstracted into skills (Wang et al., 2023; Li et al., 2026a), synthesized into reusable workflows (Wang et al., 2025; Liu et al., 2026a), or used to evolve the harness around a frozen model (Du et al., 2026; Wei et al., 2026). The dominant objective is capability accumulation: experience improves what the agent can do, while execution still passes through the agent each repetition. We ask a complementary question: *can recurring web behavior progressively leave the agent loop?* The agent is not failing on these routines; it is paying for generality that the task no longer needs. Agents are most valuable at the frontier of what is unfamiliar, ambiguous, or structurally new; inside routines that have been repeatedly exercised and verified, the system should convert experience into directly executable behavior.

We call this process *progressive behavioral consolidation* (PBC); Figure 1 traces the full lifecycle. An unfamiliar routine is initially handled by the agent; after repeated verified successes the system compiles it into a parameterized executable workflow, and future instances inside the active, evidence-backed envelope execute deterministically with no model in the loop. When deployment drift breaks the workflow, control returns to the agent at the live failure state rather than the task start; if the agent recovers and the final task effect is externally verified, that recovery becomes evidence for updating the behavior.

An active *verified envelope* Ω authorizes deterministic execution for tested context–instance cells. At its boundary, the workflow runs first and hands a failure's live state to the agent; unfamiliar contexts go to the agent directly.

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.

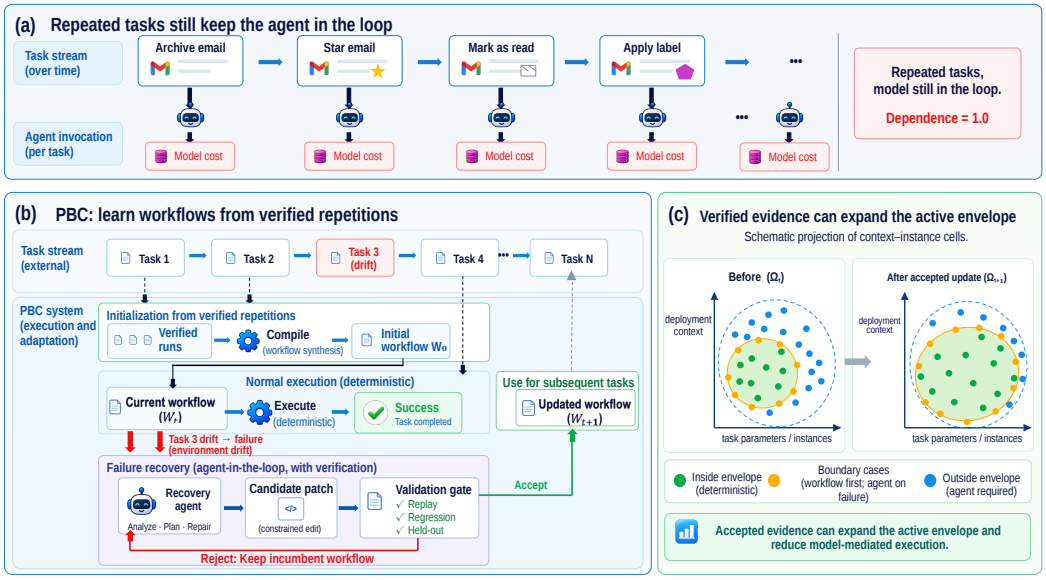


Figure 1: PBC compiles verified repetitions into a workflow, recovers from drift at the live failure state, and persists a patch only after validation. The active envelope Ω marks where the workflow can execute without a model.

A recovered patch enters the workflow only after failure replay, regression on mastered instances, and held-out validation. The artifact therefore grows through demonstrated behavior rather than unverified self-editing. That admission step is also what makes the effect measurable: a routine counts as learned here not when the agent can perform it again, but when its execution no longer requires reasoning about it. Success alone cannot separate those two states, which is why we report agent dependence and model-free coverage alongside task success.

Claim and evidence. Verified repetition can move familiar behavior from the agent loop into a persistent workflow. On Gmail, PBC reaches 0.998 success against 0.986–0.992 for the strongest baselines (paired differences at most $+0.012$, with intervals touching zero). Agent dependence falls from 1.000 to 0.306, and the final artifact replays $0.80/0.80$ of isolated and cumulative drift states without a model. On GitLab, dependence reaches 0.111 with one order of magnitude fewer tokens per stream; success is 0.948 against 1.000 for the two full-solver baselines. Every baseline we compare against holds dependence at 1.000, including the released end-to-end harness we run, which reaches 0.487 success at that dependence. Table 1 averages three independent draws per arm. The gate links these outcomes: a verified recovery can change later execution only after its patch passes the acceptance checks.

PBC combines an executable workflow, a verified envelope that controls where it runs, and a gate that tests whether recovery can become a lasting update. Together they distinguish solving a task from retaining its solution. Our cross-application results measure that distinction in execution; gate diagnostics show which recoveries actually become persistent behavior.

2 Related Work

Agent memory, skills, and workflow reuse. Language agents reuse experience as verbal reflection, demonstrations, skills, or workflow memory (Shinn et al., 2023; Zhao et al., 2024; Murty et al., 2024). Voyager stores executable code skills under an LLM controller (Wang et al., 2023); Agent Workflow Memory induces reusable web routines from trajectories (Wang et al., 2025); SGDR retrieves state-grounded text–code skills (Li et al., 2026a); ReUseIt synthesizes guarded workflows from agent attempts (Liu et al., 2026a); and SkillAxe and SkillRevise revise skills as structured packages using execution evidence, on benchmarks such as SkillsBench that score them under deterministic verifiers (Gautam et al., 2026; Liu et al., 2026b; Li et al., 2026b). These methods make past experience reusable. Our question is when a verified procedure can execute directly, with an agent re-engaged only when the workflow reaches an uncovered boundary.

Self-evolving agent harnesses. A parallel line modifies the runtime around a frozen model: ReCreate maps interaction experience into scaffold updates (Hao et al., 2026); HarnessFix localizes failed trajectories to responsible harness layers and performs scoped repair (Chen et al., 2026); AHE makes components, experience, and edit decisions observable, pairing each edit with a prediction checked against the next round (Lin et al., 2026); Living-Harness and Evo-Harness accumulate procedural harness state online (Du et al., 2026; Wei et al., 2026). These systems optimize the harness or the agent's capability; we optimize *which familiar tasks still need the agent*. The two axes are complementary—a stronger agent can serve as the recovery component inside our loop (Section 5.3).

Mechanism boundary. The distinction is not whether experience is persisted, but what persistence changes at the next execution. ReUseIt persists guarded workflows synthesized from interaction traces, yet under our matched protocol the model remains on the execution path at drift. Evo-Harness persists procedural harness state so that an agent selects actions more effectively. PBC instead persists an executable workflow together with the region in which it has been verified: the workflow selects actions inside that region, contract failure returns the live state to a recovery agent, and a recovery becomes lasting behavior only after replay, regression, and held-out validation. The envelope therefore decides where the model may exit; the gate decides which recovery may move that boundary.

Reliable workflow automation. Deterministic execution of web procedures is the province of recorded workflows, program synthesis, and test automation. Benchmarks such as MiniWoB, WebArena, WebArena-Infinity, and WorkArena provide executable tasks with environment feedback (Shi et al., 2017; Zhou et al., 2024; Zhou, 2026; Drouin et al., 2024); TimeWarp re-creates era-representative UI versions and evaluates robustness across them (Ishmam & Marino, 2026); τ-bench and ReliabilityBench emphasize repeated-run consistency beyond average success (Yao et al., 2024; Gupta, 2026). These suites score outcomes: whether a run succeeded, and whether success repeats across attempts. The regime behind the outcome is not measured, so a system that re-solves each repetition and one that has learned to execute it score identically; dependence and model-free coverage separate the two. Recorded scripts, however, neither recover from drift nor decide when their own update is safe. PBC studies how a behavior *matures* from repeated verified experience; because an accepted self-update persists across future repetitions, verification is the acceptance criterion for self-modification, not a post-hoc evaluation.

3 Problem Formulation

A task template τ generates parameterized instances $x \sim \mathcal{X}_\tau$; a deployment is a repeated stream $x_1, \ldots, x_T$ from that template, with drift that changes the environment along the way. A system may hold a procedural skill S_τ (reusable knowledge that an agent policy π interprets) and a workflow W_τ (an explicitly executable parameterized program). The two execution regimes are

$$a_{1:T} \sim \pi(\cdot \mid x, O(E), S_\tau) \qquad \text{vs.} \qquad a_{1:T} = W_\tau(x, O_{W_\tau}(E)), \tag{1}$$

where $O(E)$ is the broad observation a general agent consumes and $O_{W_\tau}(E)$ is the narrow state the workflow's selectors and verifier query.

Agent dependence is the fraction of instances on which model-based action selection is required, $D(N) = \frac{1}{N}\sum_i \mathbb{1}[\text{instance } i \text{ requires model-based action selection}]$, so a fully agent-mediated method has $D(N) = 1$ no matter how capable it becomes (Appendix D.6). The *verified behavior envelope* Ω_t is the active set of (context, instance) cells for which the system holds evidence that its current artifact executes deterministically and correctly. A deployment method for repeated tasks should therefore achieve

$$\text{Success} \uparrow, \qquad \text{Agent Dependence} \downarrow, \qquad \text{Verified Coverage} \uparrow \tag{2}$$

without unacceptable regression. Capability improvement (higher success at $D{=}1$) and behavioral consolidation (lower D with broader evidence-backed Ω_t) are different objectives, and the experiments below measure them separately. The boundary this paper studies is *compilability*: whether a recovery that solved a drifted instance can be turned into an edit that preserves the artifact's verified behavior, and therefore whether the instance changes anything about later execution.

4 PROGRESSIVE BEHAVIORAL CONSOLIDATION

4.1 OVERVIEW

An executable workflow handles familiar tasks efficiently, but deployment drift can break its assumptions. PBC uses the workflow for verified routine execution and invokes an agent when adaptation is needed:

$$\text{execution: contracts}(\Omega) \rightarrow \text{workflow} \rightarrow \text{verified success},$$
$$\text{consolidation: contract failure} \rightarrow \text{agent recovery} \rightarrow \text{local patch} \rightarrow \text{verify}.$$

The loop preserves verified behavior: a patch enters the workflow only after a three-check persistence gate, and the active envelope Ω changes through verified evidence (Figure 1). In our experiments, three verified successes across at least two parameter instances seed the initial workflow. Deployment experience then determines which adaptations become persistent.

4.2 BEHAVIORAL REPRESENTATION

A consolidated behavior is a tuple

$$B = (C,\ W,\ V,\ \Omega,\ H), \tag{3}$$

where C is the task contract, W the parameterized executable workflow, V the authoritative external verifier, and H the append-only verification history. At time t,

$$\Omega_t = \{(c, x) : E_t(c, x; W_t) = 1\} \subseteq \text{cells}(H_t) \tag{4}$$

where $E_t(c, x; W_t) = 1$ means that W_t holds current gate-verified evidence on cell (c, x). Context identity is the benchmark's explicitly versioned drift configuration (`pristine`, `drift:` a, or cumulative `drift:` a+b), and x is a probed instance. History and active set move differently. H_t is append-only, so it is monotone by construction: every verification event is recorded and none is removed. $\Omega_t \subseteq \text{cells}(H_t)$ is the subset of that history currently authorized for deterministic execution, and it is not monotone: it *expands* when the gate admits evidence for a cell that was not previously authorized, *invalidates* a cell when the context that qualified it changes or its evidence stops applying, and *reactivates* the cell when re-qualification succeeds. Invalidation withdraws execution rights only; the record stays in H_t, which permits reactivation without re-deriving the evidence.

The active envelope is both a routing certificate and an evaluation object. A covered cell authorizes direct workflow execution; a boundary cell runs workflow-first and transfers the live state to the agent on contract failure; an unrepresented context remains agent-first. Routing therefore follows evidence attached to the artifact, not a model's confidence about familiarity. For evaluation, we reconstruct each drift state and replay the final artifact without a model, producing isolated coverage (one drift at a time) and cumulative coverage (the stacked context) under a common protocol. Thus Ω_t controls current direct execution, while replay measures what the final artifact can execute.

4.3 FAST LOCAL CONSOLIDATION

Fast consolidation turns one verified recovery into a constrained, persistent edit, through three gates.

Gate 1: verified recovery. A precondition, action, postcondition, or verifier failure preserves the live browser state and hands control to the recovery agent, which continues from that state—including completed side effects—rather than replaying the successful prefix. At most k attempts are made; a trajectory becomes eligible for compilation only when its final state passes the external verifier. Appendix A.4 specifies the uniform takeover rule used by every workflow-based arm.

Preserving the live state matters because recovery is not a second full solution. It identifies the smallest behavioral delta between the incumbent artifact and a verified completion: a changed selector, an inserted menu step, a dialog handler, or a revised postcondition. The recovery trace is normalized into an effect-centered representation before compilation so that incidental observations do not become workflow conditions. This keeps the editable surface narrow and makes each accepted update attributable to a concrete contract failure.

Gate 2: constrained local compilation. The incumbent workflow is a tested artifact, so adaptation preserves as much of it as possible. The compiler's only output channel is a list of *patch ops* over a small versioned edit language—`set_selector`, `insert_step`, `replace_step`, `remove_step`, `add_dialog_handler`, `modify_postcondition`, `set_step_timeout`—each anchored to a named step of the incumbent.

Algorithm 1 Progressive behavioral consolidation

1: **execute:** $r \leftarrow \textsc{Execute}(W, x)$ in context c; **if** verified(r) **then** record cell $(c, x) \to \Omega$; **return** deterministic success
2: **recover:** $T \leftarrow \textsc{Recover}(r.\text{state}, x)$; **if** no verified T **then return** AgentFallback
3: **assimilate (fast):** $W' \leftarrow \textsc{Apply}(W, \textsc{Compile}(F(T)))$; persist W' only if Replay $\wedge$ Regression $\wedge$ Generalize
4: **accumulate:** $H \leftarrow H \cup \{\text{IR}(W, T)\}$ ▷ evidence log: audits, and the re-consolidation extension of Section 4.4

Gate 3: selective persistence. A candidate W' is persisted only if it passes (i) **failure replay** on the triggering instance, (ii) **regression** on previously mastered parameters and states, and (iii) **unseen-parameter validation** on a disjoint split. Failed candidates return feedback to the next compile attempt, up to budget K; if none passes, the system retains the previous workflow.

These checks answer complementary questions. Failure replay tests the observed recovery; regression protects the active envelope; held-out validation tests whether the patch extends beyond the triggering page state, with finite empirical evidence rather than a certificate over an unsampled parameter region. Acceptance commits the workflow and its evidence; rejection retains the incumbent and keeps that boundary agent-mediated.

4.4 Re-consolidation: an optional long-horizon extension

Fast local consolidation produces every main-table result. Over long streams, successive patches can make $W + \Delta_1 + \Delta_2 + \cdots$ increasingly specialized. An optional re-consolidation layer aligns the history H into invariants and context-guarded variants, then proposes a replacement that must dominate every cell of Ω plus a probe set. In the 24-stage extension (Section 5.4; Appendix A.2), it reduces artifact complexity from 24 to 13 at matched success, dependence, and tokens.

5 Experiments

We ask two questions: *does consolidation preserve success while changing the execution regime* (Section 5.2), and *which gains come from the recovery solver versus the verified envelope* (Section 5.3)? Section 5.4 then reports an optional long-horizon extension that no main-table number depends on.

5.1 Experimental setup

Environment and drifts. WebArena-Infinity Gmail provides parameterized office tasks with deterministic state-API verification (Zhou, 2026). We use three recurring workflows (`mark_read`, `archive`, `label`), each deployed as a drift stream under eight cumulative drift stages: three shallow drifts (element renames, confirmation dialogs, timing shifts) followed by five structural drifts (procedure split, inserted menu, swapped buttons, conditionally gated action, removed button). Later stages inherit earlier changes. A stream is one (template, order) pair; the campaign covers three templates $\times$ {canonical, permuted} orders, five verification rows per stage, 40 verified tasks per stream. Success and dependence are measured on executed rows only, 32 per stream: each stage's first row injects the drift and is bookkeeping, the remaining four are scored.

Campaign. The main results are a single-source campaign: one runtime (a self-hosted Debian image that hosts application, browser, verifier and agent runtime together), one frozen gateway model, and three independent draws per stream and arm. Tokens are counted by the in-runtime call meter, which records every model call—recovery repeats, compilation and re-consolidation included—so no arm is credited with unmeasured work. Every row is the mean over that arm's three draws, with standard deviation across draws, and every table names its campaign and solver (per-draw values in Appendix C.6; conventions in Appendix E).

Systems. **Full Agent** uses model-mediated execution on every task. **WF+Fallback** runs a fixed seed workflow W_0 and falls back to the agent. **Evo-Harness** runs the authors' released implementation under our campaign runtime, with the shared gateway model, our step budget, and drift injected through our proxy (Wei et al., 2026). **ReUseIt** synthesizes guarded workflows from an agent's successful and failed attempts, taking fallback actions from successes and condition checks

Table 1: Cross-application results under a shared verifier, solver, drift, budget, and accounting protocol. Values are means ± standard deviations over three draws; coverage is model-free replay of the final artifact. Appendix C.6 gives per-draw and Appendix C.5 per-workflow results.

Method	Draws	Success ↑	Dep. ↓	Isol.↑	Cumul.↑	Tok./stream (M) ↓
Gmail (three workflows × two drift orders)						
Full Agent	3	0.992 ± 0.004	1.000 ± 0.000	0.222 ± 0.000	0.000 ± 0.000	7.59 ± 0.09
WF + Fallback	3	0.986 ± 0.006	1.000 ± 0.000	0.222 ± 0.000	0.000 ± 0.000	8.23 ± 0.23
ReUseIt	3	0.988 ± 0.003	1.000 ± 0.000	0.222 ± 0.000	0.000 ± 0.000	15.27 ± 0.42
Evo-Harness	3	0.487 ± 0.068	1.000 ± 0.000	–	–	7.10 ± 0.92
PBC (ours)	3	0.998 ± 0.003	$\mathbf{0.306 \pm 0.134}$	$\mathbf{0.802 \pm 0.065}$	$\mathbf{0.799 \pm 0.060}$	$\mathbf{6.09 \pm 1.05}$
GitLab (three issue-tracking workflows; the fourth fails for every arm, Appendix C.5)						
Full Agent	3	$\mathbf{1.000 \pm 0.000}$	1.000 ± 0.000	–	–	2.70 ± 0.01
WF + Fallback	3	0.694 ± 0.012	1.000 ± 0.000	0.250 ± 0.000	0.000 ± 0.000	1.45 ± 0.04
ReUseIt	3	0.913 ± 0.006	1.000 ± 0.000	0.250 ± 0.000	0.000 ± 0.000	4.91 ± 0.29
Evo-Harness	3	$\mathbf{1.000 \pm 0.000}$	1.000 ± 0.000	–	–	2.82 ± 0.00
PBC (ours)	3	0.948 ± 0.000	$\mathbf{0.111 \pm 0.006}$	$\mathbf{0.722 \pm 0.096}$	$\mathbf{0.861 \pm 0.048}$	$\mathbf{0.34 \pm 0.01}$

Table 2: The envelope ablation under the clean serving convention: all six Gmail streams of one reference draw (both rows are the same draw over the same cells, comparable with each other but not with Table 1). The −envelope configuration also carries the re-consolidation layer, whose effect on these streams is at the noise level; the solver substitution is in Appendix A.7.

Method	Success ↑	Dep. ↓	Isol.↑	Cumul.↑	Tokens/stream ↓
PBC (ours)	**1.000**	**0.151**	0.796	0.833	5.26M
PBC −envelope	0.990	0.255	**0.907**	**0.875**	6.59M

from failures (Liu et al., 2026a); our protocol re-explores once per drift in place of its published failure path, which asks the user for guidance. Its published semantics keep the model in the loop behind those guards, so we report both that native reading and the matched protocol shared by every arm here, in which a re-synthesis must clear the same persistence gate before it serves execution. **PBC** adds the verified envelope and persistence gate. Controlled substitutions remove the envelope or replace the recovery backend; every arm shares the verifier, deployment instances, model, drift channel, step budget, and accounting protocol, and coverage is measured offline by model-free replay of each final artifact (Appendix A.5).

Metrics. Task success; execution-time agent dependence D (the fraction of tasks on which the model was invoked at all); total model tokens; and *isolated* (last-stage) and *cumulative* deterministic coverage of the persistent artifact, measured model-free. Coverage records what the artifact has been proven to execute, so a system can execute deterministically and still fail; we report all three together.

5.2 Main results: consolidation changes execution

Table 1 shows the central result. On Gmail, PBC reaches 0.998 success, compared with 0.992 for Full Agent, 0.988 for ReUseIt, and 0.986 for WF+Fallback: the paired differences run $+0.007$ to $+0.012$ with intervals touching zero, so success is matched while the execution regime changes. Agent dependence falls from 1.000 to 0.306. The final artifact executes 0.802/0.799 of isolated/cumulative drift states without a model, versus 0.222/0.000 for the unchanged seed workflows. On GitLab, PBC reaches 0.948 success against 1.000 for the two full-solver baselines. The corresponding dependence is 0.111 versus 1.000, and tokens fall to 0.34M per stream against 2.7–4.9M. Thus Gmail shows a large execution shift at comparable success; GitLab shows the same shift with an approximately five-point success tradeoff. The two applications differ in what that trade buys: on Gmail the regime change is free, while on GitLab it is priced at about five points of success, a difference that tracks how much of the recurring routine was already consolidated rather than anything about the interface.

How to read the four quantities. They answer different questions about the same runs: success, whether the task was accomplished; dependence, who selected the actions; coverage, what the final

artifact executes with no model in the loop; tokens, what the repetition cost at the gateway. A method can move one of them alone—a deterministic script that fails every drifted task has low dependence, and an agent invoked throughout keeps success high—so the result claimed here is the joint movement of the four, not any single column.

Persistence changes the execution regime. ReUseIt's guarded workflow continues to use a model at drift. Under the matched persistence gate, none of its 144 Gmail or 72 GitLab proposals is accepted, so the seed workflow remains the deployed artifact. This is a substantive distinction between making a procedure available to an agent and moving its execution into an artifact. The matched arm retains the seed's model-free coverage even though the agent can still solve many later tasks.

We also run ReUseIt under its published, ungated document-guided semantics. On Gmail it reaches 0.957 success, compared with 0.988 for its matched-gate arm (paired difference -0.031, CI $[-0.056, -0.009]$). On GitLab it reaches 1.000, recovering the `close_issue` rows that the matched arm loses. Dependence remains 1.000 in both applications. Gmail token use rises to 26.3M per stream, and none of the 76 Gmail or 61 GitLab documents it deploys passes the shared regression check. The comparison shows what ungated adoption buys in immediate reach and what it leaves unresolved: an agent still selects actions on every later instance. Appendix C.2 reports the three arms and paired comparisons.

Statistics. We pair the same streams after averaging each arm over three draws. On Gmail, the mean success difference from Full Agent is $+0.007$ (95% bootstrap CI $[+0.000, +0.015]$), from WF+Fallback $+0.012$ ($[+0.003, +0.023]$), from ReUseIt $+0.010$ ($[-0.002, +0.023]$), and from Evo-Harness $+0.511$ ($[+0.375, +0.657]$). Thus the three strongest Gmail baselines are close in success, whereas their dependence remains 1.000. On GitLab's three jointly solvable workflows, the difference from Full Agent and Evo-Harness is -0.052 (both $[-0.156, +0.000]$), while the differences from ReUseIt and WF+Fallback are $+0.035$ ($[+0.000, +0.104]$) and $+0.253$ ($[+0.021, +0.719]$). These intervals keep the success comparison separate from the directly observed execution shift.

Why success holds as the model leaves the loop. On the mastered-task diagnostic of the introduction the compiled workflow solved every trial at zero model tokens (Appendix D.6). In the stage diagnostic (Appendix C.4), PBC holds 1.000 success across the three shallow stages at 0.292 dependence and 0.997 across the five structural ones at 0.314, while Evo-Harness reaches 0.519 and 0.469: the artifact absorbs the late structural stages instead of losing them. The two dependence levels are close, so structural drift does not push execution back onto the model any more than shallow drift does.

Dependence records which path served each online task; coverage tests the final artifact after the stream, with no agent or model call. Here, dependence falls while task success stays high on Gmail, and the final workflow passes model-free replay on reconstructed drift states. Element, timing, and procedure-split stages reach zero dependence after consolidation; later boundary states still invoke recovery. The pattern identifies retained behavior rather than a transient change in scheduling.

The gate exposes the consolidation boundary. PBC accepts 3.28 updates from 38.3 attempts per Gmail stream (2.50 from 6.9 on GitLab), recording replay, regression, held-out generalization, and patch application for each attempt (Table 3). Regression is the binding check: it fails in 606 of the 630 Gmail attempts that are refused, 53 of which pass replay and are refused by regression alone, while replay alone accounts for 24; no candidate is refused by the held-out check on its own (on Gmail that check is not reached once replay or regression has already refused an attempt, and on GitLab all three fail together in 52 of 53 refusals). The archive and mark-read streams isolate the distinction: the agent solves the drifted task in both, but only archive yields a candidate that passes the gate. An independent structural suite consolidates 17/30 scenarios (Appendix C.8). Gate throughput measures when successful recovery becomes persistent behavior, and per-attempt rejection labels make it auditable (Appendix C.1); the envelope records where it may replace model-based action selection. The asymmetry between the checks is the substantive finding: regression refused 53 candidates that replay had already passed, while held-out validation never refused one on its own, so preserving behavior that was already mastered, not the novelty of a fix, is what most often blocks consolidation.

In archive, the accepted selector rewrite moves the primary control onto an attribute preserved by later structural changes, so subsequent instances remain on the deterministic path. In mark-read,

Table 3: Consolidation attempts, accepted updates and refusal reasons per configuration, all under the clean serving convention (Appendix E.2). Both rows aggregate the six Gmail streams of one reference draw; the PBC row is the control of the gate ablation in Appendix E.3.

Configuration	Attempts/stream	Accepted/stream	Rejection reason
PBC (ours)	38.3	3.28	regression 606 of 630 refusals (53 alone); replay 577
PBC −envelope	34.0	3.50	replay 177 of 183; regression 180 (6 alone)

the same recovery solver completes every drifted task, but candidate edits fail replay or held-out validation and remain outside the active envelope (Appendix B.1). The contrast makes compilability observable: a recovered task supplies evidence, but only a validated edit changes future execution.

5.3 Which input drives what: solver, envelope, and supervision

Each component is isolated by a controlled substitution on the Gmail grid, with one draw per arm so that all switches see the identical stream (Table 3).

A stronger solver raises the ceiling, not the throughput. Holding the loop, gate, envelope, and tasks fixed while replacing browser-use with the lightweight JSON recovery agent moves success from 0.880 to 0.458 and dependence from 0.172 to 0.781 (one reference draw; Table 5). Recovery quality determines which boundary states yield eligible traces; the compiler, gate, and envelope determine whether and where those traces become persistent behavior. The architecture thus separates two improvement directions—a stronger solver expands the recoverable frontier, while better compilation converts more verified rescues into accepted updates without changing the recovery model.

The envelope ablation tests whether recording *where* verified behavior may replace the agent changes online execution. On the same six Gmail streams and one reference draw (Table 2), removing the envelope raises dependence from 0.151 to 0.255 and tokens from 5.26M to 6.59M per stream, while success is 0.990 versus 1.000. The no-envelope arm accepts more updates and has higher final-artifact coverage ($0.907/0.875$ versus $0.796/0.833$). Thus the envelope controls when an accepted workflow is used; it is an execution policy, not a mechanism for maximizing offline coverage.

The aggregate hides useful variation. Removing the envelope improves isolated coverage on archive-canonical (1.000 versus 0.667) and label-canonical (1.000 versus 0.556), but reduces both isolated and cumulative coverage on label-permuted ($0.444/0.250$ versus $0.556/0.500$). On archive-canonical, the no-envelope arm spends 5.74M tokens against 0.37M with the envelope, and its online dependence rises from 0.000 to 0.406. This cell illustrates the envelope's operational role: after a procedure is verified in the current context, routing through that evidence avoids repeatedly purchasing recovery. The clean-serving convention is specified in Appendix E.2.

5.4 Long-horizon artifact growth (optional extension)

On 24-stage cycle streams, the optional re-consolidation layer bounds artifact growth (final complexity 13 versus 24) and expands the verified envelope (135 versus 110 cells) at matched success, dependence, and tokens (Appendix D.1). It folds a stack of accepted local patches into a smaller workflow only after a dominance sweep over previously authorized cells and probes. This extension addresses artifact maintenance over long streams; the execution gains in Tables 1 and 2 come from fast local consolidation alone.

6 Discussion

From recovery to service. Experience can improve an agent's next attempt, but it can also remove the need for an agent on the next repetition. PBC makes the latter measurable. The unit of progress changes with it: a better trajectory helps once, while an accepted update changes every later execution inside the envelope. A verified recovery can become a workflow update, and the active envelope then determines where that update serves future tasks without model-based action selection. On Gmail and GitLab, dependence falls to 0.306 and 0.111, respectively. The success results

delimit the gain: it comes at comparable success on Gmail and with an approximately five-point tradeoff on GitLab.

What becomes persistent. A completed recovery is evidence for a candidate patch, not an automatic update. The archive and mark-read streams show why: the agent recovers in both, but only archive produces an edit that passes the gate. Replay, regression, and held-out validation determine whether the edit can extend the artifact's verified envelope, and acceptance is not a tuned threshold: a candidate that fails any check leaves the incumbent in place, with the failure recorded for audit.

The boundary the gate exposes is not the drift label, and not whether the agent solved the task, but whether the recovery can be expressed as a stable edit of the incumbent workflow. In mark-read the agent finishes the drifted instance and the edit still fails validation, so the rescue stays one-time; in archive an accepted selector rewrite targets an attribute that later drifts retain, and later instances execute without the model.

Two routes to improvement. Recovery quality sets how many boundary failures yield a verified completion; compilation quality sets how many of those become persistent updates (Section 5.3). This separation gives future work a concrete target: richer patch operators and better effect abstractions should be judged by accepted updates and later model-free execution, not by the number of candidate edits generated. A patch language that converted even a third of the regression-blocked candidates would move dependence on the label streams, where the gate refuses 37 of 43 attempts and a third of the executed rows stay model-mediated (Appendix C.1).

Deployment boundary. The envelope makes model use conditional on current evidence, and that authorization is local to the tested context: moving a procedure to an unseen drift order does not carry its verification record, and across the twelve deployment cells one scored task is lost in transfer ($31/32$). Because local coverage has to be re-earned either way, rebuilding costs $1.6\times$ to $5.1\times$ less than shipping the donor artifact (Appendix E.4).

Evidence over time. The append-only verification history and the active envelope have different jobs. History records why an update was once accepted; the envelope authorizes what the current workflow may execute now. An operator therefore reads two different objects: the record of what has been verified at some point, and the subset of it that may run today. If a context changes, the system can withdraw a cell from direct execution while retaining its audit record, then requalify it against the new context. This makes deterministic coverage an operational quantity tied to current evidence, rather than an irreversible claim that a workflow will succeed in every future interface state.

Limitations. The study covers two applications under one model and runtime configuration. Model-free replay measures reconstructed drift states rather than arbitrary unseen states. Coverage is measured over sampled context–instance cells; extending an envelope to new parameters requires new validation evidence. The method also requires checkable task effects and safe replay. Workflows with irreversible side effects would need isolated replay or an equivalent effect checker before a candidate patch could be admitted. These conditions bound where the workflow can replace the agent.

7 Conclusion

PBC converts verified repetitions into executable, auditable behavior. The agent remains available for unfamiliar or broken cases, while the workflow serves the contexts it has earned the right to handle. This shifts the question for self-evolving agents from how much experience they store to how much verified behavior they can execute without reasoning through it again. Two concrete problems follow from the gate's asymmetry. How far can a patch language be extended before a candidate's regression check stops being cheap to evaluate? And what would let a verified cell be re-qualified after a context change without replaying the whole stream? Progress on either would raise the fraction of recoveries that become persistent behavior, which the acceptance data identify as the binding constraint on how much of a routine leaves the agent loop.

AI USE STATEMENT

Generative AI tools assisted with code implementation, experimental planning, analysis and interpretation of results, literature organization, figure construction, and manuscript drafting and editing. Early visual references were used only during design iteration and were not treated as scientific evidence. The authors reviewed and revised AI-assisted content, checked the final code, figures, citations, and empirical claims against the underlying records, and take full responsibility for the manuscript and its supporting artifacts.

ETHICS STATEMENT

This work evaluates web-agent adaptation in controlled benchmark environments and involves no human participants or collection of new personally identifying information. In deployment, compiled browser workflows may access sensitive data or execute consequential actions. We therefore recommend least-privilege credentials, sensitive-data redaction, task-specific external verification, auditable patch histories, abstention under weak verification, and human approval for high-impact operations. Released artifacts will exclude credentials, private infrastructure paths, and data that cannot be redistributed under their source licenses.

REPRODUCIBILITY STATEMENT

Appendices A–E document the experimental protocol, re-consolidation configuration, baseline adaptations, solver settings, end-to-end cases, full results, boundary analyses, and data-quality controls. The supplementary package contains the behavior representation and verified envelope, workflow engine and patch language, validation gate, experiment drivers, fixed configurations, and checkpointed outputs. Each reported result is identified by campaign, runtime, and solver configuration so that the corresponding execution records can be traced and reproduced.

REFERENCES

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable llm agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026.

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.

Yuetian Du, Yucheng Wang, He Xu, Jiexu Xu, Shanwen Tan, Bing Zhao, Boyu Yang, Zhijie Xu, Ming Kong, Hu Wei, Jie Liu, and Qiang Zhu. Living-harness is an interactive-agent evolver. *arXiv preprint arXiv:2607.26598*, 2026.

Srishti Gautam, Arjun Radhakrishna, and Sumit Gulwani. Skillaxe: Sharpening llm-authored agent skills through evaluation-guided self-refinement. *arXiv preprint arXiv:2606.10546*, 2026.

Aayush Gupta. Reliabilitybench: Evaluating llm agent reliability under production-like stress conditions. *arXiv preprint arXiv:2601.06112*, 2026.

Zhezheng Hao, Hong Wang, Jian Luo, Jianqing Zhang, Yuyan Zhou, Qiang Lin, Can Wang, Hande Dong, and Jiawei Chen. ReCreate: Reasoning and creating domain agents driven by experience. *arXiv preprint arXiv:2601.11100*, 2026.

Md Farhan Ishmam and Kenneth Marino. Timewarp: Evaluating web agents by revisiting the past. *arXiv preprint arXiv:2603.04949*, 2026.

Jiaxi Li, Ke Deng, Yun Wang, Jingyuan Huang, Yucheng Shi, Qiaoyu Tan, Jin Lu, and Ninghao Liu. Online skill learning for web agents via state-grounded dynamic retrieval. *arXiv preprint arXiv:2606.04391*, 2026a.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026b.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*, 2018.

Yimeng Liu, Misha Sra, Jeevana Priya Inala, and Chenglong Wang. Reuseit: Synthesizing reusable ai agent workflows for web automation. In *Proceedings of the 31st International Conference on Intelligent User Interfaces*, 2026a. doi: 10.1145/3742413.3789083.

Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, et al. Skillrevise: Improving llm-authored agent skills via trace-conditioned skill revision. *arXiv preprint arXiv:2606.01139*, 2026b.

Shikhar Murty, Christopher D. Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. Bagel: Bootstrapping agents by guiding exploration with language. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 36894–36910. PMLR, 2024.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3135–3144. PMLR, 2017.

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.

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.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025.

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, et al. Evo-Harness: Context-to-harness skill compilation for self-evolving agents. *arXiv preprint arXiv:2608.15071*, 2026.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Shuyan Zhou. Webarena-infinity: Generating browser environments with verifiable tasks at scale. *shuyanzhou.com*, March 2026. URL `https://webarena.dev/webarena-infinity/`.

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*, volume 2024, pp. 15585–15606, 2024. URL `https://proceedings.iclr.cc/paper_files/paper/2024/file/4410c0711e9154a7a2d26f9b3816d1ef-Paper-Conference.pdf`.

APPENDIX: SERVING AND CAMPAIGN CONVENTION

The *clean serving convention* of Appendix E.2 supplies all main comparisons: a unit enters a reported row when no gateway rate limit terminated an episode and was scored as a failure, and the affected Gmail and GitLab cells were re-run from stage 0 under a rate-limit-aware client with one gateway key per unit; runtime, model, browser-use solver, and three independent draws per stream and arm are otherwise shared. The *campaign* supplies the mechanism diagnostics that were not re-run (solver substitution, long-horizon growth, boundary studies) and uses the lightweight JSON recovery agent there. Captions identify the convention; numbers are never pooled across them. The in-runtime meter counts recovery, compilation, and re-consolidation calls.

A EXPERIMENTAL DETAILS

Environment, model, protocol, drift definitions, ablation knobs, and the frozen configuration.

A.1 EXPERIMENTAL PROTOCOL

The compiler model and prompts are fixed across all experiments (Table 6). Recovery is repeated up to three times per drift scenario; the first chronological success supplies the shared trajectory to all repair arms. Repair candidates receive at most three validation-guided attempts. Regression uses reserved base and drifted instances; generalization uses a disjoint parameter partition. Familiarity-protocol arms persist per-episode state and resume from the last completed episode; re-consolidation sweeps are audited per cell with a global budget of 150 cells (dominance rows are never skipped).

A.2 RE-CONSOLIDATION EXTENSION: TRIGGERS, ALIGNMENT AND SWEEP

This section documents the extension that Section 4.4 measures and does not adopt. Re-consolidation fires when at least three assimilations have been accepted since the last re-consolidation, or when artifact complexity has grown by more than 50%; a failed sweep cools down until new evidence arrives. The corpus H contains, for each consolidated episode, an effect-centered intermediate representation (IR) of the full execution—workflow steps converted to IR when the workflow ran, merged with the recovery tail when the agent took over—so complete skeletons, not just recovery fragments, are available to the compiler. The many-trace alignment marks an effect as *invariant* when it appears in a sufficient fraction of contexts; remaining effects become *variants*, each parameterized by its own trace's parameters and guarded by its context. Conflicting alternative paths that realize the same effect are detected by effect-verb families and left to the agent rather than compiled into redundant steps. Timeouts learned by earlier patches are carried over to the re-compiled artifact, since waiting evidence is not recoverable from the corpus alone. The dominance sweep evaluates, per context, the grid of old envelope cells unioned with probe rows, executed model-free under a global cell budget (150 cells); dominance rows are never skipped, and newly covered cells join Ω as coverage gains on acceptance.

A.3 BASELINE PROTOCOLS AND ADAPTATION NOTES

Every repair arm receives the same chronological first verified recovery trajectory; ground-truth drift labels never appear in any prompt; and all LLM-based methods share the frozen model, temperature, and the identical replay+regression+generalization gate. ReUseIt follows the paper's exploration scale (five pristine and five variation demonstrations; guard induction from failed traces), and our protocol adds a re-exploration pass per drift that the published failure path does not perform; candidates are judged by our gate in both compiled and guided modes. Because interactive user guidance is unavailable in the benchmark, re-exploration supplies additional model interaction rather than replacing it with oracle supervision. Appendix C.2 reports the two modes side by side. Evo-Harness runs the authors' released implementation re-wired to our gateway model and drift channel (solver, selector, proposer, curator), with no seed skills, our step budget and timeout, and byte-identical drift injection through a server-side proxy. HarnessFix-adapted mirrors its diagnose-first, repair-second structure over the patch DSL without the recovery trace. SGDR's public implementation targets classic WebArena infrastructure and is excluded from the matched streams.

A.4 THE UNIFORM FAILURE-TRIGGERED RECOVERY RULE

Every workflow-based arm in the main table shares one rule for when the recovery agent takes over, so no arm is penalised by a stricter takeover condition than ours. A workflow execution counts as a failure whenever it does not leave the verified task effect—not only when a step errors. Concretely, the agent is invoked in three cases: (i) a step fails, (ii) a step cannot resolve its target, and (iii) every step executes but the verifier rejects the final state, which is the signature of structural drift (the workflow acted on a shifted element that still exists). Case (iii) matters: without it, a candidate update that silently keeps completing the wrong steps is scored as a plain failure with no recovery, which understates every arm and, for PBC, would also deny the gate the trajectory that would have fixed it. Case (iii) was enforced in all rows of the main table for all workflow-based arms (Full Agent solves from scratch and has no workflow to fail).

A.5 SOLVER AND BACKEND CONFIGURATION OF THE ARMS

The current campaign runs *one* solver configuration across every arm: the campaign's browser-use solver (used for every arm), reasoning disabled at the request level, wired into each paradigm's execution path—as the solving agent for Full Agent, as the fallback for WF+Fallback, as the recovery agent for ReUseIt (the Gmail arm's re-exploration demonstrations are collected by the campaign's lightweight JSON agent, the arm's original configuration; the GitLab arm and every other row use browser-use throughout), as the end-to-end solver of Evo-Harness, and as the in-situ recovery agent inside PBC and its ablations. We do not annotate the main table with solver names because the backend is a property of the campaign's runtime, not a knob that differs between rows; the drift injection channel, verifier, task set, and step budget are likewise shared.

Two substitutions measure the sensitivity of the results to that choice, and both are reported in Appendix A.7: PBC-JSON replaces only the recovery backend inside the fixed loop with the campaign's lightweight JSON recovery agent (group C of Table 1), and the artifact-producing baseline ReUseIt is re-run with the JSON backend in place of browser-use. Neither substitution changes the ordering used in the main text.

A.6 SEED VERSUS CONSOLIDATED ARTIFACT UNDER IDENTICAL DRIFTS

A model-free probe separates "the drifts do nothing" from "the artifact learned around them". In one episode set per drift, the app catalog's *seed* workflow (the testid-locating three-step workflow every arm starts from) and the *final consolidated artifact* of the same template's stream are executed against every drift in isolation, with no agent and no model call in the loop.

Drift (isolated)	Seed workflow	Consolidated artifact
element	$0/5$	$5/5$
dialog	$0/5$	$5/5$
timing	$0/5$	$5/5$
split	$0/5$	$5/5$
insert	$0/5$	$5/5$
swap	$5/5$	$5/5$
conditional	$0/5$	$5/5$
remove	$0/5$	$5/5$
pristine (no drift)	$5/5$	$5/5$

Table 4: Model-free robustness under isolated drifts (archive and mark-read, canonical order, five instances per drift).

This is the control behind the stage table. The structural drifts are effective against the artifact the deployment started with: in isolation the seed is defeated by seven of the eight drifts and survives only swap, which exchanges testids while the action attribute travels with the button. That the structural stages (4–8) do not collapse is therefore the consolidated artifact's doing rather than a property of the drifts, and the accepted updates are what moved its locators onto attributes the drifts do not touch. Dependence on those stages is not zero (0.314 against 0.292 over stages 1–3) because the gate refuses some of their candidates; where it accepts, the corresponding rows run model-free.

Method	Recovery backend	Success	Dep.	Isol.	Tokens/stream
PBC	browser-use (campaign solver)	0.880	0.172	0.870	2.91M
PBC	lightweight JSON agent	0.458	0.781	0.444	6.77M
ReUseIt	browser-use (campaign solver)	0.781	1.000	0.222	10.07M
ReUseIt	lightweight JSON agent	0.312	1.000	0.222	9.40M

Table 5: Solver sensitivity on the campaign grid (one reference draw). Only the recovery agent changes; all other protocol choices are fixed.

A.7 Solver sensitivity on the campaign grid

The substitution is deliberately narrow: the loop, the gate, the envelope and the task set are untouched, so a change in success measures how much of the result is carried by the recovery solver rather than by consolidation. Read together with Table 1, it separates two questions that an aggregate success number conflates—*how often the agent can solve a drifted task* (solver quality) and *how often a solved task becomes part of the artifact* (compilation under accumulation).

A.8 Core Configuration

Item	Frozen value
Compilation/recovery model	deepseek-v4-flash (gateway)
Temperature / max output	0.0 / 3,000 tokens
Agent action budget	26 steps
Recovery repeats / compile attempts	3 / 3
Familiarity buffer	$\geq$3 successes over $\geq$2 distinct instances
Re-consolidation triggers	$\geq$3 assimilations or complexity $+50\%$
Dominance sweep budget	150 cells (dominance rows never skipped)
Action / navigation timeout	8s / 20s
Timing diagnostic probe	12s
Observation elements / prompt budget	280 / 24K characters
Regression instances	2 reserved instances
Generalization instances	up to 5 disjoint instances

Table 6: Frozen configuration used by the reported experiments.

B End-to-End Case Study

One drift that the loop absorbs on the first attempt, and the same drift type on a second stream where the gate refuses every candidate. All numbers are task and gate records of the current campaign, not schematics.

B.1 A drift that consolidates on the first attempt

Stream: archive email, canonical order, PBC arm.

1. **Incumbent artifact.** The artifact begins as the seed workflow the app catalog defines: three steps, each locating its control by `data-testid`.
2. **The drift arrives.** Stage 1 renames the control the workflow's first step clicks, so the locator stops resolving. The stage's first instance (the trigger row) fails inside the workflow, and control passes to the recovery agent *from the live failure state* rather than from a fresh page.
3. **Verified recovery, then the gate.** The agent completes the instance; the compiler turns the trajectory into a candidate update; and the gate checks it three ways: *replay* on the failure instance (✓), *regression* on the two reserved instances (✓), and *generalization* on five held-out parameter sets ($5/5$). One attempt, accepted.
4. **What the accepted update changed.** The final artifact locates the primary control as `[data-action="archive-email"],[data-testid="toolbar-archive"]` and carries an explicit `action_timeout_ms: 12000`

learned at stage 3 (timing drift). The catalog's seed locates the same control by testid alone, so the stable attribute is the compiled update's doing, not the seed's.

5. **Observable consequence.** All four remaining instances of the stage run *without the model*: 0 agent steps, 0 calls, three deterministic workflow steps, 0.36 s each. At the end of the stream the envelope holds 59 verified cells (from 18 after the first stage), isolated coverage is $9/9$ drift states and cumulative coverage $8/8$ stacked contexts, and the unit's 32 executed rows all pass with dependence 0.000.

B.2 The same drift type, refused: element drift on mark-read

Stream: mark read, canonical order, same arm. Stage 1 renames a different control (the entry checkbox), and here the loop does not absorb the drift. Each of the stage's five rows produces a recovery and a candidate update, and every candidate is refused: with three attempts per candidate, *replay* fails in most attempts, *regression* fails in some, and *generalization* never reaches the required $5/5$ (several attempts fail replay while passing regression, one candidate is refused for a patch-application error, `step 'select_email'`). Fifteen attempts, five candidates, none accepted.

	Archive, stage 1	Mark-read, stage 1	Label, stage 1
Candidates (rows)	1	5	3
Gate attempts	1	15	3
Accepted	1 (first attempt)	0	1 (third candidate, first attempt)
Refusal signature	—	replay $\times$ / regression $\times$; unseen $< 5/5$	regression $\times$ on candidate 1
Stage's later instances	0 calls, 0.36 s each	agent-mediated, $4/4$ still pass	$4/4$ model-free after acceptance

The failure is not the solver's: in every one of the five rows the agent completed the drifted task and the verifier confirmed it. What fails is the step that turns a verified rescue into a persistent update—the candidate either does not replay where the experience was learned or does not survive the held-out instances. Note also what the gate does *not* do: because the stream keeps running, a refused candidate is not a lost stage. The refusal is retried on the next instance with the validation feedback attached, and the stage's remaining tasks stay correct because the agent takes them.

B.3 What to take from the case

Recovery quality and consolidation success are different variables. The mark-read stage shows a solver that solves every instance and still publishes nothing; the archive stage shows a solver that solves one instance and publishes an update that then covers the next four. Only the second removes the model from the loop, and only the gate's per-attempt record (replay, regression, generalization, patch errors) distinguishes them. This is also why the main table reports dependence next to success: on the mark-read unit the two arms look identical in success and differ completely in what was learned.

C Full Results

Complete tables behind the main-text claims: per-stream, per-draw, component substitutions and the boundary diagnostics.

C.1 Full Continual-Stream Results

Table 7 reports per-stream rows for the six principal systems over the six deployment streams (three workflows $\times$ two orders; 40 executions each). Isolated coverage tests the final workflow on independently reconstructed historical drift states; cumulative coverage tests it under stacked deployment contexts.

Per-stream narrative. Three patterns cover the streams, and they are visible in Table 7 rather than asserted here. *Streams where the artifact takes over*: the archive streams need three and five attempts in total, two updates are accepted, and every later instance of the drift—including the structural drifts that arrive on top of it—runs through the compiled artifact with no model call (dependence 0.000/0.031, coverage at or near 1.000). *Streams where the gate refuses*: on the label streams 37 of 43 and 33 of 38 attempts are refused, only six and five updates are accepted, so a third of the rows

stay model-mediated (dependence 0.312) and the artifact covers 0.556/0.500 of the states; mark-read-canonical is refused almost as often (17 of 19) but its two accepted updates suffice, and that stream ends fully covered at dependence 0.125. *Streams where the released harness fails*: Evo-Harness drops to 0.150–0.525 on the label and mark-read orders where every other arm stays at or above 0.969, so the residual failures in this table are the harness's, not a property of the loop.

Table 7: Per-stream results (**clean serving convention**, one reference draw per arm). The envelope ablation has its own tables (2, 3) and is not repeated here.

Workflow	Order	Method	Success	Dep.	Iso. cov.	Cum. cov.
archive	canonical	Full Agent	1.000	1.000	0.222	0.000
archive	canonical	WF+Fallback	0.969	1.000	0.222	0.000
archive	canonical	ReUseIt	0.969	1.000	0.222	0.000
archive	canonical	Evo-Harness	0.725	1.000	0.000	0.000
archive	canonical	PBC (ours)	1.000	0.000	0.667	1.000
archive	permuted	Full Agent	1.000	1.000	0.222	0.000
archive	permuted	WF+Fallback	0.969	1.000	0.222	0.000
archive	permuted	ReUseIt	1.000	1.000	0.222	0.000
archive	permuted	Evo-Harness	0.425	1.000	0.000	0.000
archive	permuted	PBC (ours)	1.000	0.031	1.000	1.000
label	canonical	Full Agent	0.975	1.000	0.222	0.000
label	canonical	WF+Fallback	1.000	1.000	0.222	0.000
label	canonical	ReUseIt	1.000	1.000	0.222	0.000
label	canonical	Evo-Harness	0.175	1.000	0.000	0.000
label	canonical	PBC (ours)	1.000	0.312	0.556	0.500
label	permuted	Full Agent	1.000	1.000	0.222	0.000
label	permuted	WF+Fallback	1.000	1.000	0.222	0.000
label	permuted	ReUseIt	0.969	1.000	0.222	0.000
label	permuted	Evo-Harness	0.150	1.000	0.000	0.000
label	permuted	PBC (ours)	1.000	0.312	0.556	0.500
mark-read	canonical	Full Agent	1.000	1.000	0.222	0.000
mark-read	canonical	WF+Fallback	1.000	1.000	0.222	0.000
mark-read	canonical	ReUseIt	1.000	1.000	0.222	0.000
mark-read	canonical	Evo-Harness	0.575	1.000	0.000	0.000
mark-read	canonical	PBC (ours)	1.000	0.125	1.000	1.000
mark-read	permuted	Full Agent	1.000	1.000	0.222	0.000
mark-read	permuted	WF+Fallback	1.000	1.000	0.222	0.000
mark-read	permuted	ReUseIt	1.000	1.000	0.222	0.000
mark-read	permuted	Evo-Harness	0.525	1.000	0.000	0.000
mark-read	permuted	PBC (ours)	1.000	0.125	1.000	1.000

Why ReUseIt remains fully agent-mediated. Across the three reported draws, ReUseIt proposes 144 Gmail and 72 GitLab workflow re-syntheses; none passes the shared persistence gate. Its deployed artifact therefore remains the seed workflow, so each scored drift instance returns to the recovery agent and dependence stays at 1.000. This differs from Full Agent, which has no artifact, and WF+Fallback, which never attempts an update. ReUseIt attempts to learn, but no accepted behavior reaches future execution—the distinction between producing a candidate and changing the execution regime.

C.2 ReUseIt under its own execution semantics

The main table reports ReUseIt under the matched protocol, where a re-synthesis must clear the shared persistence gate before it serves execution. Its published semantics are different: the synthesized workflow carries guards, the model is still called to act behind them, and a re-synthesis is deployed without an external gate. We ran that reading as a third arm on the *full* streams—six Gmail streams and the three jointly solvable GitLab workflows, three draws each, same solver, model, drift injection, step budget and in-runtime meter as every other arm (Table 8). The native arm differs

Table 8: ReUseIt under its own semantics against the shared protocol and PBC, on the full streams (means ± standard deviations over three draws; the GitLab block is over the three jointly solvable workflows, as in Table 10, since the milestone workflow fails for every arm). Coverage is model-free replay of the final artifact; for the native arm the artifact is a document the model follows, so its figures are a counterfactual replay of that document, recorded but never part of its online execution.

Method	Draws	Success ↑	Dep. ↓	Isol.↑	Cumul.↑	Tok./stream (M) ↓
Gmail (three workflows × two drift orders)						
ReUseIt (native semantics)	3	0.957 ± 0.006	1.000 ± 0.000	0.000 ± 0.000	0.014 ± 0.024	26.31 ± 1.52
ReUseIt (shared gate)	3	0.988 ± 0.003	1.000 ± 0.000	0.222 ± 0.000	0.000 ± 0.000	15.27 ± 0.42
PBC (ours)	3	0.998 ± 0.003	0.306 ± 0.134	0.802 ± 0.065	0.799 ± 0.060	6.09 ± 1.05
GitLab (three jointly solvable workflows)						
ReUseIt (native semantics)	3	1.000 ± 0.000	1.000 ± 0.000	0.000 ± 0.000	0.056 ± 0.048	6.13 ± 0.09
ReUseIt (shared gate)	3	0.913 ± 0.006	1.000 ± 0.000	0.250 ± 0.000	0.000 ± 0.000	4.91 ± 0.29
PBC (ours)	3	0.948 ± 0.000	0.111 ± 0.006	0.722 ± 0.096	0.861 ± 0.048	0.34 ± 0.01

from the matched ReUseIt arm in exactly two respects: every row is executed with the deployed document prepended to the solver's task text, and a re-synthesis is adopted unconditionally. Both arms start from the same seed workflow and re-explore at the paper's scale, and the native arm still *evaluates* the shared gate on every candidate it adopts, so the two readings—success, dependence, tokens—are read off identical candidates.

Removing the gate does not raise success. Given its own protocol, ReUseIt reaches 0.957 on Gmail—*below* the matched reading (0.988; paired -0.031, CI $[-0.056, -0.009]$) and below PBC (0.998; -0.042, CI $[-0.069, -0.016]$)—and 1.000 on the three jointly solvable GitLab workflows, above both (0.913, paired $+0.087$, CI $[+0.000, +0.260]$; PBC 0.948, $+0.052$, CI $[+0.000, +0.156]$). The reviewer's question therefore resolves against the simpler reading: on Gmail ungated adoption costs about three points of success, on GitLab it recovers the `close_issue` rows the gated arm loses (1.000 against 0.740 there), and neither application shows a success gain that would justify taking the model out of the gate's evidence chain.

What the ungated adoption actually deploys. Across the draws the native arm triggered 137 times on Gmail and 61 times on GitLab. Gmail produced a candidate on 76 triggers (61 ended with all re-exploration demonstrations failing, so no synthesis existed); *every* adopted candidate on both applications is one the shared gate refuses ($76/76$ and $61/61$). The refusal is not conservatism: of the 76 deployed Gmail documents, 18 reproduce the triggering fix (replay), 12 handle unseen instances, and **0 pass regression**—every document the ungated arm installs breaks behavior that had already been verified ($0/61$ on GitLab as well). Deploying them does not make the tasks fail, because the model still reads the page and can work around a stale document; what it cannot do is make the artifact executable.

The artifact does not become execution. Replaying the deployed document model-free covers 0.000 of the isolated states on Gmail (0.000 on GitLab) and 0.014 of the cumulative contexts (0.056 on GitLab). The native arm's cost shows why: with the model on every row it needs 26.3M tokens per Gmail stream against PBC's 6.1M, and 6.1M against 0.34M on GitLab, while PBC is also the only arm whose final artifact both executes model-free and covers the states it has been verified in (0.802/0.799 on Gmail, 0.722/0.861 on GitLab). Read together with Table 1, the comparison separates two questions a single success number merges: whether a method can solve a drifted task, and whether what it produces becomes durable execution.

Hard streams under a clean serving path. The campaign collected those cells while the shared gateway was rejecting bursts of requests (HTTP 429; 283–563 rejections per unit on Gmail, 61–336 on two GitLab shards), and a rejected call kills a browser-use episode, so the rows were scored as task failures and the affected cells read 0.41–0.78. Under the clean serving path of Appendix E.2 (rate-limit-aware client, one gateway key per unit, capped concurrency) the four hard streams pool to $364/384$ for the native arm, $380/384$ for the matched arm and $383/384$ for PBC at dependence 0.294 and 8.34M tokens; the campaign's spread on those cells was therefore a serving artefact, not a property of the arms, and canonical-order streams were unaffected. Appendix E.2 states the exclusion rule this implies.

Table 9: Where the agent leaves the loop: success and agent dependence by drift stage, pooled over the three draws of the clean rebuild (Gmail). Stages 1–3 are shallow drifts; stages 4–8 are structural drifts that arrive cumulatively on top of them. This is the stage diagnostic behind Section 5.2, not the head-to-head comparison of Table 1.

Stage	Drift	PBC dep. ↓	PBC succ. ↑	Evo-Harness succ. ↑	Accepted	Gap
1	element	0.208	1.000	0.444	16	+0.556
2	dialog	0.306	1.000	0.567	14	+0.433
3	timing	0.361	1.000	0.544	11	+0.456
4	split	0.333	1.000	0.489	6	+0.511
5	insert	0.292	1.000	0.400	5	+0.600
6	swap	0.222	1.000	0.489	3	+0.511
7	conditional	0.278	1.000	0.533	4	+0.467
8	remove	0.444	0.986	0.433	0	+0.553
1–3	shallow	0.292	1.000	0.519	41	+0.481
4–8	structural	0.314	0.997	0.469	18	+0.528

C.3 Per-Stream Results

The per-stream rows of the clean rebuild are in Table 7. The cross-environment table averages three draws per stream and arm (Appendix C.6); the paired statistics in Section 5.2 are computed on the six stream pairs after each arm is averaged over its draws.

C.4 Where the loop opens: by drift stage

The stage table is the fine-grained complement to the cross-environment table: it says where inside a stream the artifact stops absorbing drift, which a single aggregate cannot.

C.5 GitLab: the second application in detail

Aggregate without the milestone workflow. One of the four GitLab workflows (changing an issue's milestone) fails at the application level for every arm: the agent does not reliably select the required milestone from the dropdown, no arm consolidates a verified recovery there, and the workflow contributes zeros to every row of the main table (Evo-Harness succeeds on 3 of 120 rows, every other arm on none). That workflow is also the shard where the campaign's serving rejected the most requests (285–332 per unit), so its zeros were re-checked under the clean serving path of Appendix E.2: the rebuilt units reproduce them for every arm, and the native-semantics arm also scores $0/32$ there with no gate candidate and no adoption, which fixes the reading as an application-level failure rather than a rate-limit artefact. The main table's GitLab row covers the three workflows every arm can solve; the milestone workflow appears in the per-workflow table below.

Method	Success ↑	Dep. ↓	Isol.↑	Cumul.↑
Full Agent	1.000	1.000	–	–
WF + Fallback	0.694	1.000	0.250	0.000
ReUseIt	0.913	1.000	0.250	0.000
Evo-Harness	1.000	1.000	–	–
PBC (ours)	0.948	0.111	0.722	0.861

Table 10: GitLab aggregate over the three jointly solvable workflows (clean serving convention; three draws per arm).

Per-workflow results. Table 11 gives every stream of the second application, including the one that fails for all arms, so the zeros in Table 1 can be traced to a single workflow rather than to the loop. The strictest reading—restricting every arm to the (workflow, stage) cells all arms executed—coincides with the main table's GitLab panel, since every arm ran the full grid; that view is therefore not repeated here.

Method	Workflow	Success	Dep.	Tokens (K)
Full Agent	set_label	1.000	1.000	2948.9
Full Agent	set_assignee	1.000	1.000	2813.8
Full Agent	close_issue	1.000	1.000	2339.7
Full Agent	set_milestone	0.000	1.000	22456.0
WF + Fallback	set_label	0.979	1.000	1636.0
WF + Fallback	set_assignee	0.979	1.000	1202.6
WF + Fallback	close_issue	0.125	1.000	1496.9
WF + Fallback	set_milestone	0.000	1.000	3257.0
ReUseIt	set_label	1.000	1.000	4504.1
ReUseIt	set_assignee	1.000	1.000	4274.4
ReUseIt	close_issue	0.740	1.000	5958.2
ReUseIt	set_milestone	0.000	1.000	3122.1
Evo-Harness	set_label	1.000	1.000	2974.0
Evo-Harness	set_assignee	1.000	1.000	2966.3
Evo-Harness	close_issue	1.000	1.000	2523.4
Evo-Harness	set_milestone	0.025	1.000	35317.0
PBC (ours)	set_label	1.000	0.021	180.2
PBC (ours)	set_assignee	1.000	0.031	174.5
PBC (ours)	close_issue	0.844	0.281	662.3
PBC (ours)	set_milestone	0.000	1.000	3779.4

Table 11: GitLab per-workflow results (clean serving convention; three draws pooled per arm; tokens are per-draw means).

C.6 PER-DRAW METRICS

Every row of Table 1 is the mean of three independent draws of the campaign protocol, and the standard deviation there is taken across those draws. The tables below list the draws themselves, so each aggregate can be recomputed from its parts and the spread is visible rather than implied. For a row given as a range, the three values are in draw order. Success and dependence are pooled over the environment's executed rows (six Gmail streams, four GitLab workflows; trigger rows are bookkeeping and are never scored), coverage replays the final artifact of that draw model-free, and tokens are the run meter's per-stream figure for the draw.

Draw sets are fixed before reporting and no draw is selected on its outcome; an unfinished unit is dropped by the same rule for every arm. The units are the clean rebuild of Appendix E.2: all five Gmail arms were re-run, while the GitLab cells that showed no rate-limit event (set_label, set_assignee) keep their campaign runs, which is why a GitLab per-draw value can coincide with the campaign's. One further note: the Gmail coverage columns are measured for every arm against the workflow it deploys, which is why the arms that never update their workflow (Full Agent, WF+Fallback, ReUseIt) carry the seed workflow's figures while Evo-Harness, which solves from scratch, has none.

Method	Draw	Success ↑	Dep. ↓	Isol.↑	Cumul.↑	Tokens/stream
PBC (ours)	1	1.000	0.151	0.796	0.833	5.26M
	2	1.000	0.396	0.870	0.833	7.27M
	3	0.995	0.370	0.741	0.729	5.75M
Evo-Harness	1–3	0.429 / 0.562 / 0.471	1.000	–	–	6.73 / 8.14 / 6.42M
Full Agent	1–3	0.996 / 0.992 / 0.988	1.000	0.222	0.000	7.67 / 7.61 / 7.50M
WF + Fallback	1–3	0.990 / 0.990 / 0.979	1.000	0.222	0.000	8.03 / 8.19 / 8.48M
ReUseIt	1–3	0.990 / 0.984 / 0.990	1.000	0.222	0.000	15.58 / 14.80 / 15.43M

Table 12: Gmail, per draw (clean serving convention, six streams per draw). The dependence, isolated and cumulative columns are identical across draws for every baseline, so they are printed once; success and tokens are listed in draw order.

The spreads are worth reading against the means. On GitLab our dependence stays within 0.008 across draws (0.328–0.336) while the baselines move by at most 0.014 in success, so draw-to-draw variation is small for every arm there. On Gmail our own spread is wider than the baselines' (0.995–

Method	Draw	Success ↑	Dep. ↓	Isol.↑	Cumul.↑	Tokens/stream
PBC (ours)	1	0.711	0.328	0.562	0.625	1.24M
	2	0.711	0.336	0.688	0.688	1.14M
	3	0.711	0.336	0.562	0.625	1.22M
Evo-Harness	1–3	0.762 / 0.750 / 0.756	1.000	–	–	12.29 / 9.82 / 10.73M
Full Agent	1–3	0.750 / 0.750 / 0.750	1.000	–	–	8.10 / 7.59 / 7.23M
WF + Fallback	1–3	0.516 / 0.516 / 0.531	1.000	0.250	0.000	2.00 / 1.83 / 1.86M
ReUseIt	1–3	0.688 / 0.688 / 0.680	1.000	0.250	0.000	4.65 / 4.14 / 4.61M

Table 13: GitLab, per draw (clean serving convention, second application, four workflows per draw, the milestone workflow included). Dependence is identical across draws for every arm and is printed once; success and tokens are listed in draw order.

1.000 success, 5.26–7.27M tokens, dependence 0.151–0.396), and the token variation is carried by the streams where candidates are refused and the agent takes over the rows it then rescues—the label streams most of all (11.2–17.9M across draws)—the mechanism Section 5.2 reads from the token column.

C.7 Mechanism Ablations

Naming. All configurations run the *same* loop and differ only in two switches: the verified envelope (the recording structure over which coverage is defined) and the re-consolidation extension of Section 4.4. We therefore name them by what is switched off — ours; ours −slow; ours −envelope; and *compile-only (ours)* (−slow, −envelope). Any row labelled *Compiler* in earlier drafts refers to this last configuration, not to a third-party system.

Table 14: Evidence contribution by Gmail drift type (two seeds). B=symptoms, D=complete verified recovery, E=unified evidence.

Variation	B	D	E
Element	0.700	0.900	**1.000**
Dialog	0.000	0.900	**1.000**
Timing	1.000	1.000	1.000
Silent	0.000	0.400	0.400
Workflow	0.000	**0.500**	0.400
Overall	0.340	0.740	**0.760**

Table 15: Modification-scope ablation under matched recovery evidence and validation.

Edit strategy	Repair ↑	Regression failure ↓
Whole-workflow rewrite	0.219	0.667
Minimal patch (ours)	**0.719**	**0.091**

C.8 Boundary Analyses

Recovery-backend substitution. Because consolidation reuses the recovery trajectory as supervision, the recovery agent is the method's most consequential single component. We swap it inside the fixed loop: everything else—task set, drift injection, verifier, gate, envelope, model, step budget, runtime—is unchanged. The substitution separates two quantities an aggregate success number conflates: how often a drifted instance is rescued (solver quality) and how often the resulting candidate survives the gate (compilability). The campaign's solver rescues more instances than the lightweight JSON agent, and the extra candidates it produces are refused at the same rate as the JSON agent's, so the gate—not the solver—is what bounds what the artifact learns (Table 5).

Deep structural drift. We inject five structural semantics (remove, swap, split, insert, conditional) over both seeds, 30 scenario-runs. Localized repair baselines recover zero of 30; ReUseIt compiles 6/30; the compiler reaches 17/30 (77% conditioned on verified recovery). Compilability splits by semantic: split is almost always compilable; swap is hardest (1/6; testid swaps break the anchor between failure and fix). Where recovery itself fails, the case stays with the agent.

Table 16: Persistence-gate diagnostic. Each cell is accepted candidates / accepted false consolidations. Rules are nested and reuse identical candidates.

Candidate source	Replay only	+ Regression	+ Held-out parameters
Ours: local patches (24)	20 / 1	19 / 0	19 / 0
ReUseIt: rewrites (17)	5 / 5	1 / 1	0 / 0

Drift order decides learnability. Under the permuted order, dialog becomes unlearnable for every arm (0/5): arriving after timing and element drifts, its patches are rejected in sequence and the missed consolidation cascades. Consolidation throughput drops (two accepted per permuted stream vs. three on canonical), and final coverage falls. What can be consolidated depends on the order in which the world changes—a deployment property that offline skill induction does not expose.

The extension stops for lack of evidence, not on a schedule. Replacement remains possible deep in a stream, and each accepted fold follows a fresh batch of assimilations. Between replacements, every trigger ends with the verified incumbent retained. On GitLab the extension stops for the opposite reason: after the first fold the fast layer absorbs every stage deterministically, no new evidence arrives, and the trigger never re-arms. The verified envelope makes these boundaries safe, because unverifiable drift, adversarial orders and dominance-rejected candidates stay outside Ω, and the system degrades to a strong recovery loop instead of corrupting behavior it has already mastered.

D Additional Analyses

The long-horizon measurement of the re-consolidation extension, the isolated structural suite, acquisition asymmetry, the cross-application grid, the second domain, token amortization and where the agent's input budget goes.

D.1 Familiarity-Protocol Details

Long stream: robustness at the doubled horizon. The 51-episode long stream interleaves both drift orders and rotates task parameters every episode: success is at parity across arms (20–23/51, within takeover variance), and PBC again holds lower final complexity (16 vs. 17) with more verified cells (32 vs. 29); the two persistence styles price structural breadth differently (Table 18).

Table 17: Familiarity protocol, canonical stream, earlier campaign (archive; 19 episodes: 3 cold-start + 16 drift).

Arm	Success	Agent dep.	Final cx	Ω cells	LLM calls	Tokens
PBC (full)	**14/19**	16/19	**16**	**32**	**163**	**1.31M**
PBC-norecon	11/19	16/19	17	29	251	2.01M
PBC-single	11/19	16/19	17	29	247	1.98M
Compile-only (ours, W_0 given)	12/19	13/19	–	–	231	1.88M

The compiler arm receives a hand-validated W_0; PBC arms start from zero experience. Complexity is the final artifact score; cells are verified envelope entries. All four arms reach identical final coverage (0.556 isolated / 0.375 cumulative): the extra PBC mechanisms do not harm reachable coverage on this stream.

Table 18: Familiarity protocol, long stream, earlier campaign (archive; 51 episodes: cold start + 24 stages with per-episode parameter rotation).

Arm	Success	Agent dep.	Final cx	Ω cells	LLM calls	Tokens
PBC (full)	20/51	48/51	**16**	**32**	931	7.51M
PBC-norecon	21/51	48/51	17	29	942	7.65M
PBC-single	23/51	48/51	17	29	944	7.63M
Compile-only (ours, W_0 given)	23/51	45/51	–	–	915	7.42M

Isolated coverage: PBC arms 0.556 (element, dialog, timing, swap, pristine), compiler 0.778 (additionally conditional and remove via its patch stack); cumulative coverage 0.125 for all arms. The compiler's extra isolated states come at complexity 17 versus the re-consolidated artifact's 16—the two persistence styles price structural breadth differently.

Table 19: Familiarity protocol, cycle stream, earlier campaign (archive; 27 episodes: 12 stages, three repairable drifts cycled with deepening variants).

Arm	Success	Agent dep.	Final cx	Ω cells	LLM calls	Tokens
PBC (full)	27/27	9/27	**19**	**67**	47	0.39M
PBC-norecon	27/27	9/27	20	64	45	0.38M
PBC-single	27/27	9/27	20	64	53	0.46M
Compile-only (ours, W_0 given)	27/27	6/27	–	–	26	0.23M

All arms reach 1.0 isolated and cumulative coverage. The PBC re-consolidation events: episode 7 replacement accepted (complexity 17→16, +3 cells, 0 dominance misses); later triggers in this window produce no dominating candidate and the verified incumbent is retained.

Table 20: Cycle-24 streams (51 episodes: cold start + 24 stages, three repairable drifts cycled eight times with deepening variants), across templates and applications.

Stream	Arm	Success	Agent-free	Final cx	Ω cells	Folds
Gmail archive	PBC	49/51	36/51	**13**	**135**	8, 40
	PBC-norecon	51/51	38/51	24	110	—
Gmail star	PBC	48/51	33/51	**11**	**127**	8, 22, 46
	PBC-norecon	48/51	32/51	20	98	—
GitLab relabel	PBC	48/51	45/51	**27**	**69**	8
	PBC-norecon	48/51	45/51	30	65	—

Folds lists episodes with an accepted replacement. Every PBC arm reaches full isolated coverage on its stream (archive and star 4/4 states; GitLab 4/4), and the GitLab artifact covers all 24 cumulative contexts model-free. Success is matched within takeover variance on every stream; the separation is entirely in artifact complexity and verified-cell count.

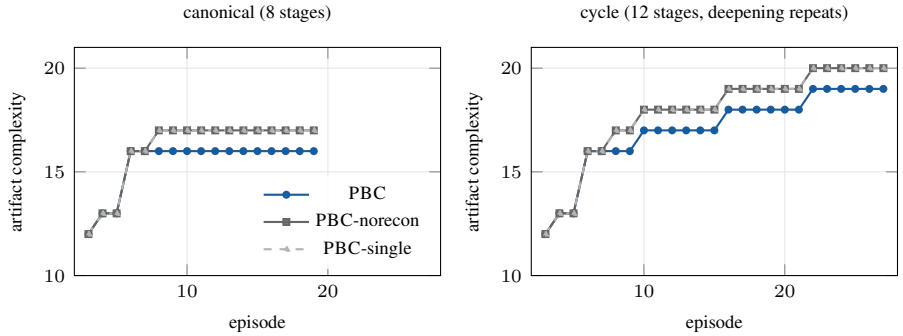


Figure 2: Artifact complexity along the shorter familiarity streams (archive task). Without re-consolidation (grey), every assimilation permanently raises complexity; with re-consolidation (blue), the accepted replacement at the first trigger flattens the curve, and the gap persists to the end of the stream.

Permuted familiarity stream. Under the permuted order (19 episodes, server-run, all arms on identical code), final coverage is 0.444/0.250 for every arm and the dialog drift is never consolidated (0/5): its patches are rejected in sequence after timing and element drifts have already been absorbed, and the missed consolidation cascades. Success: PBC 11/19, norecon 10/19, single 13/19, compiler 12/19—within agent-takeover variance, as expected on a stream whose differences are concentrated in consolidation throughput rather than solving.

D.2 Structural-Change Suite

Fifteen scenarios (three workflows × five structural semantics) are executed twice under the frozen protocol. Pooled end-to-end acceptance is 17/30 (56.7%, Wilson 95% CI 39.2–72.6); conditioned on the identical verified-recovery set, 17/22 (77.3%). All localized repair baselines remain 0/30; ReUseIt reaches 6/30. Per-scenario cross-run agreement is 12/15 for the compiler (flips on archive–insert, label–split, mark-read–swap).

Table 21: Structural semantics pooled over two independent runs (**earlier campaign**; successful runs; 6 trials per semantic, 30 total).

Semantic	Static	Recovery	Agent	Healer	Failure-only	HarnessFix	ReUseIt	Compile-only (ours)
Remove	0	4	4	0	0	0	2	4
Swap	0	4	5	0	0	0	0	1
Split	0	6	4	0	0	0	1	5
Insert	0	4	4	0	0	0	0	3
Conditional	0	4	4	0	0	0	3	4
Total	0	22	21	0	0	0	6	**17**

D.3 Acquisition Asymmetry

First-time consolidation is substantially harder than updating an incumbent procedure. Under a five-stage cold-start protocol, Gmail acquisition succeeds on the first attempt for all five templates, but one-demonstration compilation passes the full gate only for archive and label; star, mark-read, and unstar are correctly rejected (single trajectories cover only the primary tab). A demonstration-count sweep shows a step function, not a curve: compilable templates pass with $K{=}1$, non-compilable templates are rejected for all $K \in \{1, 2\}$—compilability is a task-structure property. On GitLab, acquisition fails on all four templates within the step budget, whereas 10/20 GitLab drift scenarios compile once a validated workflow exists (Appendix D.4): cold start must discover the complete task skeleton, while re-consolidation only has to infer the increment.

Table 22: End-to-end first-time consolidation (five-stage protocol).

Environment	Template	Acq.	First workflow	Rep. (held-out)	Re-consolidation
Gmail	archive email	1/1	✓	4/4	3/3
Gmail	label email	1/1	✓	1/4	3/3
Gmail	star / mark-read	1+1 (C)	gate rejection	—	—
Gmail	unstar	1/1	gate rejection	—	—
GitLab	four templates	0/4	no trajectory	—	—

D.4 Cross-Application Baseline Grid

The scenario grid of §5.2 was replicated on WebArena-Infinity GitLab (issue tracking): four templates (relabel, reassign, re-milestone, close) crossed with the five per-template drifts gives twenty scenarios, run under the same arms, frozen model, budgets, and replay–regression–generalization gate. Every LLM-based arm receives the same chronological first verified recovery trajectory for a scenario; where no verified recovery exists, the compiler receives nothing and abstains.

Table 23: GitLab cross-application grid (**earlier campaign**; 20 scenarios; agent steps are per invocation, tokens pooled).

Arm	Success	Agent steps	Prompt tokens	Compiled	Unseen cells
Agent-only	2/20	25.1	0.99M	—	—
Static workflow	0/20	0	0	—	—
Workflow + fallback	15/20	9.3	1.37M	0/20	—
Selector healer	0/20	—	—	0/20	—
ReUseIt	3/20	—	68K	3/20	13/75
Ours (compiler)	10/20	—	121K	10/20	22/75

Three observations. First, the frozen workflow never survives a drift cold (0/20) and agent-only solving rarely fits the step budget (2/20): the eight-step GitLab skeletons are harder to discover than Gmail's three-step ones, and takeover succeeds precisely because it inherits the workflow's UI state (9.3 steps per invocation against 25.1 from scratch). Second, the workflow-first fallback is the strongest end-to-end runner (15/20 scenarios) but pays a fresh agent episode for every repeat (1.37M prompt tokens pooled) and compiles nothing; every repeat re-pays the agent. Third, the compiler accepts 10/20 at roughly a tenth of that budget (121K tokens, four of ten on the first attempt) and covers 22/75 unseen-instance cells among attempted repairs, ahead of every repair baseline (ReUseIt 3/20 and 13/75; selector healer 0/20). Locator-level drifts (element, dialog, timing) account for ten of the twelve acceptances; the ten rejections split into three silent-decoy scenarios where the takeover agent itself fails—no verified trajectory exists, so the compiler correctly declines to consolidate

an unverified signal—and seven gate rejections dominated by workflow-semantics drifts (0/4) that change what the task is rather than where a control lives.

D.5 Supplementary Generalization: MiniWoB++

As a third environment we ran the same scenario protocol on MiniWoB++ (Liu et al., 2018) (seven task types—logins, form sequences, checkbox lists, search, and email reply—crossed with the drift levels, 37 scenarios). The point of this grid is narrow: recovery-to-consolidation is not a property of Gmail's DOM. On miniature interfaces the agent alone is strong (30/37), the frozen workflow still dies on drift (3/37), and the compiler accepts 25/37—ahead of every repair baseline (ReUseIt 7/37, selector healer 9/37 repaired)—including 15/21 at the intermediate drift level and 6/7 at the hardest. Consolidation from a verified recovery transfers across a third, structurally simpler environment; we keep it out of the main comparison because these tasks do not exercise the long-horizon axes the study is about.

D.6 Efficiency of Mastered Repetition

A 300-execution mastered-task diagnostic compares execution regimes on Gmail tasks the systems have already consolidated. The compiled workflow completes 100% of repetitions with zero model tokens and zero agent calls; the full agent completes 53.3% at 34.3M tokens; adding the verified skill document raises success to 59.7% at 30.6M tokens with dependence unchanged at 1.0. A context-component attribution (replaying logged trajectories, rebuilding per-step prompts, tokenizing each component) shows why: 88.8% of the full agent's prompt tokens are serialized page state versus 2.2% reasoning output; the skill document does not reduce the observation share (87.0%). On mastered work the model is paid mostly to re-read the page, not to decide.

Table 24: Component attribution of model context on mastered Gmail tasks (per-task means, scaled to server-reported totals).

Method	Observation	Skill/instruction	History	Reasoning	Total
Full agent	18.9K (88.8%)	1.8K	0.1K	0.5K (2.2%)	21.3K
Agent + skill	14.4K (87.0%)	1.9K	0.1K	0.2K	16.5K
Compiled workflow	0	0	0	0	0

D.7 Token Amortization

The six-stream window is front-loaded with novelty, so the question is whether the consolidation cost has been paid by the time the window closes. Averaged over the three draws of the clean rebuild it has: summing the six per-stream figures, the loop spends 36.6M tokens against 42.6M for Evo-Harness (6.09M and 7.10M per stream in Table 1), so $K^* = 0$ against every agent-mediated baseline—no additional covered repetition is needed to pay compilation back, and after the window executions the artifact covers run deterministically at $\approx$0 tokens while the baselines keep paying their measured marginal cost per repetition (Evo-Harness: 177K tokens per execution). The accounting is the run meter for every arm—solver, compilation and re-consolidation calls alike; the workflow arms are charged in addition for their 48 drift-trigger rows, which are checkpoints rather than scored tasks, so the loop is charged 240 rows (192 scored plus 48 triggers) against Evo-Harness's 240 solved rows. The per-stream split is still instructive: on the four streams whose per-execution cost is below Evo-Harness's the loop is already 19.2M cheaper in total, while the two label streams spend 13.1M more—refused and retried candidates, the churn Section 5.2 reads—which nets to the 6.0M window advantage. Inside the window the loop is also below Full Agent (45.5M) and WF+Fallback (49.4M). The calculation assumes post-window traffic revisits consolidated behavior—the normal case for recurring-routine deployments—and is directional rather than a measured end-to-end saving.

Table 25: Long-horizon consolidation on the 51-episode cycle-24 streams with 24 drift stages (earlier campaign).

Stream	Method	Final complexity	Envelope cells	Accepted folds
Gmail archive (24 stages)	PBC	**13**	**135**	2
	PBC-norecon	24	110	–
Gmail star (24 stages)	PBC	**11**	**127**	3
	PBC-norecon	20	98	–
GitLab (cross-app)	PBC	**27**	**69**	1
	PBC-norecon	30	65	–

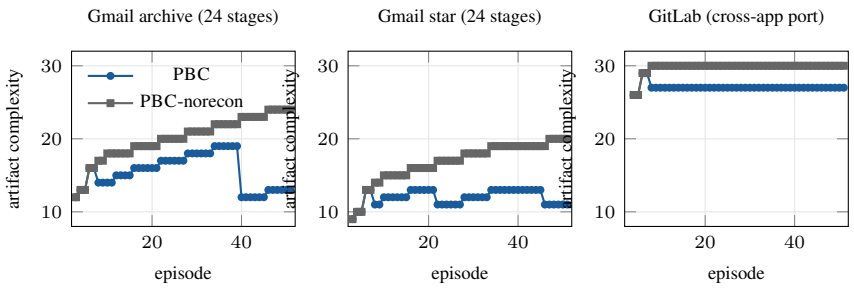


Figure 3: Artifact complexity on the cycle-24 streams for the re-consolidation extension (**earlier campaign**, JSON recovery backend). Without re-consolidation (grey) complexity climbs monotonically; with it (blue), accepted replacements fold the stack back and the verified envelope ends larger at matched success.

E Reproducibility and Data Quality

Campaign bookkeeping and exclusion rules for the reported results.

Runtime and serving sensitivity. The same arm can differ substantially across serving paths because structured-output behavior and request-level reasoning settings affect the solver. Every primary comparison therefore uses one runtime and serving configuration; the container campaign is the reference for all current numbers. Excluded units and raw per-unit records are documented below.

E.1 Campaign Details

What each table measures. Tables 1–3, the per-draw tables of Appendix C.6 and the gate ablation of Appendix E.3 report the clean serving convention of Appendix E.2: the affected units were re-run from stage 0, the remaining cells are runs that showed no rate-limit event. The solver-sensitivity, boundary and mechanism tables use the earlier campaign grid. The cross-environment results average three draws; ablation, component, stage, and solver-sensitivity tables use one reference draw on a shared grid. Table 25, Figure 3, and Appendix D.1 use the earlier JSON-backend campaign to measure long-horizon growth and individual mechanisms. Captions identify the convention, and results are neither pooled nor compared numerically across conventions.

E.2 Data Quality

Data-quality note. Four rules determine which units enter Table 1: (i) a unit is excluded and re-run when a gateway rate limit *changed its result*—it terminated a model call or a task episode and the outcome was scored as a failure. A rate-limited call that the client retried successfully, with the episode continuing normally and the retries counted by the meter, leaves the unit valid: rate-limit counts locate a problem but are not themselves the criterion. Applied to the served campaign, the affected units are exactly those whose logs show terminated episodes: every retained `close_issue` unit (4–62 episodes terminated by a rate limit; 33–387 events) and every `set_milestone` unit (37–79; 390–504), whereas `set_label` and `set_assignee` show no rate-limit event at all and are kept. Where no log survives—all Gmail campaign draws—the check falls back to the trajectory:

a unit whose executed rows all succeeded cannot have lost an episode that was scored as a failure, so its success, dependence and token columns stand, while its coverage column remains unproven (a killed recovery on a trigger row leaves no failing row behind). (ii) A wave with an incompatible recovery/compiler overlay is excluded because compilation failed before every gate attempt; (iii) one completed unit with zero recorded model calls is excluded as degenerate; and (iv) every included row uses the same in-runtime meter and scoring convention. The audit of the served campaign counted 61–336 rejections in each retained GitLab draw (concentrated on two shards, `set_milestone` and `close_issue`) and 283–563 per unit in the first pass of the Gmail hard streams, of which 4–79 per affected unit terminated an episode and were therefore scored as failures; the affected cells were re-run under rule (i) and are reported in Appendix C.2, while the campaign's own values for them are kept as provenance only. The per-arm client ledger is retained only as a consistency check because it under-counts browser-use calls by construction.

E.3 Which gate check carries the weight

The persistence gate applies three checks in sequence: replay, regression, and held-out generalization. Re-running the same six Gmail streams under `XB_GATE_POLICY=regression` (held-out dropped) and `XB_GATE_POLICY=replay` (regression dropped as well), with the shipped `full` policy as the control on the same serving path, solver, step budget and stream definitions, separates their contributions: two draws per cell, each compared against the control's same-index draw. Table 26 counts the cells that change; "worse" means the cell loses a scored row, isolated or cumulative coverage, or a previously verified behavior (retention).

Policy	Cells	Identical	Differ	Worse	Better
held-out dropped (replay+regression)	12	4	8	3	3
regression dropped (replay only)	12	1	11	9	0

Table 26: End-to-end gate ablation on the six Gmail streams, two draws per stream. Each cell is compared against the control's same-index draw and every landed cell is reported; no cell is selected on its outcome.

The two policies separate cleanly. Dropping the held-out check leaves behavior within draw-to-draw variation: the differences run both ways (3 cells better, 3 worse, 2 moved without a loss or a gain) and the check never decides a candidate on its own—0 of the 484 attempts the arm records were refused by held-out alone, and 0 of the control's 689. Dropping regression as well is systematic: 11 of the 12 cells move, nine of them worse and none better, and only there do scored rows, coverage and retention collapse together—both label-canonical cells end with the final artifact covering *no* isolated state (0.000 against 0.556 and 0.667), the archive-permuted cell ends at 0.111 against 1.000 and $30/32$ against $32/32$ success, and one label-canonical cell ends with seven previously verified behaviors broken where the control breaks none.

Two mechanisms produce the pattern. First, the weaker rule changes the stopping condition of the search: `repair_and_validate` returns the first attempt its policy accepts, so accepting a replay-only patch ends the search before the candidate that would have passed regression appears. One mark-read cell therefore ends with *no* adoption where the control adopts two, at isolated coverage 0.222 against 1.000 and cumulative coverage 0.000 against 1.000. Second, the same adoption count can hide different candidates: one label cell admits five updates under both policies, yet its cumulative coverage is 0.500 against 0.875, and a second admits five against six while breaking seven retained behaviors. The arm records 700 attempts and 661 refusals over the twelve cells, of which 0 are refused by held-out alone and the control's 689 attempts likewise contain none. The gate's value is therefore not "one more filter": replay establishes reproducibility, regression is the load-bearing quality condition that decides where the search stops, and the held-out check has no measurable marginal effect in this corpus—its two labels (offline prior and online ablation) agree.

E.4 Deployment transfer: an unseen drift order

The main table builds its artifact on the drift orders it is reported on; a deployment is asked whether the artifact survives an order it has not met. We ship the canonical artifact of each template to `permuted_b`, an order no experiment has ever run, and compare it there against a fresh build on the same order—same runtime, gateway model, browser-use solver, step budget and gate. The document travels: every arm solves the new order (32/32) on all three templates, with one exception—on label

the shipped artifact loses one scored row that a fresh build holds. The record does not travel. A transferred PBC arrives without the verified cells of its home order, so the engine falls back to recovery until they are re-earned; mark_read shows that in its purest form, where the *same* document runs the whole order with zero model calls under document-mode execution while PBC pays twenty recoveries for the same rows. The re-earning cost, not the acquisition, decides the comparison: shipping costs 1.6×, 2.6× and 5.1× a local rebuild across the three templates. The deployment reading is therefore rebuild, do not ship—and what must be rebuilt is the record, not the procedure.

Method	Success	Dep.	Isol./Cum.	Serve (M)	Acq.+serve (M)
`archive_email`					
PBC transferred	32/32	1/32	1.00/1.00	0.73	1.10
PBC seed (order control)	32/32	0/32	1.00/1.00	0.43	0.43
ReUseIt gated transferred	31/32	8/32	0.67/0.75	1.90	2.27
ReUseIt native transferred	32/32	32/32	0.00/0.12	17.55	17.92
donor, own order	32/32	0/32	0.67/1.00	0.37	0.37
`mark_read_email`					
PBC transferred	32/32	20/32	0.89/0.38	2.96	3.63
PBC seed (order control)	32/32	4/32	1.00/1.00	0.71	0.71
ReUseIt gated transferred	32/32	0/32	1.00/1.00	0.00	0.67
ReUseIt native transferred	32/32	32/32	1.00/1.00	12.95	13.62
donor, own order	32/32	4/32	1.00/1.00	0.67	0.67
`label_email`					
PBC transferred	31/32	28/32	0.33/0.12	26.00	41.32
PBC seed (order control)	32/32	28/32	0.33/0.12	26.13	26.13
ReUseIt gated transferred	32/32	24/32	0.56/0.25	17.73	33.06
ReUseIt native transferred	32/32	32/32	0.00/0.00	22.01	37.33
donor, own order	32/32	10/32	0.56/0.50	15.33	15.33

Table 27: Deployment transfer to an unseen drift order (`permuted_b`), one draw per cell, same runtime and gate as every other table. The last column charges a *transferred* artifact for the donor run as well as its own serving; a fresh build and the donor's home order pay their own run only, so their total equals the serving figure. ReUseIt's coverage is a counterfactual replay of its document, whose execution is always model-mediated.