Compile Once, Govern Every Repair: Deterministic Replay for Repeated GUI Work

Richard Abrich
OpenAdapt.AI (MLDSAI Inc.)
richard@openadapt.ai

Draft of July 2026

Abstract

Reasoning through a known workflow on every run is wasteful and unsafe. It adds latency, inference cost, and run-to-run variation, and—because a rendered success banner is not a persisted write—it can report success after a partial, duplicate, stale, or rejected business effect. We present OpenAdapt, a demonstration compiler that converts one recorded GUI trace into a deterministic program. Healthy replay follows structured elements and local visual evidence with *no model calls*; when evidence changes, a resolution ladder re-resolves the target and records the change as a reviewable *repair*; and the runtime *refuses* to continue when configured identity, postcondition, effect, or policy checks cannot verify an action. The central safety mechanism is system-of-record effect verification: a consequential step declares a typed effect and checks it against the application's own state, not the screen. Measured end to end through the live replayer and judged by an independent system-of-record ground truth, screen-only verification silently accepted 54 of 90 injected-fault runs; an out-of-band effect check cut that to 9 of 90 and a complete system-of-record read path to 0 of 90, the residual nine being one collateral-write class outside the out-of-band oracle's read coverage. The finding replicates on a second, non-healthcare lending domain: 21 of 33 silent wrong effects under screen-only verification, 0 of 33 under effect verification. Bounded experiments span a public OpenEMR workflow, a CI-reproducible fixture, a 29-application public-web corpus, transactional fault injection, adversarial identity tests, and scoped qualifications of native Windows UIA, macOS, and real-network RDP mechanisms. Across these tasks compiled replay reduced latency and per-run inference while preserving independently checked effects and ambiguity refusal. Every headline number in this paper is bound by a released machine-check to its raw artifact; the release publishes the implementation, reviewed artifacts, and failure taxonomy.

1 Introduction

Much operational computer work is *repeated*: the business intent is stable, the sequence is mostly known, errors are costly, and the same task may run hundreds of times. Reasoning through such a workflow on every run is both wasteful and unsafe. Wasteful, because a general model re-derives the same plan each time, paying latency, inference cost, and run-to-run variation. Unsafe, because the usual success signal is the *screen*: a model or a script sees a rendered "Saved" banner and stops—but a banner is not a persisted write. The same green pixels can follow a partial save, a duplicate submission, a stale overwrite, or a silently rejected transaction. The failure mode that matters in production is not the visible crash; it is the *silent wrong action* that reports success.

Graphical user interfaces remain the practical integration boundary for legacy, third-party, and remote applications, so this problem cannot be assumed away with an API. Selector-based automation is fast and deterministic when stable structure is available but brittle under interface drift, while visual scripting extends to pixel surfaces [15]. Recent computer-use agents instead observe the screen and choose each action with a general model [1, 11, 4], a useful formulation for previously unseen tasks and broad benchmarks [16, 6, 14]. Neither line closes the safety gap: RPA does not check the system of record, and an agent re-reasons every run yet still trusts the screen.

OpenAdapt treats a successful human demonstration as source material for a *program* rather than as context for a model on every run (Figure 1). Compilation extracts parameters, redundant target evidence, postconditions, and risk metadata. Replay follows the compiled program deterministically and makes zero model calls on the healthy path; models are optional repair tiers, not the controller. Where a step is consequential, an independent verifier checks the effect against the application's own system of record, and the runtime refuses rather than guessing when it cannot verify. The distinction the paper foregrounds is not whether a task *completed* but whether identity and external effects were *verified*, and whether the system can *refuse* when evidence is insufficient. In an injected-fault study this distinction is decisive, and we measure it end to end through the live replayer and judge it by an independent system-of-record ground truth: screen-only verification silently accepted 54 of 90 fault runs, an out-of-band effect check cut that to 9 of 90, and a complete system-of-record read path to 0 of 90 (Section 6). The same gap replicates on a second, non-healthcare lending domain.

This paper makes five contributions:

1. A demonstration compiler and backend-neutral workflow representation for deterministic repeated GUI execution.
2. A resolution ladder that prefers structured identity, then local visual evidence, and records bounded drift repairs as reviewable changes.
3. System-of-record effect verification that treats the screen as a weak oracle and the application's own state as the strong oracle, closing the gap between a rendered success and a persisted effect.
4. Governance mechanisms that distinguish runnable from certified, verify identity and system effects where configured, and refuse rather than silently degrade.
5. An evaluation protocol that reports task success *together with* silent incorrect success and over-halt, with raw artifacts, a failure taxonomy, and *machine-checked paper constants* that bind every headline number to a released benchmark file.

The evaluation deliberately separates a broad product architecture from the scope of each measured result. Browser workflows supply the comparative and breadth studies. Additional task-bound qualifications cover Windows UIA, native macOS, and real-network RDP with independent effect oracles and ambiguity or stale-target refusals. These results establish working substrate mechanisms for the named environments; they are not a population estimate for arbitrary enterprise applications. Citrix ICA/HDX is a pixel-only remote-display environment in the same class as the measured RDP backend, but no Citrix result is reported here: the qualification covers RDP over Aardwolf into a Parallels Windows guest, not an ICA/HDX session, and RDP evidence does not transfer to Citrix. The limitations section states this scope precisely and names the specific validation it would take.

2 Related Work

GUI automation and RPA. Robotic process automation systems execute rules over existing application interfaces and organizational workflows; their adoption and governance questions are surveyed by van der Aalst et al. [13]. Screenshot-driven automation predates current multimodal agents. Sikuli made visual patterns a first-class interface for search and scripting [15]. OpenAdapt shares the objective of operating interfaces that lack a practical API, but combines structured elements and visual evidence in an ordered ladder and makes a repair an auditable program change.

Programming by demonstration and synthesis. Programming by demonstration infers reusable behavior from user actions rather than requiring conventional source code [5]. Programming-by- example systems such as FlashFill demonstrate how compact examples can produce deterministic programs while exposing ambiguity in the inferred intent [8], and web-scraping-by-demonstration systems such as Rousillon compile direct manipulation into a reusable program [3]. A GUI trace similarly under-specifies intent: coordinates do not identify the semantic target, and one trajectory does not determine every branch or loop.

Table 1: Where OpenAdapt sits. Columns are the properties that matter for repeated, consequential GUI work; ✓ = designed-in, ∼ = partial or tool-dependent, blank = not addressed by the paradigm.

Approach	No per-run model call	Deterministic healthy path	Repairs under drift	Verifies effect	Refuses on ambiguity
Selector RPA [13]	✓	✓			
Visual scripting [15]	✓	✓	∼		
PBD / synthesis [5, 8]	✓	✓			∼
Computer-use agents [1, 16]			✓		∼
OpenAdapt	✓	✓	✓	✓	✓

OpenAdapt therefore retains redundant target evidence, represents control flow explicitly, and quarantines underdetermined induction instead of filling gaps with an unverified guess. It also departs from this line in two ways: it repairs targets at runtime under drift, and it verifies downstream effects rather than only reproducing input actions.

Computer-use agents and benchmarks. A rapidly growing line of computer-use agents observes the screen and chooses each action with a general multimodal model [1, 11, 4], evaluated on breadth-oriented benchmarks such as WebArena, WorkArena, and OSWorld [16, 6, 14]. These emphasize task completion under online reasoning on previously unseen tasks. Our question is complementary: after a workflow has been *demonstrated*, when can execution be compiled so that repeated healthy runs need no model decision? We retain task-level oracles but additionally measure silent incorrect success and over-halt, which distinguish unsafe success from conservative refusal—quantities these benchmarks do not report.

Runtime verification, oracles, and end-to-end effects. Runtime verification synthesizes monitors that evaluate execution traces against explicit safety properties [9], and the end-to-end argument places a correctness check at the layer that can establish the intended property rather than an intermediate subsystem [12]. Deciding whether an observed outcome is correct is the *test-oracle problem* [2]; the faults OpenAdapt targets—duplicate submission, double delivery, stale overwrite—are the same exactly-once and idempotency hazards studied in transactional systems [7]. These principles motivate OpenAdapt's separation between screen postconditions and system-of-record effects: a rendered success banner cannot independently establish that a transaction was persisted correctly. Unlike a general formal-verification claim, the guarantees here remain scoped to armed steps, declared effects, configured verifiers, and named policies.

3 System Design

3.1 Record and compile

A recorder captures actions, frames, element observations, and structural state when available. Declared secret fields are replaced with runtime parameters. The compiler emits a bundle containing a workflow program, target crops, locators, landmarks, postconditions, parameter declarations, and a manifest. The linear trace is the simplest program; the intermediate representation also supports states, guarded transitions, branches, loops, and subflows. Induction that cannot determine a branch condition quarantines the result rather than inventing one.

3.2 Resolution ladder

For an action target, replay tries evidence in descending order of fidelity: structured DOM or accessibility identity, local and global template matching, OCR, landmark geometry, and an optional grounding model. A backend presents a small common contract for observation and actuation. This provides a pixel floor without discarding structured evidence on browsers and native desktops.

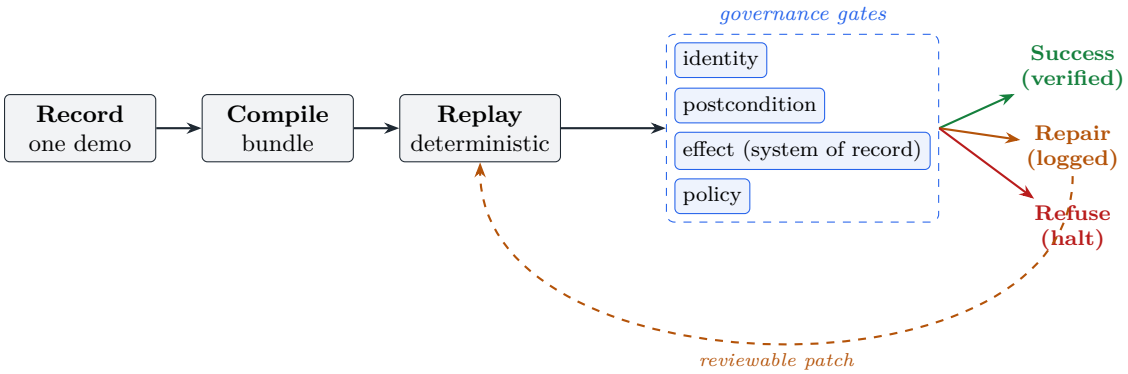

Figure 1: OpenAdapt compiles one demonstration into a deterministic program. Healthy replay makes no model calls. Each step passes governance gates; the runtime verifies and succeeds, records a bounded *repair* when a lower- fidelity rung re-resolves a changed target, or *refuses* when a configured check cannot be verified. A repair is an auditable patch, never a silent overwrite of the deployed bundle.

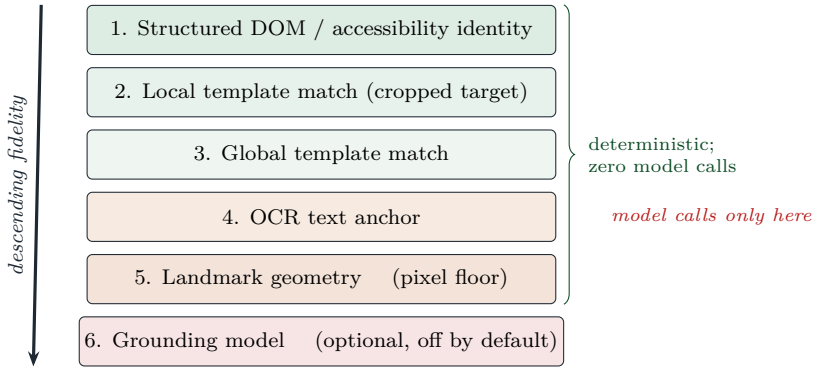

Figure 2: The resolution ladder tries evidence in descending order of fidelity. Structured identity is preferred; local visual evidence gives a pixel floor for substrates without a DOM. Rungs 1–5 are deterministic and make no model calls. A grounding model is an optional last rung, disabled by default, and its rescue is subject to the same identity and effect checks.

Healthy execution stops at deterministic tiers and makes zero model calls. A lower rung resolving a changed but verified target creates a repair patch. The runtime can save a healed derivative bundle, but does not silently replace the deployed source bundle.

3.3 Postconditions and reports

Each step may carry visual or structural postconditions such as text presence, region stability, URL change, or window state. Replay writes a human-readable and machine-readable report with rung, confidence, identity verdict, postconditions, effects, model calls, elapsed time, repair, and halt reason. Postconditions are necessary but not sufficient: a success banner is evidence about the screen, not independent evidence about a database write.

3.4 Backends

The browser backend is the reference implementation and supplies published end-to-end evidence against a real third-party application. Windows UI Automation adds structured candidate enumeration and native actions behind a typed, authenticated in-session contract. Native macOS binds capture and input to a unique

application window. The RDP backend exposes the same runtime contract over a pixel-only network session. Section 4 reports scoped task and effect qualifications for all three mechanisms. Citrix/ICA/HDX remains a separate deployment environment and does not inherit RDP qualification.

4 Governed Execution

4.1 Identity before action

Target resolution answers where to act; identity verification asks whether the resolved target belongs to the intended entity. Structured text is authoritative where available. On pure-pixel substrates, OCR ambiguity can make distinct identifiers observationally identical. The safe response is abstention and halt, not a guessed click. Importantly, identity guarantees apply only to armed steps. Current real bundles can contain unarmed clicks, so coverage is reported per workflow and certification policy can impose a minimum.

4.2 Effect verification

A screen may paint success after a partial, duplicate, rejected, or stale write. Consequential steps can therefore declare typed effects and use an independent verifier against REST, FHIR, or document state. The verifier snapshots the system of record before and after an action and returns confirmed, refuted, or indeterminate. A declared effect without a configured verifier halts.

Crucially, the strong-oracle read is performed *out of band*: the verifier queries the system of record through its own interface (a SQL read on a read-only role, a REST or FHIR read, or a document or file read), independently of the substrate that carried the write. It does not parse the pixel stream or scrape the rendered page. The same effect contract therefore holds whether the write was driven through a browser DOM, native accessibility, or a pixel-only remote session: what changes is only the connector, not the check. Because the read is out of band, an unreachable, unreadable, or unauthenticated system of record maps to indeterminate and halts; the runtime never upgrades "I could not read the record" into a reported success.

Two consequences follow for pixel-only substrates such as RDP and Citrix. First, the effect check still applies as long as some out-of-band read path exists: the RDP qualification in Section 6, for instance, reads the persisted file through guest tools, a channel independent of the RDP display. Second, when a locked-down session exposes *no* such path, no database, no reachable API, and no readable file, the transactional strong oracle is simply unavailable. The only fallback is a same-application re-read of the record, which cannot catch a partial save, a duplicate, a stale overwrite, or a lost update, and the honest runtime treats that as a halt rather than a confirmation. On such substrates the safety guarantee rests instead on the identity gate and halt-on-ambiguity, not on effect verification. Effects are currently authored per deployment; the compiler does not infer them from a single demonstration. This is the classic *oracle problem* [2] made concrete: the screen is a weak oracle and the application's own state is the strong oracle (Figure 3). Many of the faults it catches—duplicate submission, double delivery, stale overwrite—are exactly-once and idempotency hazards long studied in transactional systems [7].

4.3 Policy and refusal

The linter exposes unarmed clicks, vacuous postconditions, and risk-classifier gaps. Certification evaluates a named policy before deployment. A strict run gate can require certification, identity and effect coverage, encryption, and manifest integrity. Passing a policy means only that its assertions hold; it is not a general safety proof.

4.4 Sanitized artifact derivatives

Compilation is not de-identification. When an artifact crosses a boundary, the launch protocol inventories the source, transforms a copy with type-specific handlers, rescans the derivative, records unresolved findings and tool versions, and binds local review to the derivative hash. Unknown or unresolved content is refused. Sanitized authoring evidence is distinct from runtime observations: opening a live patient record can reintroduce protected information, which must remain inside the declared trusted execution boundary.

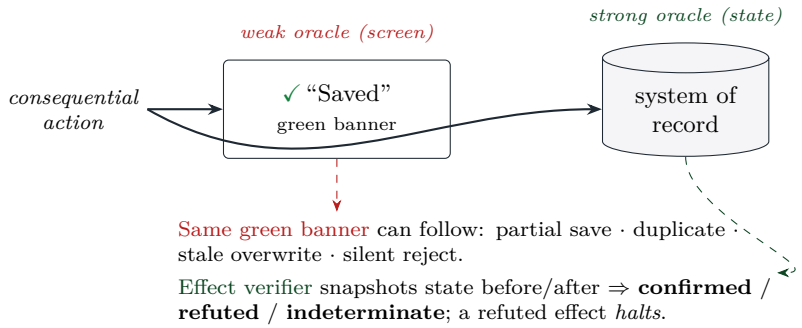

Figure 3: Effect verification separates the weak screen oracle from the strong system-of-record oracle. A rendered success is evidence about pixels, not about a persisted write; only an independent before/after snapshot of the application's state can refute a silent wrong action. In the fault study this reduced silent acceptance from 50 of 90 runs to 0 of 90.

5 Evaluation Methodology

We separate comparative task performance, breadth, transactional safety, and identity availability. A single aggregate success rate would hide materially different errors.

Comparative conditions. Both repeated-workflow comparisons were generated on 2026-07-08 on the recorded platform `macOS-15.7.3-arm64-arm-64bit`. The compiled and agent arms drove the same vision-only Playwright backend: screenshots in and pixel clicks, typed text, key presses, and scrolling out, with no DOM selectors at runtime. The agent condition used `claude-sonnet-5` with the `computer_20251124` tool and `computer-use-2025-11-24` beta header. Agent costs are calculated from recorded token usage and the list-price schedule stored in each raw result; the compiled condition recorded no model calls. The agent is a single-attempt observe-act loop given the task goal and the same settled screenshots the replayer uses, without a system prompt, planning or reflection scaffolding, or whole-task retry, and bounded by per-run action and cost caps. It is a fair same-backend reference rather than a maximally scaffolded agent, so the latency and cost gaps should be read as lower bounds on what scaffolding could close, not as a reliability comparison.

Outcome taxonomy. Success means the external task oracle is satisfied. A safe halt stops without the intended effect. Over-halt is a safe halt when the intended correct action was in fact available. Silent incorrect success means the runtime reports success while the external oracle disagrees. False abort means the intended effect occurred but the runtime reports failure. We report run counts rather than confidence intervals for small bounded studies.

OpenEMR comparison. The task logs into the shared public OpenEMR demonstration site, searches a patient, opens Patient Messages, adds a parameterized note, and saves. The compiled arm has 20 runs and the computer-use-agent arm 10. Both use the same task-level success check. The shared public instance is mutable and the study is not fully reproducible in CI; it is field evidence, not a production study.

MockMed comparison. The bundled browser fixture performs a triage workflow. The compiled arm has 100 runs and the agent arm 20. This is the CI-reproducible anchor, but the application is purpose-built and is not evidence about hostile enterprise UIs.

Public-web reliability corpus. Twenty-nine public, unauthenticated applications are recorded, compiled, and replayed once on an unchanged UI using an external goal oracle. This study is descriptive: one run per application does not meet the repeated-trial standard for a capability claim, and the selection excludes SSO-walled and desktop targets.

Transactional fault model. Seven persistence faults and controls are injected at a real HTTP persistence boundary behind the MockMed UI, with 10 repeats per class. Database state, not the UI report, is the oracle. This per-class study measures what screen-only verification silently mishandles; it is separate from the end-to-end study below.

End-to-end effect verification. The headline safety result is measured through the actual governed replay path, not an in-process stub. Ten persistence scenarios, seven fault classes plus an accepted-write control and two delivery controls, are each run nine times per arm. Every write is issued by the replayer's actuator as an HTTP POST to an on-disk SQLite system of record, and each run is judged by three genuinely distinct paths: the write itself; an effect verifier that reads the record back over a different endpoint and connection than the write; and an independent ground truth that opens the SQLite file read-only over every table and classifies the before/after state without ever seeing the write's success flag. Because the verifier and the ground truth share neither code nor state, the effect verifier's zero is earned rather than definitional. We evaluate three arms: screen-only banner verification, an out-of-band effect oracle whose read path covers the record surface, and a complete effect oracle whose read path covers every mutable surface the action can touch. This supersedes an earlier in-process study in which the effect verifier and the ground truth read the same object, which made its zero circular by construction. Zero model calls, localhost only.

Second domain: lending. To test generalization beyond healthcare, the same effect-verification protocol is replicated on MockLoan, a loan-disbursement authorization workflow with its own ledger. Eleven tasks spanning all seven divergence categories plus clean and idempotent controls are each run three times per arm, judged by an independent REST verifier that reads the ledger at an endpoint the application never calls. One task seeds a same-name decoy loan to exercise the identity gate on the money-movement step. Zero model calls.

Released benchmark. The metric and fault taxonomy are packaged as EffectBench, a standalone, versioned, and independently runnable benchmark (specification version 1.0.0), so that the silent-wrong-effect rate can be measured against any GUI or API automation, not only ours.

Identity ladder. Adversarial same-name/same-date-of-birth homonyms differ by confusable glyphs such as O/0 and l/1. The actual replayer stack is evaluated with structured and pure-pixel configurations. False accept and over-halt are both reported because a system can obtain zero wrong clicks by refusing every correct one.

Desktop and remote-substrate qualification. These mechanism studies are task-bound rather than comparative. Windows UIA is evaluated on one in-tree WinForms Patient Notes workflow in a Windows 11 ARM VM. Each counted trial starts from a clean snapshot, writes a trial-unique note, and uses an independent SQLite query as the effect oracle; stale and ambiguous candidate controls must leave every row unchanged. Native macOS is evaluated on one macOS 15.7.3 arm64 host using TextEdit, exact file-byte readback, and a two-window ambiguity control. The preserved batch report classifies graceful- close cleanup warnings as failure; a separate hash-bound offline adjudication accepts only the three action effects and ambiguity refusal after verifying the exact harness processes and temporary root were absent. RDP is evaluated over Aardwolf 0.2.14 into one 1280x800 Parallels Windows 11 snapshot: input through the Windows Run dialog creates a trial-unique file whose exact contents are read back through guest tools independently of the RDP channel. Each promoted task has three counted trials, records model calls, silent incorrect success and over-halt, and retains environment-cleanup evidence.

6 Results

Headline: an independent effect check catches silent wrong actions that the screen misses, and only as far as it can read. The paper's central claim is a safety one, and we measure it end to end through the live replayer. Ten persistence scenarios, seven fault classes plus an accepted-write control and two delivery controls, are each driven nine times per arm through the real governed path: the replayer's actuator

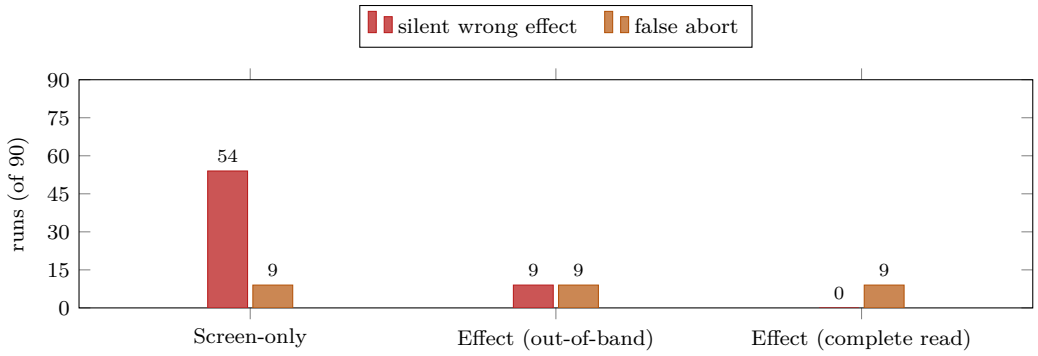

Figure 4: Same 90 injected-fault runs, three verification policies, measured end to end through the live replayer and judged by an independent ground truth. Screen-only verification silently accepted 54 wrong persisted effects; an out-of-band effect oracle that reads the record back cut that to 9 (one uncovered collateral-write class); a complete system-of-record read path drove it to 0. False aborts are a constant 9, a safe halt on an unknown-outcome commit. This is the failure mode general computer-use agents and selector RPA do not measure.

issues an HTTP write to an on-disk system of record, and every run is judged three genuinely independent ways (Section 5). Judged by the *screen*, the same signal an agent or a script trusts, replay silently accepted a wrong persisted effect on 54 of 90 fault runs (60%). An out-of-band effect verifier that reads the record back over a different endpoint than the write drove silent acceptance down to 9 of 90 (10%). Extending that verifier to a complete system-of-record read path, one that covers every mutable surface the action can touch, drove it to 0 of 90 (Figure 4). The residual 9 of 90 is the honest mechanism finding: they are a single fault class, a collateral write to a billing surface the out-of-band oracle's read path never covered, so the guarantee is exactly as strong as the oracle's read coverage. False aborts are a constant 9 of 90 in all three arms, a conservative halt on a commit whose outcome is unknown to the client, so they do not distinguish the oracles. This end-to-end measurement supersedes an earlier in-process fault study whose effect and ground-truth checks read the same object, which made its zero circular by construction; the numbers here come from the real replayer with three genuinely distinct write, read-back, and ground-truth paths.

Per class, the screen misses five of seven fault types. In a companion transactional study at the same persistence boundary, screen-only verification silently mishandled five of seven injected fault classes: partial save, duplicate submission, optimistic UI followed by rejection, stale overwrite, and double delivery. There were 10 consistent repeats per class. When the step declared the expected effect and a system-of-record verifier was configured, those five classes refuted and halted. This improvement has two non-optional preconditions: the effect must be authored and the verifier deployed.

Generalization: the same silent-wrong gap on a second, non-healthcare domain. The effect-verification result is not specific to clinical records. On MockLoan, a loan-disbursement authorization workflow with its own ledger, screen-only verification silently accepted 21 of 33 wrong ledger effects (64%) while over-halting none; reading the true effect from the ledger through an independent verifier drove silent wrong effects to 0 of 33, at a cost of 6 of 33 safe over-halts. Broken out by class, screen-only verification again silently mishandled five of seven injected fault classes. A decoy-loan task shows the identity gate doing its job on the money-movement step: a blind write to a same-name loan is a silent wrong effect under screen-only verification and a caught, safe over-halt under effect verification. This is the same mechanism as the healthcare study, on a different system of record.

Compiled replay removes per-run reasoning on demonstrated tasks. Table 2 shows that, on these two already-demonstrated tasks, compiled replay completed every measured run with lower median latency and no model calls. The unequal, small samples do not establish superior reliability. The correct inference is narrower: repeated reasoning was unnecessary for the healthy executions measured here.

Table 2: Bounded repeated-workflow comparison. Model API costs are estimates derived from recorded token usage and artifact-declared list prices; they exclude authoring and operations.

Task	Arm	Success	n	Median (s)	Model cost/run
OpenEMR	Compiled	20/20	20	39.2	$0.00
OpenEMR	Agent	10/10	10	70.4	$0.55
MockMed	Compiled	100/100	100	4.9	$0.00
MockMed	Agent	20/20	20	37.5	$0.27

Table 3: Scoped substrate qualifications. Each row is one named task and environment, not an arbitrary-application support claim.

Substrate	Effect trials	Refusal controls	Independent oracle
Windows UIA	3/3	stale 3/3; ambiguous 3/3	SQLite row state
Native macOS	3/3	ambiguous 1/1	exact file bytes
Network RDP	3/3	readiness/timeout gate	guest-tools file readback

Repair under drift: the compiler heals what breaks the agent's day. The comparison above is the healthy path. To exercise the "govern every repair" half of the thesis, we forced the MockMed interface to re-render with a dark palette, invalidating every recorded template crop. As a single observation per arm, compiled replay self-healed in 9.7 s with 8 target repairs and zero model calls, while the same computer-use agent under the same drift took 87.4 s and $0.63 across 24 model calls. One run is not a reliability claim, but it shows the mechanism working end to end: the resolution ladder re-resolved changed targets deterministically and logged each as a reviewable patch rather than re-reasoning the whole task.

Breadth: weak postconditions can still emit a green report. On the one-run public-web corpus, all 29 recordings compiled; 17 replays reached a verified success, 10 halted safely, and 2 reported success while the external oracle disagreed. Both silent wrong actions were vacuous successes under an environment blocker rather than a harmful persisted write, but they demonstrate that weak postconditions can still yield a green report. Because each app ran once and the corpus excludes authenticated enterprise and desktop applications, these counts are failure discovery, not a generalization rate estimate.

Identity: zero false accepts, honestly at the cost of availability. The integrated identity ladder recorded zero false accepts in every tested configuration. Structured browser identity also had zero over-halts on 14 correct homonym cases. Pure-pixel configurations halted on all correct confusable cases: zero false accepts at 100% over-halt. This is the intended safety/availability disclosure, not evidence that pixel-only identity is ready for dense clinical lists.

Substrate reach: the same governed semantics across four backends. The governed contract is not browser-specific. Table 3 qualifies three additional substrates on named tasks, each with an independent effect oracle and refusal controls.

The Windows UIA counted matrix recorded 12 native structural-action receipts, three independently confirmed note effects, three stale-target refusals, three ambiguous-target refusals, zero silent incorrect successes, zero over-halts, and zero model calls. The native receipts establish delivery to a unique refreshed fingerprint; the SQLite oracle establishes the business effect.

On macOS, all three isolated TextEdit replace-and-save trials matched the exact expected file bytes, and the two-window control refused without changing either file. There were zero silent incorrect successes and zero over-halts. We retain the original failed batch classification and use its separate adjudication only for these action-effect and refusal observations.

The network RDP batch completed three of three Windows Run-dialog unique-file tasks with exact guest-tools readback. Trial latencies were 51.845, 10.467, and 7.477 seconds, with zero failures, zero silent incorrect successes, zero over-halts, and zero model calls. Snapshot cleanup restored the pre-run inventory

and suspended base state. Together these studies qualify the named substrate mechanisms; they do not transfer to other applications, environments, or Citrix.

7 Limitations and Threats to Validity

The evidence does not establish long-term reliability, maintenance effort, total cost of ownership, multi-tenant scale, or safety in clinical production. OpenEMR uses a shared public demonstration site and a small sample. MockMed is a fixture, and the second-domain lending study is likewise a bounded mechanism study (eleven tasks, three trials each) on a purpose-built MockLoan ledger, not a production lending system. The breadth corpus has one trial per application and selection bias toward public no-auth browser apps. No published study covers weeks of natural drift or a representative distribution of customer workflows.

Identity is only checked on armed steps. Risk classification uses keywords and can miss an icon-only or indirect write. Visual postconditions do not verify a system-of-record effect. Effect declarations and verifiers require deployment work. Optional model rescue can false-rescue an ambiguous state, and measured grounding is weak on dense lists. Human review can catch contextual sanitation errors but is not proof of de-identification.

The substrate qualifications cover one Windows UIA workflow and VM snapshot, one macOS host and TextEdit task, and one RDP task and snapshot. They establish working observation, actuation, refusal, and effect-oracle mechanisms in those environments, not reliability across application families or deployment fleets. Remote display introduces compression, scaling, latency, coordinate, lock-screen, credential, input-acceptance, and identity-availability variables. No measured result covers Citrix ICA/HDX, so RDP evidence does not transfer to a Citrix deployment. Citrix is a pixel-only remote-display environment in the same class as the measured RDP backend, and the resolution ladder and identity gate are substrate-independent by construction, but the paper reports no ICA/HDX measurement. A concrete validation would record and compile a workflow inside a real Citrix Workspace session, replay it through the vision-only rungs of the ladder over the ICA/HDX channel, and confirm the identity gate refuses ambiguous targets and that an out-of-band effect oracle catches an injected transactional fault. That study is not reported here.

Effect verification also has a substrate-dependent reach that the RDP row can obscure. The strong oracle is only available when the system of record can be read out of band, through a database, API, or file interface independent of the substrate that carried the write. The RDP qualification reads the persisted file through guest tools, an out-of-band channel that a locked-down Citrix session may not expose. Where no such read path exists, the transactional strong oracle is unavailable; a same-application re-read cannot catch a partial, duplicate, stale, or lost-update write, and on such substrates safety rests on the identity gate and halt-on-ambiguity rather than on effect verification. Even where a read path exists, an out-of-band effect oracle is only as strong as its read coverage: in the end-to-end study a collateral write to a billing surface the oracle did not read slipped through (9 of 90) until the read path was extended to every mutable surface, so effect verification guarantees only what it reads.

Cost values use recorded provider usage and then-current price metadata. They do not include recording, compilation, operator review, exception handling, infrastructure, support, or maintenance. Provider prices and models change.

Finally, the authors develop the evaluated system. Raw artifacts and scripts reduce but do not eliminate experimenter bias. Independent replication and pre-registered longitudinal workloads remain necessary.

8 Reproducibility and Artifacts

The source repository and versioned evaluation artifacts are published together [10] at `https://github.com/OpenAdaptAI/openadapt-flow`. The paper package includes a script that reads benchmark JSON and fails if a headline constant drifts. Raw run rows, aggregate JSON, task definitions, benchmark prose, failure taxonomies, and selected run reports are versioned with the implementation.

Evidence carries one of four labels:

- **CI-reproducible**: fixture and environment can run in automated CI, subject to browser and dependency availability.

- **Field**: real third-party target, but shared mutable state or credentials prevent full CI reproduction.
- **Fixture/fault injection**: deterministic mechanism study, not a population estimate.
- **Descriptive**: useful for discovering failures, but missing the repeated trials needed for comparative inference.

Before submission, the release candidate must re-run promoted experiments, archive its exact commit and environment, record task-level oracles, and check that no application artifact contains sensitive or prohibited data. The repository's `paper/ARTIFACT_CHECKLIST.md` is the submission gate.

9 Conclusion

Repeated GUI work does not require a general model to rediscover the same plan on every run. A demonstration compiler can retain the useful flexibility of multi-modal target resolution while recovering deterministic execution, inspectable programs, and bounded repair. The critical distinction is not whether a task completed, but whether identity and external effects were verified and whether the system can refuse when evidence is insufficient.

The results establish the compiler/runtime thesis on bounded browser workflows and show that the same governed semantics can drive named Windows UIA, native macOS, and network RDP tasks with independently checked effects and explicit refusal. The next evidence target is longitudinal, repeated operation of one consequential customer workflow with authoring time, intervention rate, silent incorrect success, over-halt, recovery time, and system-of-record outcomes measured across natural interface change.

References

[1] Anthropic. Developing a computer use model. https://www.anthropic.com/news/developing-computer-use, 2024. Accessed 2026.

[2] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2015.

[3] Sarah E. Chasins, Maria Mueller, and Rastislav Bodík. Rousillon: Scraping distributed hierarchical web data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, pages 963–975. ACM, 2018.

[4] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

[5] Allen Cypher, editor. *Watch What I Do: Programming by Demonstration*. MIT Press, 1993.

[6] Alexandre Drouin et al. Workarena: How capable are web agents at solving common knowledge work tasks? *Proceedings of Machine Learning Research*, 2024.

[7] Jim Gray. The transaction concept: Virtues and limitations. In *Proceedings of the 7th International Conference on Very Large Data Bases*, pages 144–154, 1981.

[8] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 317–330. ACM, 2011.

[9] Klaus Havelund and Grigore Roşu. Synthesizing monitors for safety properties. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.

[10] OpenAdapt Contributors. openadapt-flow: Demonstration compiler and governed gui runtime. https://github.com/OpenAdaptAI/openadapt-flow, 2026. Software and evaluation artifacts; access date to be fixed at submission.

[11] Yujia Qin et al. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

[12] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[13] Wil M. P. van der Aalst, Martin Bichler, and Armin Heinzl. Robotic process automation. *Business & Information Systems Engineering*, 60(4):269–272, 2018.

[14] Tianbao Xie et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37, 2024.

[15] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. Sikuli: Using gui screenshots for search and automation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, pages 183–192. ACM, 2009.

[16] Shuyan Zhou et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.