BOUNDARYLOOP: FROM FAILURE EVIDENCE TO PERSISTENT WEB CAPABILITIES

Anonymous authors
Paper under double-blind review

ABSTRACT

An episodic WebAgent repair becomes a reusable capability when its executable behavior survives new task parameters and later runs. We study how failure evidence supports that transition. BoundaryLoop turns failed WebAgent episodes into browser-tested instance workflows, compares repairs to extract parameterized family procedures, and admits those procedures to executable memory after transfer and boundary validation. The WebAgent supplies task, action, DOM, and browser-state evidence; an auxiliary CodeAgent synthesizes and tests the correction. In a frozen WebArena Shopping Admin comparison with 84 runs per condition, the Base WebAgent records 2/84 execution successes and 0/84 clean completions. Static repair passes the evaluator in 59/84 runs but has parameter pollution in 84/84, whereas promoted family workflows complete 84/84 runs without recorded pollution or drift. Compact aligned evidence matches the 6/24 accepted-repair yield of full raw evidence with 12.7% fewer input tokens. A frozen family procedure transfers to 3/3 original instances excluded from abstraction and 5/5 post-freeze generated instances. In a separate later-runtime study, stored workflows pass 33/33 runs versus 0/33 for the matched BaseAgent. These results locate capability acquisition in cross-instance invariance and validated reuse.

1 INTRODUCTION

Web agents must turn a natural-language goal into actions on an interface whose controls and state can change between observations. An agent may select an order status yet leave the filter unapplied, or reach the right grid and answer from the wrong rows. In benchmarks such as WebArena and BrowserGym, a plausible trajectory and a verified task state are therefore different objects (Zhou et al., 2023b; Le Sellier De Chezelles et al., 2024).

The distinction persists after a successful retry. A repair might restore one failed episode by selecting the right control or replaying a useful sequence, but the sequence may still encode an entity, date, or browser state from that episode. When the task changes, those hidden bindings turn a seemingly successful repair into another failure. We therefore ask: *when does repairing a WebAgent failure constitute acquiring a persistent executable capability?* The object we seek is a task-conditioned procedure whose behavior survives parameter changes and whose applicability can be tested before reuse.

BoundaryLoop treats acquisition as a transition: *repair* $\rightarrow$ *cross-instance invariance* $\rightarrow$ *promotion gates* $\rightarrow$ *persistent executable capability*. The online WebAgent records the task, actions, grounded controls, and observed browser states from a failed episode. An OpenCode-based CodeAgent uses that record and same-family demonstrations to synthesize a complete instance workflow, which real-browser replay accepts or returns as a structured counterexample. Comparing successful instance workflows reveals stable operations and changing arguments; parameterization makes the latter explicit. Transfer, regression, non-target, empty-result, and browser-boundary tests then govern admission to a Registry. The model weights stay fixed as the executable repertoire grows (Figure 1).

Three observations motivate this separation of stages. In a frozen 84-run-per-condition comparison, static repair satisfies the evaluator in 59 runs even though every run carries a stale parameter, while family workflows satisfy the clean task contract in all 84. Failure-conditioned synthesis succeeds where task-only synthesis does not in a controlled evidence study. A frozen family workflow trans-

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.

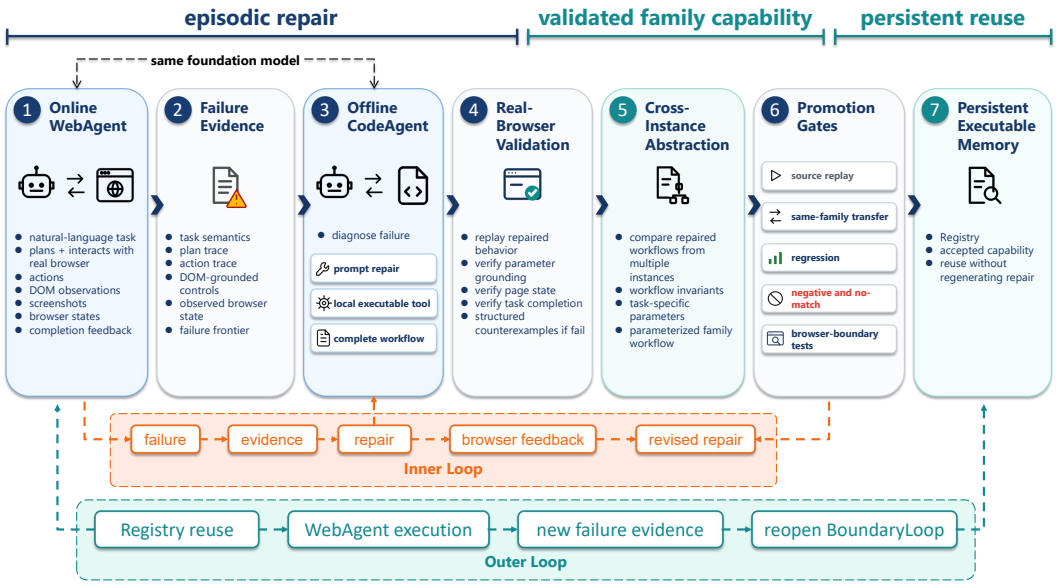

Figure 1: BoundaryLoop converts repair into capability through two nested loops. Browser replay returns counterexamples to instance repair; cross-instance abstraction and promotion gates govern admission to persistent memory. Later capability failures reopen the repair loop.

fers to new parameter combinations; separate later-runtime cohorts show stored workflows remain callable in AgentLab.

Our contributions are:

- an operational criterion for Web capability acquisition based on parameterized cross-instance behavior, boundary validation, and later executable reuse;
- a two-role repair loop in which grounded failure evidence guides executable synthesis and browser counterexamples guide revision;
- experiments that separate instance recovery from family capability through polluted static repair, evidence-conditioned synthesis, frozen transfer, gate rejection, and persistent reuse.

2 Related Work

Web grounding and planning. Mind2Web supplies human interaction trajectories, while WebAgent and instruction-tuned navigation develop decomposition and action generation from Web observations (Deng et al., 2023; Gur et al., 2023; Furuta et al., 2023). SeeAct, WebVoyager, SeeClick, and UGround improve the grounding of actions in visual or structural page evidence; SteP and WebDreamer further organize decisions through specialized policies or predicted outcomes (Zheng et al., 2024a; He et al., 2024; Cheng et al., 2024; Gou et al., 2024; Sodhi et al., 2023; Gu et al., 2024). These methods address interaction within an episode. We study how an observed failure becomes an executable correction that survives a change of task instance.

Reflection and experiential adaptation. ReAct, Self-Refine, Reflexion, CRITIC, ExpeL, LATS, and AdaPlanner show how feedback, experience, and search can revise subsequent decisions (Yao et al., 2023; Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2023; Zhao et al., 2023; Zhou et al., 2023a; Sun et al., 2023). For GUI tasks, Agent S retrieves experience for planning and CUADebug localizes a consequential step from traces and visual states (Agashe et al., 2024; Zhang et al., 2026a). A corrected next attempt is useful evidence; whether it defines a reusable procedure depends on parameter binding and cross-instance replay. BoundaryLoop makes that test part of the repair lifecycle.

Workflow memory and skill synthesis. Agent Workflow Memory induces reusable workflows from demonstrations; Voyager builds an executable skill library in Minecraft, while SkillWeaver discovers and practices reusable Web APIs (Wang et al., 2024c; 2023; Zheng et al., 2025). SCAFFOLD abstracts successful Web trajectories into recursive parametric skills; ContractSkill gives a draft skill explicit structure for verification and local repair; Executable Agentic Memory organizes action groups for later GUI execution (He et al., 2026; Lu et al., 2026; Qin et al., 2026). CREATOR, Large Language Models as Tool Makers, and CodeAct further establish code as an agent action medium (Qian et al., 2023; Cai et al., 2023; Wang et al., 2024b). BoundaryLoop starts instead from a failed WebAgent episode. Its admission question is whether a correction inferred from that failure preserves control semantics when task parameters change, including at empty-result and browser-state boundaries.

Capability evaluation and self-improvement. WebArena, WorkArena, and BrowserGym provide stateful environments for measuring interaction (Zhou et al., 2023b; Drouin et al., 2024; Le Sellier De Chezelles et al., 2024); τ-bench and ToolSandbox expose multi-turn state and tool side effects (Yao et al., 2024; Lu et al., 2024a). WebRL adapts policy parameters through generated curricula (Qi et al., 2024), whereas BoundaryLoop holds model weights fixed and grows an external executable repertoire. AgentDojo and ToolEmu evaluate adversarial or unsafe behavior in tool-using agents (Debenedetti et al., 2024; Ruan et al., 2023). We measure task completion together with parameter binding, browser state, and applicability boundaries. Appendix D situates this criterion in the broader GUI and agent literature.

3 Problem Formulation

Let a task be $x = (g, f, \theta)$, where g is a user goal, f a task family, and θ its instance parameters. A family groups tasks with a shared template and completion semantics; parameters may specify a date interval, product, quantity, or order status. An interaction produces a trajectory $\tau = (o_0, a_0, o_1, \ldots, a_{T-1}, o_T)$, where observations include available DOM/control evidence and browser state.

Instance repair and family capability. An instance repair w_x is an executable workflow constructed to recover task x. A family workflow $W_f(\theta)$ exposes explicit parameter slots and reusable control logic. Successful execution of w_x establishes recovery of its source instance. The stronger claim for W_f requires the same control logic to bind new parameters, complete instances excluded from abstraction, preserve previously successful behavior, and reject tasks outside its applicability contract.

Observable completion. Task acceptance requires a conjunction of observable conditions:

$$A(x, W) = X \wedge G \wedge E \wedge C \wedge \neg P \wedge \neg D, \tag{1}$$

where X is valid execution, G verified task-parameter grounding, E task-relevant supporting evidence, C the required terminal state or submitted grounded answer, P parameter pollution, and D unresolved environment/data mismatch. Evaluator output is recorded alongside this contract. When it conflicts with rendered evidence, an independent snapshot-based check resolves whether the page state supports the submitted answer.

Evidence sufficiency and capability acquisition. Let $\mathcal{E}_x$ denote the observable failure record and $z_x = \alpha(\mathcal{E}_x)$ an evidence view produced by selection and alignment. Under fixed model and repair budgets, z_x is *repair-sufficient* for an instance only if the synthesized workflow $w_x = \mathcal{C}(x, z_x)$ satisfies Eq. 1 in browser replay. Capability acquisition is a stronger transition:

$$\{(x_i, w_{x_i})\}_{i \in \mathcal{D}_f} \xrightarrow{\text{cross-instance abstraction}} W_f(\theta) \xrightarrow{\text{frozen transfer and reuse}} \mathcal{M}, \tag{2}$$

where $\mathcal{D}_f$ contains validated development instances and $\mathcal{M}$ is persistent executable memory. The transition separates three empirical questions: whether evidence enables an instance repair, whether repaired instances expose a stable family invariant, and whether the frozen invariant remains executable on later tasks. Evidence views are compared by the repairs they support and the context they consume.

Two distinct boundaries. The *failure frontier* is the earliest observed failed operation or state-contract contradiction in a replay. It localizes the next executable correction. The *capability boundary* is the tested set of task conditions under which a frozen workflow satisfies its contract. Repair moves the failure frontier; transfer and promotion tests establish the capability boundary.

4 BOUNDARYLOOP

BoundaryLoop links instance repair to task-family reuse through two loops (Figure 1). Browser replay tests each repair and returns counterexamples; cross-instance abstraction and promotion gates determine which workflows enter persistent memory.

4.1 CROSS-VIEW FAILURE EVIDENCE

The repair package links four observable views: task semantics and parameters; actions and recorded plans; DOM-grounded controls; and browser states before and after the operation. Each observation carries its origin in the failed episode, diagnostic inspection, or repair replay. The resulting record preserves which state the WebAgent actually observed and which evidence was collected later for diagnosis.

Alignment is implemented as an explicit contract across views. For an order-status query, task language is normalized to a canonical status and mapped to the website's option label and value. Validation then connects the selected GUI state, applied filter indicator, observed GUI request, and returned evidence. Semantically equivalent labels may use different strings; the contract compares their website-level meaning.

$$s_{\text{task}} \equiv s_{\text{GUI}} \equiv s_{\text{applied}} \equiv s_{\text{request}} \equiv s_{\text{evidence}}, \tag{3}$$

where $\equiv$ denotes semantic equivalence under the website mapping. A disagreement identifies a concrete repair target and the first state at which the intended parameter diverged from browser behavior. Figure 2 traces this divergence in an order-status example.

The package also contains same-family demonstrations, a deterministic diagnostic summary, and previous validation feedback. Demonstrations supply candidate navigation and control structure, while the target task supplies its own entities and parameters. This division lets the CodeAgent use successful peer trajectories without copying their answers or parameter values into the repair. Private evaluation data and future transfer tasks remain separated from synthesis.

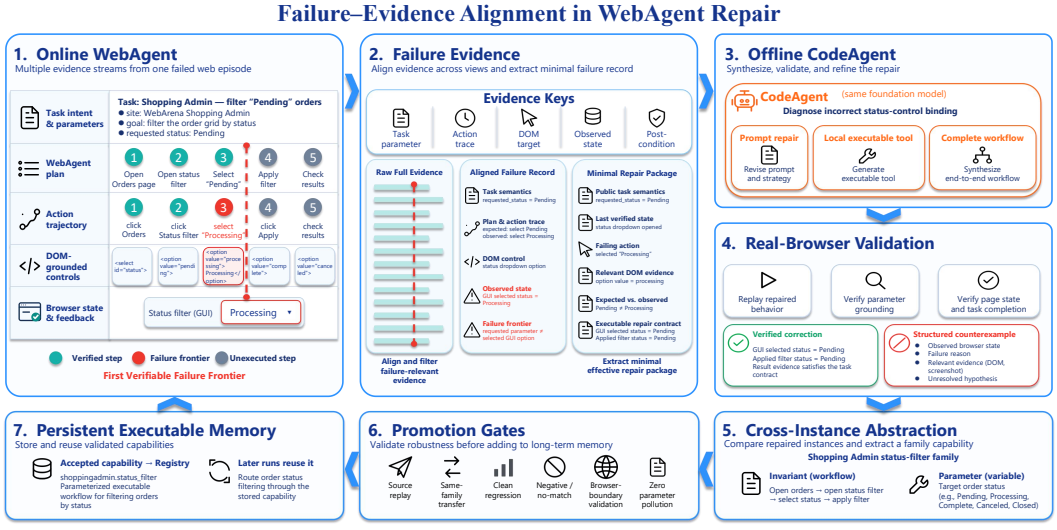

Figure 2: Cross-view alignment localizes an order-status failure to the selection action. The repair contract specifies the browser state that replay must verify.

4.2 WORKFLOW-FIRST EXECUTABLE REPAIR

BoundaryLoop first asks the CodeAgent to recover one failed instance. The output is a complete executable workflow: ordered browser operations with explicit task parameters, preconditions, post-conditions, and an evidence handoff. Treating this artifact as an executable hypothesis makes the failure concrete enough to test, while leaving the question of family-level reuse to later abstraction and promotion.

The controller validates structure and permitted operations, then executes the workflow in an isolated browser session. A second stage runs a complete WebAgent episode with the workflow available, testing invocation, control handoff, evidence return, and terminal completion. Browser feedback therefore evaluates both the generated procedure and its integration with the online agent. Shared execution-layer defects can be repaired through a separate allow-listed source-patch path.

Each repair attempt records its input manifest, model route, output artifact, static checks, and replay evidence. Appendix A consolidates cohort identity, artifact provenance, and attribution boundaries.

4.3 COUNTEREXAMPLE-GUIDED ITERATION

At iteration k, validation returns feedback F_k containing the failed operation, observed controls, parameter/state discrepancies, and completion status. The next artifact is generated conditionally:

$$w_{k+1} = \mathcal{C}_\phi(x, \mathcal{E}, \mathcal{D}_f, w_k, F_k), \tag{4}$$

where $\mathcal{E}$ is the failure evidence and $\mathcal{D}_f$ the allowed same-family demonstrations. The next workflow therefore conditions on both the observed failure frontier and the behavior already verified in replay.

A verified-prefix check protects earlier successful workflow steps while allowing later page evidence to revoke a step whose parameter semantics are wrong. The next repair can focus on the failed frontier instead of rewriting unrelated behavior. Iterations continue until the workflow satisfies the browser contract or exhausts the cohort's repair budget. Network, authentication, and server failures are separated from semantic repair feedback.

Browser state is reset and verified between episodes because persistent filters can change later results even in read-only tasks. Failed replays remain part of the evidence record, and failures of stored workflows create new evolution requests that reopen the loop.

4.4 CROSS-INSTANCE ABSTRACTION AND PERSISTENT ACCEPTANCE

Successful same-family workflows expose two structures: invariant browser operations and instance-specific arguments. Parameterization replaces changing values with typed slots while retaining navigation, control binding, and state checks. In the staged family study, the CodeAgent synthesizes instance repairs and a deterministic abstraction step assembles the parameterized family workflow; these are distinct provenance stages. The family workflow is replayed on development instances and then frozen before transfer evaluation. This freeze separates abstraction from the tasks used to test it.

Promotion gates cover source replay, same-family development, frozen transfer, original-success regression, non-target behavior, no-match behavior, and browser-level boundaries. Together they verify that the invariant executes, the parameter slots bind the requested values, and the applicability contract controls invocation. Non-target tasks test routing precision; empty-result tasks require the intended filter to be visibly applied before an empty answer is accepted; browser boundaries inspect the live control state.

Accepted family workflows enter persistent executable memory, implemented by a versioned Registry with applicability and parameter contracts. Runtime routing selects the workflow, binds the new task parameters, and returns checked evidence to the WebAgent. Each successful invocation reuses the promoted capability without regenerating a tool.

5 EXPERIMENTAL DESIGN

The acquisition sequence in Eq. 2 gives each experiment a distinct question. Evidence-view comparisons ask whether a failed episode supplies information for instance repair. A staged family study asks whether independently repaired instances reveal a procedure that survives new parameters. Gate audits test whether a source-successful procedure deserves admission, while repeated runs and later Registry invocation test the behavior of admitted workflows. Appendix A records each cohort's configuration and denominator.

The main controlled cohorts use WebArena Shopping Admin. Original tasks provide failure and development trajectories; snapshot-instantiated same-family tasks provide parameter variations with separate public instructions and private evaluation data. A separate WebArena Shopping storefront deployment tests the workflow and Registry interface on another website, using public browser/URL contracts and an isolated Registry. The frozen 84-run comparison uses the earlier custom GUI executor, while the later evidence, family-transfer, and reuse studies use BrowserGym/AgentLab GenericAgent with Playwright. The offline repair role uses an OpenCode controller. Within each comparison, model route, browser stack, and action budget are fixed.

For each run we retain execution, evaluator acceptance, page evidence, verified completion, parameter pollution, and environment drift, along with actions, online calls, tokens, and wall time when available. Repetitions measure stability; frozen transfer and later invocation separate generalization from one-off recovery.

6 RESULTS

Figure 3 follows the acquisition sequence. We begin with the gap between apparent repair and clean execution, then examine the evidence, abstraction, admission, and reuse that close it.

6.1 REPAIR SUCCESS IS NOT CAPABILITY ACQUISITION

The frozen-runtime study evaluated 28 database-derived tasks across review-term counting (family 288), order-attribute queries (366), and payment aggregation (367). Each configuration ran three valid repetitions, yielding 84 episodes. Besides the base WebAgent and a parameter-closure control, it compares two executable interventions: an accumulated static repair without persistent-memory routing, and the parameterized family workflows admitted to persistent memory.

Table 1: The central repair–capability contrast: 28 tasks $\times$ 3 repetitions per condition. "Clean" combines execution and page evidence with zero parameter pollution and environment drift. †Diagnostic static-repair control; *clean capability result.

Condition	Execution	Clean	Evaluator	Evidence	Pollution	Drift	Actions	Online calls
Base WebAgent	2/84	0/84	2/84	83/84	11/84	49/84	662	743
Parameter closure	1/84	0/84	2/84	83/84	8/84	49/84	661	743
Static repair†	0/84	0/84	59/84	84/84	84/84	1/84	188	1
Family workflow*	**84/84**	**84/84**	**84/84**	**84/84**	**0/84**	**0/84**	**186**	**0**

Table 1 separates execution, answer acceptance, and task-grounded completion. The Base WebAgent records 2/84 execution successes and 0/84 clean runs. Static repair retrieves page evidence in all 84 runs and satisfies the evaluator in 59, yet every run carries a parameter from the repair context. The promoted family workflows bind each new task cleanly in 84/84 runs, with zero recorded pollution or drift. Their browser macro-action count is similar to static repair (186 versus 188); the decisive difference is whether those actions follow the new task's parameters. In this executor, stored workflows also reduce online LLM calls from 743 to zero. Synthesis and validation incur separate offline costs.

6.2 FAILURE EVIDENCE PROVIDES THE REPAIR SUBSTRATE

We ask whether failure observations help the CodeAgent construct an executable repair. Across seven instances, early paired source acceptance is 0/8 for task-only input and 2/8 with failure evi-

dence. A larger study compares eight evidence views on eight drift-free failures, with three repeats each (192 sequences). Model, 20-step replay, three repair rounds, workflow contract, and disabled Registry are fixed.

Table 2: Failure evidence and repair yield. The early paired study reports source acceptance; the larger study reports accepted operational WebAgent replay. Mean tokens include repeated CodeAgent rounds. *Matches raw-full yield with 12.7% fewer input tokens.

Cohort	Repair input	N	Pass	Rate	Mean tokens	Observed role
A. Early paired source-repair study						
Paired source	Task only	8	0	0.0%	–	instruction-only control
Paired source	Failure-conditioned	8	2	25.0%	–	+2 source recoveries
B. Large evidence-view study						
Evidence view	Task only	24	0	0.0%	8.2k	instruction-only control
Evidence view	Trajectory only	24	7	29.2%	18.3k	highest observed yield
Evidence view	Raw full	24	6	25.0%	38.0k	complete-context reference
Evidence view	Aligned full	24	5	20.8%	37.7k	explicit cross-view contract
Evidence view	**Aligned minimal***	24	**6**	**25.0%**	**33.1k**	raw-full yield, 12.7% fewer tokens
Evidence view	No action trace	24	7	29.2%	30.3k	execution scaffold retained
Evidence view	No DOM/state	24	5	20.8%	23.8k	control evidence removed
Evidence view	Shuffled alignment	24	5	20.8%	39.8k	correspondence disrupted

Across failure-conditioned views, 41/168 replays pass execution, page evidence, completion, pollution, and drift checks, versus 0/24 task-only. The failure record supplies an observed discrepancy to correct.

Aligned-minimal evidence retains public task semantics, the failure frontier, three recent actions, and the browser state. It matches raw-full at 6/24 accepted replays with 12.7% fewer input tokens. Outcomes agree in 22/24 repeated runs; each view alone wins once. Accepted repairs arise on three of eight task IDs, so repeats are described rather than treated as independent tasks for inference. Trajectory-only and action-ablated views also yield repairs: the common substrate is failure-conditioned information, not a uniquely superior format.

6.3 Cross-instance invariance turns repair into a family procedure

The family-368 study follows repaired instances into a reusable procedure. Frozen BaseAgent episodes for tasks 184 and 186 each exhaust 20 online calls without satisfying the product-quantity contract. Across six evidence settings and the two instances, the CodeAgent produces 10/12 accepted real-browser instance repairs. Comparing the successful workflows reveals a shared sequence: navigate to the product grid, bind the target product and quantity constraint, apply the filter, and return page evidence. A deterministic abstraction step assembles these instance repairs into one workflow with typed bounds for scalar and interval requests.

Table 3: Family-368 instance-to-family progression. Rows are sequential validation stages, not pooled trials. "Clean" requires accepted execution with zero pollution and drift. ‡Source-repair stage; *frozen transfer set excluded from synthesis.

Stage	Units	Pass	Clean	Actions/pass	Calls/pass	What changes
Base source episodes	2 tasks	0/2	0/2	–	20	observed capability gap
CodeAgent instance repair‡	12 conditions	10/12	10/12	1	0	executable source recovery
Frozen source replay	2 tasks	2/2	2/2	1	0	artifact identity fixed
Same-family original transfer	3 tasks	3/3	3/3	1	0	instances excluded from abstraction
Post-freeze generated transfer*	**5 tasks**	**5/5**	**5/5**	**1**	**0**	parameters and empty-result boundary
Persistent-memory reuse	1 smoke	1/1	1/1	1	0	routed without regeneration

The frozen workflow passes both repaired source tasks, 3/3 same-family original instances excluded from abstraction, and 5/5 post-freeze generated tasks (Table 3). The generated set includes new parameters and an empty-result boundary, where the target filter must be visibly active before an empty answer is accepted. Transfer runs satisfy the page-grounded operational contract with zero recorded

pollution or drift. A later isolated memory route invokes the same artifact without regeneration. The accumulated Admin library contains 10 task families, while Table 3 tests this particular abstraction.

A matched-run study on five positive tasks preserves 5/5 page-grounded completions while reducing browser actions by 64.3%, online calls by 73.7%, tokens by 68.3%, and wall time by 42.4% (Table 8 in Appendix A.4).

6.4 Promotion gates reject instance-bound repairs

Source recovery exposes rather than resolves the next capability question. In two sealed repair lineages, both workflows recover their source tasks, but their subsequent tests locate different control-binding failures. Family 366 fails development validation when it cannot bind the visible `Pending` option. Family 367 passes development, regression, and routing checks, yet a no-match request for `Suspected Fraud` leaves the browser on `Pending`. The former is rejected at development, the latter at the browser-boundary gate; neither enters this lineage's persistent memory. Appendix A.2 records their provenance.

6.5 Promotion-gate ablation

The consequence of omitting a gate is visible in a policy audit of the same three historical candidate records (families 249, 250, and 253). Source-only and held-out-only rules each admit all three, including family 249, while the recorded full gate admits two. Removing the browser-boundary gate also admits all three. The boundary test separates the rejected instance-bound candidate from the two admitted workflows in this cohort (Table 9).

6.6 Promoted workflows persist across runtimes

A later experiment invokes stored workflows through AgentLab on 11 eligible generated tasks: three review-product tasks, five monthly-order-report tasks, and three payment-aggregate tasks. Each task is repeated three times under matched model routes, browser stacks, task inputs, snapshots, and 30-step budgets.

Table 4: Persistent reuse under a later AgentLab runtime. Each task is repeated three times. "Clean" denotes accepted execution with zero recorded pollution and drift; call reduction is computed within each family. *Aggregate across the listed families.

Family	Tasks × repeats	Base pass	Workflow pass	Clean	Base calls	Workflow calls	Call reduction
Review product (250)	3×3	0/9	9/9	9/9	85	9	89.4%
Monthly orders (270)	5×3	0/15	15/15	15/15	129	15	88.4%
Payment aggregate (367)	3×3	0/9	9/9	9/9	33	9	72.7%
Total*	$\mathbf{11 \times 3}$	**0/33**	**33/33**	**33/33**	**247**	**33**	**86.6%**

Stored workflows pass 33/33 episodes, versus 0/33 for the matched BaseAgent, with zero recorded pollution or drift and an 86.6% reduction in online calls (Table 4). Later invocation thus tests a property source replay cannot: the admitted executable object remains callable in another WebAgent runtime without regenerating the workflow. A separate Shopping storefront deployment verifies live parameter binding and isolated-Registry routing for three operational family entries; its public-contract results appear in Appendix A.1.

7 Mechanism and Implications

The first transformation makes a failed interaction actionable. Task parameters, recorded actions, DOM controls, and observed browser states identify where intended and realized behavior first diverge; browser replay then tests a correction at that frontier. The evidence-view comparison links such failure-conditioned input to accepted instance repairs. Compact alignment carries the same observed yield as raw-full evidence with less context, without requiring a privileged evidence format.

The second transformation separates an instance trace from a task-family procedure. Repaired workflows expose the operations that remain stable across tasks and the values that must be rebound;

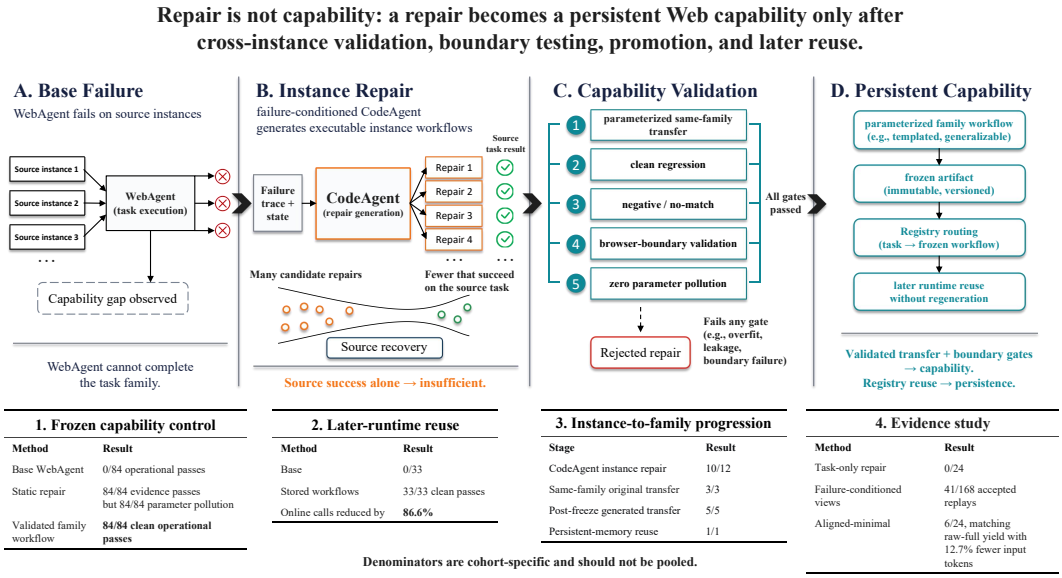

Figure 3: Evidence ladder for capability acquisition. Each stage is measured in a dedicated cohort; the denominators are not pooled.

abstraction turns this pattern into a candidate workflow. Evaluator acceptance can miss stale bindings, as Table 1 shows. Transfer and browser-grounded gates test the candidate's parameter contract before persistence, and later runtime reuse tests whether that contract survives beyond synthesis. Capability growth thus occurs in the executable repertoire rather than the model weights.

The tests distinguish two ways a plausible repair can stop short of capability. Static replay may reach a supported answer while retaining the source instance's parameter; checking the selected and applied browser state exposes this hidden binding. A different candidate may recover its source task yet mishandle a new status or an empty-result case. Here, the boundary test challenges the workflow's applicability rather than its source success. Promotion therefore changes which executable object the agent can call later, rather than merely re-scoring the original episode.

Alignment preserves the causal relation between an intended parameter, the control that should express it, and the browser state that follows. The evidence study asks whether a compact record retains enough of that relation for repair. Browser replay tests the correction, while promotion tests whether the resulting executable contract remains valid on later tasks.

8 Conclusion

BoundaryLoop gives a WebAgent a path from failed interaction to tested, reusable procedure. Failure evidence supports instance repair; cross-instance comparison exposes stable operations and changing parameters; promotion gates test the resulting workflow beyond its source task. The central contrast—59/84 evaluator-successful yet polluted static replays versus 84/84 clean family-workflow runs—shows the difference between recovering an answer and acquiring reusable execution. Frozen transfer and later Registry invocation complete that transition: the workflow remains callable after the original failure is gone.

AI use statement

AI Assistance: Yes, to aid or polish writing: AI tools revised wording and flow. Yes, for retrieval and discovery: they identified relevant papers, whose records were checked against primary sources. Yes, for research ideation or execution: they supported research-question framing, experimental design, repair-workflow implementation, and result auditing. Yes, to draft sections of the paper: they produced prose drafts revised against the recorded evidence.

Ethics statement

Experiments use a controlled benchmark deployment. Credentials, sessions, and private database answers are excluded from repair prompts and manuscript assets. Real administrative deployment would additionally require authorization, least privilege, and safeguards for irreversible actions.

Reproducibility statement

The evidence map and read-only audit script reproduce reported aggregates from retained manifests, configurations, artifact hashes, and replay references. Appendix A identifies cohort boundaries and gate decisions. Public instructions and private evaluation records remain separated.

References

Tamer Abuelsaad, Deepak Akkil, Prasenjit Dey, Ashish Jagmohan, Aditya Vempaty, and Ravi Kokku. Agent-e: From autonomous web navigation to foundational design principles in agentic systems, 2024. URL https://arxiv.org/abs/2407.13032.

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024. URL https://arxiv.org/abs/2410.08164.

Mohamed Aghzal, Gregory J. Stein, and Ziyu Yao. Why do llm-based web agents fail? a hierarchical planning perspective, 2026. URL https://arxiv.org/abs/2603.14248.

Gilles Baechler, Srinivas Sunkara, Maria Wang, Fedir Zubach, Hassan Mansoor, Vincent Etter, Victor Cărbune, Jason Lin, Jindong Chen, and Abhanshu Sharma. Screenai: A vision-language model for ui and infographics understanding. *arXiv preprint arXiv:2402.04615*, 2024. URL https://arxiv.org/abs/2402.04615.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023. URL https://arxiv.org/abs/2305.17126.

Yuxiang Chai, Shunye Tang, Han Xiao, Weifeng Lin, Hanhao Li, Jiayu Zhang, Liang Liu, Pengxiang Zhao, Guangyi Liu, Guozhi Wang, Shuai Ren, Rongduo Han, Haining Zhang, Siyuan Huang, and Hongsheng Li. A3: Android agent arena for mobile gui agents with essential-state procedural evaluation, 2025. URL https://arxiv.org/abs/2501.01149.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*, 2024. URL https://arxiv.org/abs/2401.10935.

Gaole Dai, Shiqi Jiang, Ting Cao, Yuanchun Li, Yuqing Yang, Rui Tan, Mo Li, and Lili Qiu. Advancing mobile gui agents: A verifier-driven approach to practical deployment. In *MobiCom 2026*, 2026. URL https://arxiv.org/abs/2503.15937.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *arXiv preprint arXiv:2406.13352*, 2024. URL https://arxiv.org/abs/2406.13352.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. *arXiv preprint arXiv:2306.06070*, 2023. URL https://arxiv.org/abs/2306.06070.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024. URL https://arxiv.org/abs/2403.07718.

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*, 2023. URL https://arxiv.org/abs/2305.11854.

Divyansh Garg, Shaun VanWeelden, Diego Caples, Andis Draguns, Nikil Ravi, Pranav Putta, Naman Garg, Tomas Abraham, Michael Lara, Federico Lopez, James Liu, Atharva Gundawar, Prannay Hebbar, Youngchul Joo, Jindong Gu, Charles London, Christian Schroeder de Witt, and Sumeet Motwani. Real: Benchmarking autonomous agents on deterministic simulations of real websites, 2025. URL https://arxiv.org/abs/2504.11543.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024. URL https://arxiv.org/abs/2410.05243.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023. URL https://arxiv.org/abs/2305.11738.

Yu Gu, Boyuan Zheng, Boyu Gou, Kai Zhang, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your LLM secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024. URL https://arxiv.org/abs/2411.06559.

Tianchen Guan, Xinlei Lin, Royce Cheng-Yue, Xiangjun Wang, and Shuyan Zhou. Component-bench: Diagnosing component-level failures in computer-use agents, 2026. URL https://arxiv.org/abs/2608.18307.

Linqiang Guo, Wei Liu, Li Gu, Yang Wang, and Tse-Hsun (Peter) Chen. Stepreflect: Structured ui transition reflection for mobile gui agents, 2026. URL https://arxiv.org/abs/2608.05587.

Izzeddin Gur, Hiroki Furuta, Austin V. Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world WebAgent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*, 2023. URL https://arxiv.org/abs/2307.12856.

Bowei He, Xiaokun Zhang, Meng Ding, and Xue Liu. Scaffold: Self-improving web agents via recursive parametric skill abstraction, 2026. URL https://arxiv.org/abs/2609.05511.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024. URL https://arxiv.org/abs/2401.13919.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Hongming Zhang, Tianqing Fang, Zhenzhong Lan, and Dong Yu. Openwebvoyager: Building multimodal web agents via iterative real-world exploration, feedback and optimization. In *ACL 2025*, 2025. URL https://aclanthology.org/2025.acl-long.1336/.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents. *arXiv preprint arXiv:2312.08914*, 2023. URL https://arxiv.org/abs/2312.08914.

Jaekyeom Kim, Dong-Ki Kim, Lajanugen Logeswaran, Sungryull Sohn, and Honglak Lee. Auto-intent: Automated intent discovery and self-exploration for large language model web agents. In *Findings of EMNLP 2024*, 2024. URL https://arxiv.org/abs/2410.22552.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024. URL https://arxiv.org/abs/2401.13649.

Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Ruslan Salakhutdinov, Nicolas Chapados, and Alexandre Lacoste. The BrowserGym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2024. URL https://arxiv.org/abs/2412.05467.

Eric Li and Jim Waldo. Websuite: Systematically evaluating why web agents fail, 2024. URL https://arxiv.org/abs/2406.01623.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use. In *ACM Multimedia 2025*, 2025. URL https://arxiv.org/abs/2504.07981.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. *arXiv preprint arXiv:2411.17465*, 2024. URL https://arxiv.org/abs/2411.17465.

Zichuan Lin, Feiyu Liu, Yijun Yang, Jiafei Lyu, Yiming Gao, Yicheng Liu, Zhicong Lu, Yangbin Yu, Mingyu Yang, Junyou Li, Deheng Ye, and Jie Jiang. Ui-voyager: A self-evolving gui agent learning via failed experience, 2026. URL https://arxiv.org/abs/2603.24533.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018. URL https://arxiv.org/abs/1802.08802.

Junpeng Liu, Yifan Song, Bill Yuchen Lin, Wai Lam, Graham Neubig, Yuanzhi Li, and Xiang Yue. Visualwebbench: How far have multimodal llms evolved in web page understanding and grounding? In *COLM 2024*, 2024. URL https://arxiv.org/abs/2404.05955.

Fanbin Lu, Zhisheng Zhong, Ziqin Wei, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Steve: A step verification pipeline for computer-use agent training, 2025. URL https://arxiv.org/abs/2503.12532.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024a. URL https://arxiv.org/abs/2408.04682.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*, 2024b. URL https://arxiv.org/abs/2408.00203.

Zijian Lu, Yiping Zuo, Yupeng Nie, Xin He, Weibei Fan, Lianyong Qi, and Shi Jin. Contractskill: Repairable contract-based skills for multimodal web agents, 2026. URL https://arxiv.org/abs/2603.20340.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*, 2024. URL https://arxiv.org/abs/2402.05930.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. In *COLM 2025*, 2025. URL https://arxiv.org/abs/2504.08942.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

Fanglin Mo, Junzhe Chen, Haoxuan Zhu, and Xuming Hu. Building a stable planner: An extended finite state machine based planning module for mobile gui agent, 2025. URL https://arxiv.org/abs/2505.14141.

Anthony Nguyen. Improved gui grounding via iterative narrowing, 2024. URL https://arxiv.org/abs/2411.13591.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and Zhengyang Wu. Webcanvas: Benchmarking web agents in online environments, 2024. URL https://arxiv.org/abs/2406.12373.

Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. Large language models can self-improve at web agent tasks, 2024. URL https://arxiv.org/abs/2405.20309.

Viraj Prabhu, Yutong Dai, Matthew Fernandez, Jing Gu, Krithika Ramakrishnan, Yanqi Luo, Silvio Savarese, Caiming Xiong, Junnan Li, Zeyuan Chen, and Ran Xu. Walt: Web agents that learn tools. In *ICLR 2026*, 2026. URL https://arxiv.org/abs/2510.01524.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024. URL https://arxiv.org/abs/2411.02337.

Cheng Qian, Chi Han, Yi R. Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. *arXiv preprint arXiv:2305.14318*, 2023. URL https://arxiv.org/abs/2305.14318.

Yaoyao Qian, Yuanli Wang, Jinda Zhang, Yun Zong, Meixu Chen, Hanhan Zhou, Jindan Huang, Yifan Zeng, Xinyu Hu, Chan Hee Song, and Danqing Zhang. Webgrapheval: Multi-turn trajectory evaluation for web agents using graph representation. In *NeurIPS 2025 Workshop on Multi-Turn Interactions in Large Language Models*, 2025. URL https://arxiv.org/abs/2510.19205.

Yijun Qian, Yujie Lu, Alexander Hauptmann, and Oriana Riva. Visual grounding for user interfaces. In *NAACL 2024 Industry Track*, 2024. URL https://aclanthology.org/2024.naacl-industry.9/.

Zerui Qin, Sheng Yue, Xingyuan Hua, Yongjian Fu, and Ju Ren. Executable agentic memory for gui agent, 2026. URL https://arxiv.org/abs/2605.12294.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024. URL https://arxiv.org/abs/2405.14573.

Corby Rosset, Pratyusha Sharma, Andrew Zhao, Miguel Gonzalez-Fernandez, and Ahmed Awadallah. The art of building verifiers for computer use agents, 2026. URL https://arxiv.org/abs/2604.06240.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of lm agents with an lm-emulated sandbox. *arXiv preprint arXiv:2309.15817*, 2023. URL https://arxiv.org/abs/2309.15817.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

Paloma Sodhi, S. R. K. Branavan, Yoav Artzi, and Ryan McDonald. SteP: Stacked LLM policies for web actions. *arXiv preprint arXiv:2310.03720*, 2023. URL https://arxiv.org/abs/2310.03720.

Yixiao Song, Katherine Thai, Chau Minh Pham, Yapei Chang, Mazin Nadaf, and Mohit Iyyer. Bearcubs: A benchmark for computer-using web agents. In *COLM 2025*, 2025. URL https://arxiv.org/abs/2503.07919.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*, 2023. URL https://arxiv.org/abs/2305.16653.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *ACL 2025*, 2025. URL https://aclanthology.org/2025.acl-long.277/.

Fei Tang, Yongliang Shen, Hang Zhang, Siqi Chen, Guiyang Hou, Wenqi Zhang, Wenqiao Zhang, Kaitao Song, Weiming Lu, and Yueting Zhuang. Think twice, click once: Enhancing gui grounding via fast and slow systems, 2025. URL https://arxiv.org/abs/2503.06470.

Shizuo Tian, Hao Wen, Yuxuan Chen, Jiacheng Liu, Shanhui Zhao, Guohong Liu, Ju Ren, Yunxin Liu, and Yuanchun Li. Agentprog: Empowering long-horizon gui agents with program-guided context management, 2025. URL https://arxiv.org/abs/2512.10371.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024a. URL https://arxiv.org/abs/2401.16158.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024b. URL https://arxiv.org/abs/2402.01030.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024c. URL https://arxiv.org/abs/2409.07429.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. *arXiv preprint arXiv:2308.15272*, 2023. URL https://arxiv.org/abs/2308.15272.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. In *LLMAgents @ ICLR 2024*, 2024. URL https://arxiv.org/abs/2402.07456.

Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, Shuai Ren, Yafei Wen, Xiaoxin Chen, Aojun Zhou, and Hongsheng Li. Ui-genie: A self-improving approach for iteratively boosting mllm-based mobile gui agents. In *NeurIPS 2025*, 2025. URL https://arxiv.org/abs/2505.21496.

Bin Xie, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. Gui-explorer: Autonomous exploration and mining of transition-aware knowledge for gui agent. In *ACL 2025*, 2025a. URL https://aclanthology.org/2025.acl-long.282/.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL https://arxiv.org/abs/2404.07972.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. Scaling computer-use grounding via user interface decomposition and synthesis. In *NeurIPS 2025 Datasets and Benchmarks Track*, 2025b. URL https://arxiv.org/abs/2505.13227.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *arXiv preprint arXiv:2207.01206*, 2022. URL https://arxiv.org/abs/2207.01206.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

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

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks? *arXiv preprint arXiv:2407.15711*, 2024. URL https://arxiv.org/abs/2407.15711.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms. *arXiv preprint arXiv:2404.05719*, 2024. URL https://arxiv.org/abs/2404.05719.

Yuwen Zhai, Runze Li, Liang Wang, Nian Shi, Liwu Xu, Wei Zhang, Ran Lin, Bo Xu, and Benlei Cui. Guide: Interpretable gui agent evaluation via hierarchical diagnosis, 2026. URL https://arxiv.org/abs/2604.04399.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023. URL https://arxiv.org/abs/2312.13771.

Danyang Zhang, Situo Zhang, Ziyue Yang, Zichen Zhu, Zihan Zhao, Ruisheng Cao, Lu Chen, and Kai Yu. Progrm: Build better gui agents with progress rewards, 2025a. URL https://arxiv.org/abs/2505.18121.

Shaoqing Zhang, Zhuosheng Zhang, Kehai Chen, Xinbei Ma, Muyun Yang, Tiejun Zhao, and Min Zhang. Dynamic planning for llm-based graphical user interface automation. In *Findings of EMNLP 2024*, 2024. URL https://aclanthology.org/2024.findings-emnlp.70/.

Weijia Zhang, Kunlun Zhu, Zeyi Liu, Yinting Chen, Tianyi Ma, Jiateng Liu, Jiaxun Zhang, Bingxuan Li, Xiangru Tang, Heng Ji, and Jiaxuan You. Cuadebug: Diagnosing and repairing computer-use agent failures, 2026a. URL https://arxiv.org/abs/2608.02643.

Zhisong Zhang, Tianqing Fang, Kaixin Ma, Wenhao Yu, Hongming Zhang, Haitao Mi, and Dong Yu. Webrollback: Enhancing web agents with explicit rollback mechanisms. In *EACL 2026*, 2026b. URL https://arxiv.org/abs/2504.11788.

Ziyun Zhang, Xinyi Liu, Xiaoyi Zhang, Jun Wang, Gang Chen, and Yan Lu. Ui-evol: Automatic knowledge evolving for computer use agents. In *ICML 2025 Workshop on Computer Use Agents*, 2025b. URL https://arxiv.org/abs/2505.21964.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023. URL https://arxiv.org/abs/2308.10144.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024a. URL https://arxiv.org/abs/2401.01614.

Boyuan Zheng, Boyu Gou, Scott Salisbury, Zheng Du, Huan Sun, and Yu Su. Webolympus: An open platform for web agents on live websites. In *EMNLP 2024 System Demonstrations*, 2024b. URL https://aclanthology.org/2024.emnlp-demo.20/.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025. URL https://arxiv.org/abs/2504.07079.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *ICLR 2024*, 2024c. URL https://arxiv.org/abs/2306.07863.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023a. URL https://arxiv.org/abs/2310.04406.

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. *arXiv preprint arXiv:2307.13854*, 2023b. URL https://arxiv.org/abs/2307.13854.

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Li Erran Li. Proposer-agent-evaluator (pae): Autonomous skill discovery for foundation model internet agents. In *ICML 2025*, 2025. URL https://proceedings.mlr.press/v267/zhou25ah.html.

Zichen Zhu, Hao Tang, Yansi Li, Dingye Liu, Hongshen Xu, Kunyao Lan, Danyang Zhang, Yixuan Jiang, Hao Zhou, Chenrun Wang, Situo Zhang, Liangtai Sun, Yixiao Wang, Yuheng Sun, Lu Chen, and Kai Yu. Moba: Multifaceted memory-enhanced adaptive planning for efficient mobile task automation. In *NAACL 2025 (System Demonstrations)*, 2025. URL https://aclanthology.org/2025.naacl-demo.43/.

A Experimental Provenance and Cohort Scope

Table 5: Empirical cohorts and the lifecycle transition tested by each. Repetitions, model routes, runtimes, and denominators remain attached to their source cohort.

Cohort	Scope	Runtime / model	Transition tested
Repeated workflow control	28 generated tasks; 3 repeats; 84 runs/arm	Custom GUI executor; Qwen3.6-35B-A3B-NVFP4; 8 steps	Static repair versus stored family workflow
AgentLab workflow reuse	11 eligible generated tasks across 3 families; 3 repeats	AgentLab; Qwen3.8-27B-W8A8-DFlash2; 30 steps	Persistent reuse
Quantity-workflow transfer	5 post-freeze generated tasks; one paired run each	AgentLab; gpt-5.6-luna; 30 steps	Frozen transfer and cost
Failure conditioning	8 paired repair records/arm; 7 instances, 7 families	Same Qwen route within pairs; 20-step replay; up to 3 rounds	Failure evidence to source repair
Evidence-view study	8 failures; 8 views; 3 repeats; 192 sequences	AgentLab; DeepSeek Flash; 20-step replay; up to 3 rounds	Evidence sufficiency and compression
Instance-to-family chain	2 repaired sources; 3 original transfers; 5 post-freeze generated tasks	AgentLab; DeepSeek Flash; 20 steps	Abstraction and frozen transfer
Exact-patch rejection	2 order-task families; up to 3 repair rounds each	Sealed OpenCode repair and GUI replay	Source repair versus capability rejection
Shopping site-transfer probe	3 operational family entries; 2 workflow types; 1 rejected family	AgentLab; gpt-5.6-luna; 20 steps; separate storefront	Public browser contracts and isolated-Registry reuse

The repeated-control cohort contains 28 tasks, three repetitions, and four archived arms, for 336 valid arm records. Two generated tasks were excluded in precheck. A connectivity-affected repeat

was retained for diagnosis, and one connection-reset cell in the third valid repeat was replaced under the frozen configuration. The archived result includes these selection and retry rules.

Static compensation is the archived "Evolved Static Compensation (w/o Registry)" condition, which includes accumulated execution enhancements. Its 59 evaluator acceptances coexist with pollution in all 84 runs, making it the diagnostic repair arm rather than an intermediate level of valid capability. The parameter-closure arm predates the evidence-alignment module.

The two Base execution-positive records are task 18003 in repetition 2 and task 18009 in repetition 3. Each records evaluator failure, pollution, and drift. Evaluator-positive and execution-positive records are therefore reported as separate marginals. Source reports preserve the raw values.

For the AgentLab reuse study, evaluator counts are private-evaluator outcomes on generated tasks. In the review-product subset, five tasks were generated and three were precheck-eligible; two page/snapshot mismatches were excluded before execution. Cross-family report/payment tasks also underwent precheck, with exclusions retained in their manifests. Base pass rates use the private evaluator under the matched task snapshot.

A.1 Second-site Shopping deployment

The probe used `gpt-5.6-luna`, temperature zero, a 20-step limit, real-browser replay, and a Shopping-only Registry. Table 6 separates instance repair, parameter variation, and later routing. Family 204's parameterized search procedure reached requested result pages on two other original tasks and an empty-result boundary after two CodeAgent instance tools reached task-relevant pages. Its original-task string evaluators remain negative, and the no-match task has no private oracle. Family 139 bound category and price across two original tasks and a new parameter combination. Family 212 bound two keywords, although its original Base task had already passed. Family 138's two instance tools executed, but cross-instance extraction rejected their incompatible navigation and sorting orders. The generic evidence-alignment verifier did not mark Shopping task completion as verified even where the public URL/page contracts passed. The operational gate reports and Shopping lifecycle report retain the run-level evidence. Shopping candidates have mixed engineering lineages; these routes are separate from Admin's private-evaluator cohorts and are not additional strict CodeAgent-provenance promotions.

Table 6: Second-site Shopping probe. Counts denote distinct browser-tested tasks, not repeated trials. "Route" is a later invocation with `tool_source=registry`. The public contract checks task parameters against the live URL and page; Shopping private-oracle acceptance is unavailable.

Family	Workflow and browser evidence	Development	Transfer / boundary	Registry route
204	Query search; requested query in URL and result page	2 instance tools	2 original tasks + 1 no-match	3/3
139	Category-price; category path and price bound in URL	2 original tasks	1 new parameter task	1/1
212	Query search; requested keyword in input and URL	1 original task	1 new parameter task	1/1
138	Two executable instance tools, but distinct operation orders	2 diagnostic tasks	Family extraction rejected	None

A.2 Sealed source-success rejections

Two sealed OpenCode experiments supply counterexamples. A family-367 repair passed source, development, regression, and routing checks, but a real-GUI no-match test requested `Suspected Fraud` while the browser retained `Pending`; family 366 passed source replay but then falsely reported that the visible `Pending` option was unavailable. Both control-binding failures rejected admission before frozen transfer. They establish that source success can fail a required capability

condition, but two selected cases do not estimate a population false-promotion rate; later operational versions belong to separate engineering lineages.

A.3 Capability inventory and lineage scope

Table 7: Persistent-memory inventory and principal evidence lineage for each operational family.

Family	Workflow	Principal evidence scope
234	Most cancellations	Mixed engineering, generated-task evaluation, runtime smoke
250	Review product query	Operational v1 reuse; separate strict v2 rejected
253	Order status query	Generated-task comparison and operational registration
270	Monthly orders report	Parameterization, repeated control, AgentLab reuse
279	Bestsellers report	Early parameterized workflow and persistent-memory validation
288	Review-term count	Source/tool development, parameterization, repeated control
364	Customer phone query	Early parameterization and later reuse checks
366	Order attributes	Strict lineage rejected; later mixed operational version admitted
367	Payment aggregate	Strict lineage rejected; later operational version and reuse
368	Product quantity query	Full-instance tools, parameterized transfer, operational registration

The historical Shopping Admin scan contains 182 original task records: 89 archived strict successes, 58 evaluator-or-execution failures, and 35 runner timeouts. Execution and evidence marginals were 130/182 and 147/182. These categories provide a failure inventory, not 93 clean repairable capability gaps. Environment and evaluator causes require diagnosis. A separate historical inventory records 21 accepted source-replay repairs, but lacks a common all-attempt denominator and uniform downstream validation; no repair-success percentage or 21-to-10 promotion funnel is inferred.

Additional operational cohorts include 15 generated instances across families 249/250/253 with 15/15 workflow-enabled completions under the retrospective page/postcondition proxy, and five family-234 instances with 5/5 workflow-enabled completion. Family 249 nevertheless failed a browser boundary and was not admitted. Later mixed order-family validation recorded 18/18 positive cases and 17/17 browser-boundary cases. These are supplementary cohorts with their own artifacts and are not added to the repeated-control denominator.

A.4 Matched efficiency and gate-policy details

The five-task family-368 comparison keeps the same task inputs in both arms and records provider-reported tokens (Table 8). The three-candidate policy audit uses historical mixed-lineage records; its disagreement rate is defined relative to the recorded full gate, not an independently measured prospective error rate (Table 9).

Table 8: Matched efficiency on five positive family-368 tasks. Token counts are provider-reported.

Metric	Base	Frozen workflow	Reduction
Page-grounded completion	5/5	5/5	preserved
Browser actions	28	10	64.3%
Online LLM calls	38	10	73.7%
Total tokens	194,083	61,612	68.3%
Wall time (s)	276.2	159.0	42.4%

Table 9: Acceptance-policy audit over the same three historical candidate records. Disagreement counts candidates admitted by a reduced rule but rejected by the recorded full gate.

Acceptance policy	Accepted	Full-gate rejections	Disagreement rate
Source only	3/3	1	33.3%
Held-out only	3/3	1	33.3%
Full BoundaryLoop gate	2/3	0	0.0%
Full minus browser boundary	3/3	1	33.3%

B Repair Interface and Failure Semantics

The instance artifact contains executable operations, public parameter bindings, applicability conditions, browser-state expectations, and a completion/evidence handoff. It contains no fixed reference answer. The input package exposes public task semantics, the permitted failure trajectory, same-family development evidence, current allowed code where relevant, and sanitized validation feedback. Its manifest distinguishes original evidence from later inspection.

Table 10: Observable failure categories and the evidence needed to investigate them. Categories describe diagnosis targets, not a measured exhaustive classifier.

Category	Observable discrepancy	Repair/validation target
Goal or plan	Recorded plan/actions pursue a different public objective	Navigation and completion contracts; do not invent missing reasoning
Action grounding	Intended control and acted-on DOM element disagree	Visible-control identity, option mapping, post-action state
State preservation	Selection differs from applied/request/evidence filter	Canonical parameter chain and actual request observation
Completion	Evidence retrieved but answer absent, unsupported, or incomplete	Evidence handoff and submitted task-grounded answer
Instance binding	Workflow carries a development entity into a new task	Explicit slots and cross-instance page checks
Environment	Authentication/network/reset fails before valid execution	Bounded recovery; blocked label rather than semantic repair

The implemented status-chain checks are family-aware observable checks. They do not access model hidden states. Screenshots are stored as evidence when available, but the reported cohorts do not constitute a controlled study of multimodal representation learning. Similarly, preserving a verified prefix is a conservative controller constraint, not proof that an earlier plan was globally optimal.

C Mechanism Evidence and Open Identification Questions

Evaluation validity. Page-grounded completion requires task-specific evidence, not mere arrival at the right URL or agreement with an auxiliary string. Legacy flags are imperfect proxies; exact evaluators and rendered evidence require independent semantic audit when they disagree. The new contract has not been rerun over all historical data. The available cohorts establish finite evidence sufficiency and one operational instance-to-family chain, not a universal ranking of evidence views.

Evidence content versus alignment. The completed 192-sequence cohort compares task-only, trajectory-only, raw, aligned, compact, shuffled, action-ablated, and DOM/state-ablated views under a fixed model and repair/replay budget. Task-only passes 0/24 trials; aligned-minimal and raw full each pass 6/24, with the compact view using 12.7% fewer input tokens. This establishes finite compression without a repair-rate advantage. Because common execution-contract and selector scaffolding remains outside the rendered evidence block, the result does not identify the causal effect of alignment structure alone.

Counterexample feedback. Compare observed validation feedback with independent retries under the same total generation and replay budget. Record repeated failure-frontier errors, changes

to verified prefixes, rounds to acceptance, and censored failures. A source that exhausts the budget stays in the denominator. Authentication blocks are reported separately under predefined retry rules.

Cross-instance abstraction. The family-368 chain instantiates this protocol. Two CodeAgent-produced instance workflows were normalized into a scalar/range family workflow, replayed on three original tasks excluded from extraction, frozen, and evaluated on five subsequently generated tasks excluded from repair context. All transfer runs satisfy the declared page-grounded contract; strict evaluator mismatches for some historical rows remain separately recorded and are not relabeled as success. The abstraction step is deterministic engineering rather than direct CodeAgent output, and the isolated persistent-memory smoke establishes routing rather than a new formal admission event.

Acceptance-policy analysis. The supplemental policy audit applies source-only, held-out-only, full, and full-minus-one-gate policies to the same three historical candidate records. Source-only, held-out-only, and full-minus-boundary each accept family 249 even though the recorded full policy rejects it; the full policy accepts 2/3 and has no recorded false promotion in this cohort. Formal replay remains unknown, and the cohort is mixed-provenance, so the audit identifies a gate-specific falsification pattern rather than estimating prospective precision or recall. The machine-readable inputs and recomputed counts are archived in `reports/current/supplemental_gate_contract_audit_20260924/`.

Runtime contract diagnosis. The live family-368 diagnostic provides a concrete execution-level failure mode for persistent reuse. The active Registry artifact failed a trailing-slash URL postcondition before parameter injection; an isolated contract normalization restored execution and page evidence on 3/3 runs with zero pollution. All three runs nevertheless encountered current-page versus frozen-oracle drift. This diagnosis complements, rather than replaces, browser-boundary acceptance: contract checks govern executable replay, while boundary and oracle checks govern capability admission.

Finite evidence sufficiency. The evidence-view study supports a compact sufficient package for the observed successes: task semantics, failure frontier, a three-action window, and browser state. "Sufficient" remains conditional on the eight sources, DeepSeek Flash route, shared prompt scaffold, three repair rounds, and 20-step replay. Since action- and DOM-ablated conditions also retain successes, no package is inclusion-minimal and no global minimum is claimed.

Shared-model complementarity and lifecycle cost. Compare the two-role loop with online WebAgent reflection and a direct browser-equipped CodeAgent, under matched access and total inference budgets. Include synthesis, validation, failures, and subsequent reuse in cost accounting. Such an experiment is needed before claiming that role separation itself, rather than extra computation or privileges, explains the advantage.

D Extended Related-Work Landscape

The main text compares BoundaryLoop with the work closest to its acquisition criterion. This appendix places its evidence and promotion mechanisms in the broader Web and GUI agent literature.

Interactive Web and GUI environments. MiniWoB++, WebShop, WebLINX, and VisualWebArena progressively move from compact browser control to grounded shopping, dialogue-conditioned navigation, and multimodal tasks on realistic websites (Liu et al., 2018; Yao et al., 2022; Lù et al., 2024; Koh et al., 2024). AssistantBench adds long-horizon Web assistance, while OSWorld and AndroidWorld extend executable evaluation to stateful desktop and mobile environments (Yoran et al., 2024; Xie et al., 2024; Rawles et al., 2024). Live-site platforms such as WebOlympus and WebCanvas expose agents to changing content and interfaces; REAL instead offers deterministic replicas of real websites for reproducible state-based evaluation (Zheng et al., 2024b; Pan et al., 2024; Garg et al., 2025). BEARCUBS broadens live-Web evaluation to multimodal information seeking, and Agent-E studies architectural principles for autonomous navigation (Song et al., 2025; Abuelsaad et al., 2024). WebSuite complements end-task scores with action-level diagnostic tasks that attribute a failure to a concrete Web operation (Li & Waldo, 2024). Together, these

environments show why task completion, intermediate state, and execution trace must be considered jointly. BoundaryLoop uses this joint view to decide whether a repaired workflow has crossed from one successful episode to a reusable capability.

Grounding and stateful interface control. CogAgent, OmniParser, ShowUI, Ferret-UI, and ScreenAI improve visual understanding, interface parsing, coordinate grounding, and action prediction (Hong et al., 2023; Lu et al., 2024b; Lin et al., 2024; You et al., 2024; Baechler et al., 2024). Mobile-Agent, AppAgent, and AutoDroid show how these representations support multi-step interaction with stateful interfaces (Wang et al., 2024a; Zhang et al., 2023; Wen et al., 2023). Visual UI Grounding formalizes screenshot-to-element localization without assuming complete UI metadata, while VisualWebBench evaluates page understanding and grounding jointly (Qian et al., 2024; Liu et al., 2024). Iterative narrowing and fast–slow grounding refine the search region or invoke additional reasoning when direct localization is uncertain (Nguyen, 2024; Tang et al., 2025). ScreenSpot-Pro stresses high-resolution professional interfaces, and UI decomposition with synthesis scales grounding supervision across complex screens (Li et al., 2025; Xie et al., 2025b). ComponentBench further isolates failures at the UI-component level and exposes the effect of observation and action interfaces on agent behavior (Guan et al., 2026). BoundaryLoop connects these advances through a cross-view repair contract: a task parameter, its grounded control, the selected GUI value, and the resulting browser state must agree. The contract turns grounding from an intermediate prediction into a property checked during repair and promotion.

Failure diagnosis, reflection, and verification. Recent analyses decompose Web-agent failures into planning, low-level execution, and replanning errors, while GUI benchmarks localize failures to particular components and transitions (Aghzal et al., 2026; Guan et al., 2026). StepReflect represents intended UI transitions and compares them with paired visual observations; GUIDE aggregates subtask-level diagnoses into an interpretable trajectory verdict (Guo et al., 2026; Zhai et al., 2026). Work on computer-use verifiers separates process evidence from terminal outcomes and manages screenshot context for reliable judgment (Rosset et al., 2026); CUADebug similarly identifies a causal trajectory step before producing a structured correction (Zhang et al., 2026a). BoundaryLoop connects these diagnostic ideas to capability construction. It aligns task semantics, actions, DOM targets, observed state, and postconditions at the earliest verifiable failure frontier, then compiles the discrepancy into a repair that must survive real-browser replay. Counterexamples update the repair hypothesis rather than merely prompting another unconstrained attempt.

Trajectory evaluation and process supervision. Outcome-only evaluation obscures where an otherwise useful trajectory diverges. AgentRewardBench studies automatic judgments of Web-agent success, side effects, and repetition, showing that evaluation quality itself is a distinct systems problem (Lù et al., 2025). STEVE labels steps from before–after screenshots, while ProgRM assigns dense progress signals to preserve useful prefixes in failed trajectories (Lu et al., 2025; Zhang et al., 2025a). WebGraphEval represents multiple trajectories as an action graph, making recurring paths, redundancies, and critical decisions visible (Qian et al., 2025). Verifier-driven mobile agents and essential-state procedural evaluation similarly check intermediate state transitions instead of relying exclusively on the terminal answer (Dai et al., 2026; Chai et al., 2025). BoundaryLoop uses process evidence for a different decision: it identifies the verified prefix and failure frontier that constrain repair, then requires the resulting procedure to satisfy terminal and boundary contracts before admission to memory.

Experience, self-improvement, and executable skills. Synapse retrieves successful trajectories as control exemplars, and Web-agent self-improvement methods convert interaction experience into synthetic supervision or updated policies (Zheng et al., 2024c; Patel et al., 2024). PAE autonomously proposes, practices, and evaluates Internet skills; OpenWebVoyager improves a multimodal Web agent through real-world exploration and feedback (Zhou et al., 2025; He et al., 2025). GUI-explorer mines transition-aware knowledge from observation–action–outcome triples, while UI-Evol updates computer-use knowledge through interaction (Xie et al., 2025a; Zhang et al., 2025b). OS-Genesis synthesizes trajectories by exploring first and deriving tasks retrospectively, addressing the diversity and quality of GUI experience (Sun et al., 2025). WALT reverse-engineers website functionality into invocable search, filter, sort, and editing tools, reducing dependence on fragile low-level action sequences (Prabhu et al., 2026). OS-Copilot, UI-Genie, and UI-Voyager likewise accumulate

experience or learn from failed interactions (Wu et al., 2024; Xiao et al., 2025; Lin et al., 2026). Complementary work abstracts histories into workflows or skills, including Agent Workflow Memory, SkillWeaver, SCAFFOLD, ContractSkill, and Executable Agentic Memory (Wang et al., 2024c; Zheng et al., 2025; He et al., 2026; Lu et al., 2026; Qin et al., 2026). BoundaryLoop contributes the acceptance transition: repaired instances reveal cross-instance invariants, which are parameterized, gated, and then stored as persistent executable memory.

State, planning, and recovery. Web interaction is non-stationary within an episode because each action changes both the visible interface and the set of valid next actions. Auto-Intent discovers task intents from websites and uses them to drive self-exploration, connecting semantic task structure to executable behavior (Kim et al., 2024). Dynamic Planning of Thoughts revises plans from environmental feedback and action history, while finite-state-machine planning encodes application control logic into stable execution paths (Zhang et al., 2024; Mo et al., 2025). MobA combines memory with adaptive planning for mobile automation, and AgentProg uses program structure to manage long-horizon GUI context (Zhu et al., 2025; Tian et al., 2025). WebRollback adds an explicit transition for returning from erroneous browser states (Zhang et al., 2026b). These methods improve online control within an episode. BoundaryLoop operates across episodes: a failed transition produces repair evidence, repeated instances expose an invariant workflow, and later boundary failures reopen the loop rather than silently broadening the workflow's applicability.