VERIFIED WORLD-MODEL REPAIR FOR LONG-HORIZON INTERACTIVE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon interactive agents can fail because a false belief about the world survives the very action that should disprove it. We introduce verified world-model repair, a framework that maintains a structured belief state, localizes action-conditioned prediction errors, converts them into scoped repair programs, and commits a repair to memory only after independent validation. A stepwise recovery specialist learns complete successful recovery trajectories rather than isolated next-action corrections. We also construct the World Evolution Dataset (WED), a family-disjoint dataset and paired-recovery protocol derived from 10,000 interaction episodes and 162,756 transitions. Across five random seeds on ALFWorld failure checkpoints, the full system recovers 169 of 185 cases (91.35%), compared with 109 of 185 (58.92%) for a strong goal-state controller. Learning only the first corrective action reaches 44.14%, whereas learning the full recovery trajectory reaches 90.99%. The method remains effective on held-out scenes, objects, and task templates, and reaches 93.69% on Qwen3-8B after development-only adaptation, while limiting harmful corrections to 3.6%. On frozen WebArena and OS-World subsets, it improves programmatic task success over the strongest reflection baseline by 6.10 and 6.00 percentage points, respectively. These results indicate that reliable long-horizon recovery requires verifying a correction and carrying it through the remaining task, rather than merely revising the next action.

1 INTRODUCTION

Errors in long-horizon interaction are rarely isolated invalid actions. When feedback conflicts with an earlier belief, an agent may continue to schedule actions under the same state assumption. For example, if a click fails to open the expected dialog, subsequent typing and confirmation are directed to the wrong target. Re-selecting an action is not sufficient when the underlying assumption has not been revised. Recent work on interactive knowledge correction shows that experience can revise priors that otherwise cause repeated failures (Lee et al., 2026). We study the corresponding recovery problem: after an observation falsifies an expectation, how should an agent decide what to repair and continue using that correction for the rest of the task?

Prior work improves interaction through environment prediction, experience reuse, and feedback learning. World models estimate the state changes caused by candidate actions and use the estimate for action selection (Chae et al., 2025). Workflow and reasoning memories distil procedures or strategies from previous interactions (Wang et al., 2025b; Ouyang et al., 2026), while recovery-oriented training teaches models to continue acting after feedback (Fu et al., 2025). UI-Copilot separates persistent observation from temporary execution context and retrieves memory on demand for long-horizon tasks (Lu et al., 2026). We focus on what happens after an error: we connect the current contradiction, the scope in which a repair applies, and the subsequent execution so that a correction remains useful until task completion.

We propose EVOWORLD, an interaction loop that combines structured beliefs, action expectations, scoped repair, and observable verification. The agent records the active and completed subgoals and unresolved information, and predicts a task-relevant visible effect for the chosen action. When a new observation contradicts that effect, or when the same target makes no progress, the controller localizes the affected belief and generates a short repair program. The program is executed one safe step at a time, with a fresh observation after each step. Target-related evidence determines

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.

whether the controller proceeds or relocates the target. Experiences validated during development are compressed into a scope, trigger signal, repair sequence, and verification condition, then read as frozen context in external evaluation.

We build a recovery view of the World Evolution Dataset (WED) from 10,000 interaction trajectories and 162,756 state transitions, with family-disjoint splits. On matched ALFWorld failure checkpoints, EVOWORLD-Full recovers 169/185 cases (91.35%) across five seeds, compared with 109/185 (58.92%) for a goal-state controller. Complete-recovery supervision reaches 90.99% in a matched three-seed comparison, whereas first-action supervision reaches 44.14%. A development-selected adaptation to Qwen3-8B reaches 93.69%. On frozen external instances using OPENCUA-7B, the method obtains mean task success rates of 28.25% on OSWorld and 30.14% on WebArena, exceeding four matched baselines.

Our contributions are threefold. First, we introduce a repair framework driven by observable contradictions, with an explicit scope and acceptance condition. Second, we construct a WED recovery view and a matched-checkpoint protocol that isolates the effect of complete-recovery supervision from first-step correction. Third, we evaluate the framework through core recovery, mechanism, generalization, and browser/desktop experiments. Together, these experiments test a single question: can an interaction agent revise a falsified assumption, verify the revision, and preserve it while completing the remaining task?

2 RELATED WORK

World models and knowledge correction. WMA learns natural-language representations of web-state transitions, predicts the consequences of candidate actions, and combines them with value estimates (Chae et al., 2025). XENON revises item dependencies and failed-action knowledge in Minecraft from interaction feedback (Lee et al., 2026). Computer-use agents combine hierarchical planning and experience (Agashe et al., 2025a;b), learn visual grounding and planning jointly (Xu et al., 2025c), or verify attention-based action regions (Wu et al., 2025). EVOWORLD instead focuses on repair after an action: the predicted effect is paired with the new observation to identify the falsified belief and constrain the repair scope.

Reflection and recovery learning. ReAct alternates reasoning and acting (Yao et al., 2023); Self-Refine revises outputs from model self-feedback (Madaan et al., 2023); and Reflexion converts feedback into verbal reflection and episodic memory (Shinn et al., 2023). CRITIC adds tool interaction as external evidence for revision (Gou et al., 2024). AgentRefine learns from error-and-revision trajectories, WebSTAR filters step-level supervision, and WebRL expands training curricula from failures (Fu et al., 2025; He et al., 2026; Qi et al., 2025). We ask how much execution should be learned after a correction: first-action supervision and complete-recovery supervision are compared from the same checkpoint and with the same terminal criterion.

Experience memory and workflows. ExpeL extracts natural-language knowledge from task experience and retrieves it with examples at inference time (Zhao et al., 2024). AutoManual compiles interaction-derived rules into planning manuals (Chen et al., 2024). AWM and Agentic Plan Caching organize reusable workflows or adjustable plan templates (Wang et al., 2025b; Zhang et al., 2025). ReasoningBank and A-Mem structure self-evaluated experiences and evolving memory records (Ouyang et al., 2026; Xu et al., 2025a). EVOWORLD uses a local repair record with a trigger, scope, repair, and verification condition; the external benchmark reads frozen records, while task-local state changes with each observation. AWM-offline is therefore a direct workflow-memory comparison.

3 VERIFIED WORLD-MODEL REPAIR

Normal execution and recovery share the same action interface. The controller proposes an action and its expected effect, then updates its belief from the returned evidence. Figure 1 embeds repair in this loop: the system diagnoses the contradiction, makes a local correction, verifies it, and resumes the original task. Core training learns continuation actions from complete recovery trajectories; the

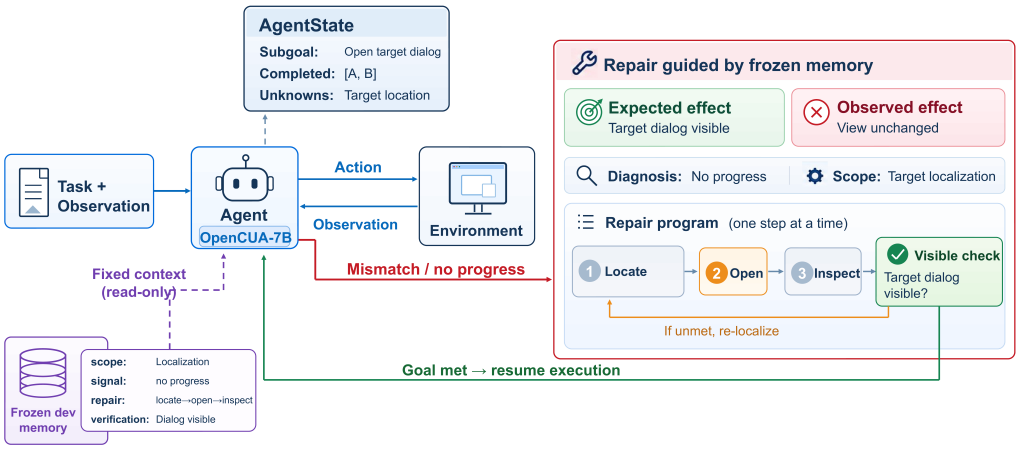


Figure 1: Verified world-model repair. The agent maintains task-local beliefs and compares the expected effect of an action with its next observation. A mismatch or persistent lack of progress triggers a scoped repair, executed one step at a time and checked with visible, target-related evidence. Frozen development records provide read-only context. Satisfying the repair goal resumes the original task; terminal evaluation follows interaction.

external instances reuse the same state, repair, and verification controller with frozen development experience.

3.1 Observable Beliefs and Expected Effects

Given a goal g, the agent receives an observation o_t and maintains a task-local belief state b_t. Synapse shows how state abstraction can remove task-irrelevant information while retaining trajectory exemplars for multi-step decisions (Zheng et al., 2024). EvoWorld organizes the state around the active recovery target: location, window or page, visible entities, active and completed subgoals, and unresolved attributes are recorded. Unknown attributes are kept explicitly, and the most recent eight actions and observation fingerprints provide a short history. New feedback updates the current belief, while completed subgoals constrain later repair.

For action a_t, the controller predicts a target-related visible effect e_t using frozen development context C_{dev}:

$$e_t = f_\theta(g, o_t, b_t, C_{\text{dev}}, a_t). \tag{1}$$

The prediction is used primarily as a post-action check rather than as a hidden reward. After executing a_t, the controller obtains o_{t+1} and compares it with e_t. For example, a click on an entry predicts that a target dialog and its controls will become visible. The dialog, text, window focus, URL, scroll position, and target entity provide observable evidence for or against the prediction.

3.2 Scoped Repair and Stepwise Execution

Repair is triggered when the next observation contradicts an expected effect or when repeated interaction with the same target makes no progress. The diagnosis identifies the mismatch, while `repair_scope` specifies the entity, location, or subgoal to reconsider. Completed subgoals remain in the task-local state. For example, an unsuccessful attempt to open a dialog calls for checking its entry point, not repeating an already completed login and navigation sequence.

A repair program is a short sequence of steps with observable targets. Only the current step is executed before another observation is obtained. The controller then advances the program, revises

the current step, or relocates the target. In the example in Figure 1, locating the entry supports an opening action; the resulting dialog supports inspection. Each transition therefore depends on evidence from the preceding action rather than on the mere presence of that action in the history.

The current-step check and the repair-goal check serve different purposes. Finding a control permits the next repair step; satisfying the repair goal returns control to the original task. The desktop implementation allows at most three bounded relocations before explicit termination when no executable alternative is available. Every executed recovery action uses the original interaction budget. This keeps local recovery coupled to the remaining task instead of creating a separate, open-ended retry procedure.

3.3 Observable verification and frozen experience

Verification compares the expected effect with evidence obtained after execution. Screenshot fingerprints provide a change cue; visible text, window focus, URLs, scroll positions, and target entities establish whether that change advances the active subgoal. A click record alone does not establish that a dialog opened. The next observation must expose the predicted dialog or its controls. A satisfied intermediate check advances the repair, whereas a visible task-completion condition permits `DONE`. The terminal evaluator assigns the task score after interaction.

Development interactions provide reusable repair experience. For each candidate record, we examine the pre-action expectation, the observed mismatch, the executed repair, and the evidence that accepts the resulting change. Accepted records contain four fields: `scope`, `signal`, `repair`, and `verification`. Scope identifies where the record applies; signal specifies the triggering observation; repair describes the local continuation; and verification gives its acceptance condition. Together, these fields preserve the connection between an error and a repair that addresses it.

During external evaluation, Full reads at most 12 records in their fixed JSON order. The file remains unchanged between episodes, while task-local beliefs continue to update after every observation. Standard, State-Chain, and Reflection use their respective task-local decision rules, AWM-offline supplies workflow context (Wang et al., 2025b), and Full supplies verified repair context. Observation, action execution, and terminal evaluation are shared. The distinction between workflow context and repair context is thus expressed in the stored experience and the control rule that uses it.

3.4 Learning complete recovery trajectories

Core ALFWorld training uses successful recovery suffixes following a failure checkpoint. A record contains observable context h_i, a target action a_i^+, a candidate-action set, and the recovery-group identifier. The first-action view retains the earliest corrective record in each group. The complete-recovery view retains the subsequent recovery records as well, exposing the learner to the states encountered after the initial correction. For checkpoint t_c and terminal step T, the action-level supervision sets are

$$\mathcal{S}_{\text{first}} = \{(h_{t_c}, a_{t_c})\}, \qquad \mathcal{S}_{\text{complete}} = \{(h_t, a_t) : t_c \le t \le T\}. \tag{2}$$

Both settings are evaluated by continuing to the same terminal criterion under the same interaction budget. Figure 2 shows the supervision span and its relation to matched-checkpoint evaluation.

Let $s_\theta(h, a)$ be the trainable score assigned to a candidate action. For the negative-action set $\mathcal{N}_i$, the implemented ranking loss and per-record objective are

$$\mathcal{L}_{\text{rank},i} = \frac{1}{|\mathcal{N}_i|} \sum_{a^- \in \mathcal{N}_i} \text{softplus}\big(0.65 - s_\theta(h_i, a_i^+) + s_\theta(h_i, a^-)\big)\,, \tag{3}$$

$$\mathcal{L}_i = w_i \left[0.80\mathcal{L}_{\text{rank},i} + 0.20\big(-s_\theta(h_i, a_i^+)\big)\right]. \tag{4}$$

Each record contributes at most five negative candidates, prioritizing actions with the same verb as the target. The task-type weight w_i and the remaining optimization settings are given in Appendix B.3 and Table C1. The ranking term separates the target from competing actions, while the second term increases its score. Complete-recovery supervision applies this objective across the retained suffix rather than only at its first step. The supervision design builds on learning from error-and-revision trajectories (Fu et al., 2025).

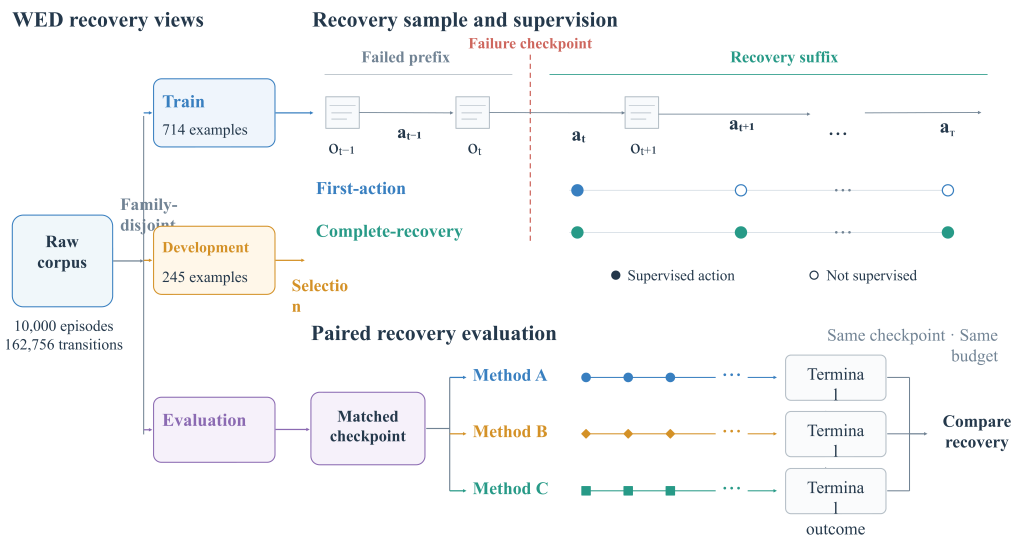


Figure 2: WED recovery views and paired evaluation. Family-disjoint training, development, and evaluation views serve learning, configuration selection, and evaluation. The supervision schematic marks either the first corrective action or the complete recovery action span. Each evaluated method continues from an independent copy of the same checkpoint under the same maximum action budget and receives its own terminal outcome.

Core ALFWorld recovery uses OpenCUA-7B with low-rank adaptation and development-based checkpoint selection (Hu et al., 2022); initialization, update counts, and selection rules are specified in Appendix C.1. The first-action and complete-recovery views contain 60/15 and 714/245 training/development records, respectively. For the Qwen3-8B experiment, the adaptation configuration is selected on development data and frozen before three-seed recovery evaluation. Browser and desktop experiments instead use a frozen OpenCUA-7B backbone with the repair controller; they do not perform this additional recovery training.

3.5 Browser and desktop instantiation

External instances use a frozen OpenCUA-7B backbone (Wang et al., 2025a) connected to OSWorld and WebArena (Xie et al., 2024; Zhou et al., 2024). Desktop observations contain screenshots and available accessibility information and use a whitelisted PyAutoGUI parser; browser observations contain screenshots and visible page candidates and use BrowserGym function calls (Drouin et al., 2024; Le Sellier de Chezelles et al., 2025). Candidates are regenerated after page changes, and the two action parsers remain separate. Both interfaces return a new observation to the same state, repair, and acceptance loop.

4 WED and Evaluation Protocol

4.1 Family-disjoint data views

AgentTrek and Learn-by-interact synthesize training data from guided replay and interaction histories, respectively (Xu et al., 2025b; Su et al., 2025). WED organizes ALFWorld records around failure checkpoints and recovery continuations (Shridhar et al., 2021). A raw episode contains the observations and actions of one execution, and a transition records an action together with its preceding and following state. The raw corpus contains 10,000 episodes and 162,756 transitions. Its canonical pilot contains 2,000 episodes and 30,766 transitions. Recovery training and development views contain 714 and 245 records, respectively. Table 1 separates these processing views and their evaluation uses.

Table 1: WED data views and evaluation units.

Data view	Scale	Use
Raw corpus	10,000 episodes; 162,756 transitions	Interaction records
Canonical pilot	2,000 episodes; 30,766 transitions	Normalized view
Recovery training	714 samples; 36 families	Parameter learning
Recovery development	245 samples; 9 families	Configuration selection
Checkpoint evaluation	37 families $\times$ 5 seeds $= 185$	Final recovery
External evaluation	361 OSWorld; 678 WebArena tasks	3 seeds per method

A `family_id` identifies the source ALFWorld task/trial instance and serves as the split unit. Checkpoints, recovery groups, and step records derived from the same family inherit the same split. The training, development, and primary evaluation sets contain 36, 9, and 37 non-overlapping families. A `group_id` links steps within one recovery process, and the step suffix in `example_id` determines their order. Retaining the earliest step of each group gives 60 training and 15 development records for first-action supervision; retaining the recovery continuation gives the complete-recovery view. Family counts remain 36 and 9 in both training views. Separate holdout experiments test changes in scenes, objects, task templates, and error factors.

4.2 Matched-checkpoint recovery and task success

Each failure checkpoint is restored in an independent environment copy. All methods receive the same task and checkpoint context and continue under the action limit recorded in that checkpoint's `matched_step_budget`. The primary comparison covers 37 checkpoint families and five seeds (185 records); the supervision-span comparison uses three seeds (111 records). A recovery is successful if the task goal is completed before the budget is exhausted. For N fixed starting points and K seeds, with $z_{is} \in \{0, 1\}$ denoting completion, the aggregate success rate is

$$\text{SuccessRate} = 100 \times \frac{1}{NK} \sum_{i=1}^{N} \sum_{s=1}^{K} z_{is}. \tag{5}$$

When the start is a failure checkpoint this is a recovery rate; from the initial state it is task success. A correct first action or a passed intermediate repair check is therefore insufficient on its own: the remaining task must also be completed. Repeated seeds share the checkpoint families, so uncertainty is reported with the corresponding family and seed structure in Appendix H. External evaluation uses 361 OSWorld tasks after excluding eight Google Drive/OAuth tasks and 678 WebArena tasks after excluding Map and Wikipedia dependencies. The frozen main comparison reports seeds 17, 29, and 43 with maximum environment steps of 20 and 30, respectively. Valid task failures remain in the fixed denominator, while engineering gaps are tracked separately.

5 Experiments

5.1 Experimental setup

The study has three stages. Stage 1 tests action-conditioned diagnosis, repair verification, and experience reuse. Stage 2 evaluates ALFWorld recovery learning, supervision span, generalization, training-token scale, and cross-backbone adaptation to Qwen3-8B. Stage 3 validates the frozen controller in browser and desktop environments. The external comparison contains Standard, State-Chain, Reflection, AWM-offline, and EvoWorld-Full, all using the same OpenCUA-7B backbone, task scope, and environment-step budgets on eight A100 GPUs.

Standard directly selects the next action. State-Chain maintains task-local state. Reflection produces a short reflection after failure or repetition. AWM-offline reads reusable workflows distilled from development interactions (Wang et al., 2025b). Full combines scoped repair, observable verification, and frozen development experience. AWM uses the corresponding desktop action interface. Training, inference, and experience-construction settings are given in Appendix C.

Table 2: Matched-checkpoint recovery on ALFWorld.

Method	Successes / total	Recovery (%)	Mean steps
No repair	32/185	17.30	16.79
ReAct	43/185	23.24	16.12
Generic reflection	39/185	21.08	16.29
State chain	40/185	21.62	16.23
Retrieved insight	36/185	19.46	16.78
Raw memory	51/185	27.57	16.21
EvoWorld-RH	73/185	39.46	16.04
EvoWorld-Goal	109/185	58.92	13.85
EvoWorld-Full	169/185	**91.35**	**10.52**

All methods use 37 checkpoint families and five seeds. Rates are computed from 185 checkpoint–seed outcomes.

5.2 Recovery performance on ALFWorld

EvoWorld-Full achieves the highest recovery rate and the fewest mean interaction steps among the nine methods in Table 2. Full recovers 169/185 cases (91.35%), compared with 109/185 (58.92%) for EvoWorld-Goal. The gain is 60 additional recoveries, or 32.43 percentage points, while mean steps decrease from 13.85 to 10.52. The paired outcomes contain 67 Full-only successes and seven Goal-only successes, accounting for the net gain of 60. Full recovers 32–35 of 37 checkpoints per seed (86.49–94.59%); its gain over Goal has a 95% family-bootstrap interval of +18.38 to +45.95 points (Appendix Tables D1 and H1).

The intermediate comparisons distinguish state tracking, experience access, and sustained recovery. State chain and generic reflection recover 40 and 39 cases, respectively; raw memory recovers 51. EvoWorld-RH reaches 73/185 and goal-state control reaches 109/185. Full improves further to 169/185. These comparisons show that the complete system's advantage extends beyond adding a state description, reflection, or retrieved record: the strongest result combines verified local repair with learned continuation through the remaining task.

A separate fresh natural-error set tests recovery from another collection of failure checkpoints. Full recovers 84/90 cases (93.33%), compared with 82/90 (91.11%) for Goal. Mean interaction steps decrease from 17.64 to 12.73, and mean prompt tokens decrease from 20,563 to 15,157. Appendix D.5 reports the seed-level results and efficiency measures.

5.3 Mechanism analysis

Complete-recovery supervision produces the largest learning contrast (Figure 3). On the same 37 checkpoint families and three seeds, first-action supervision recovers 49/111 cases (44.14%), whereas complete-recovery supervision recovers 101/111 (90.99%). Both execute until the same terminal criterion under the same budget. The paired outcomes contain 54 successes unique to complete-recovery supervision and two unique to first-action supervision, yielding 52 additional successful endpoints and a 46.85-point gain. Supervision of the continuation trains the decisions required after the first correction, aligning the training view with the terminal recovery objective. The gain has a 95% family-bootstrap interval of +31.53 to +61.26 points, retaining all seeds within each resampled checkpoint family (Appendix Table H1).

The independent memory control evaluates whether verified experience improves reuse. Across 30 families and three seeds, verified memory recovers 39/90 cases (43.33%); no memory and stale memory each recover 29/90 (32.22%). The associated opportunity-level analysis records 18 repeated errors in 591 opportunities (3.05%) for verified repair, compared with 306 in 715 (42.80%) for the earlier controller. The recovery comparison uses paired checkpoints, while the repeated-error comparison uses each condition's recorded opportunities. Together, they connect accepted repair experience with both higher recovery and less recurrence. The memory audit accepts one of three candidate signatures and rejects two; verified experience is activated in 15 interactions (Appendix Table D4).

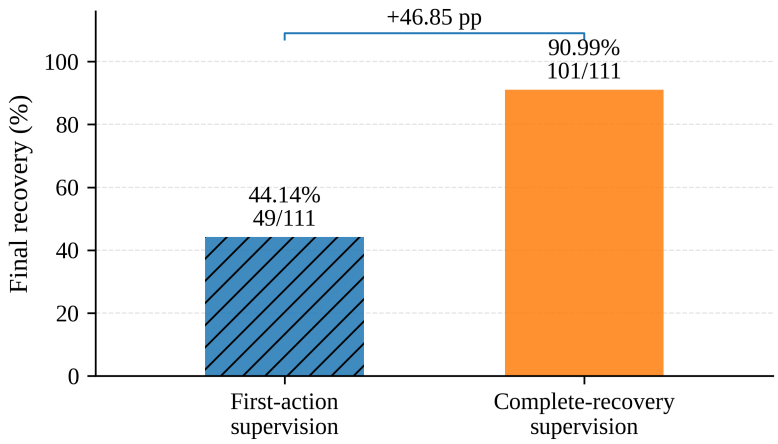


Figure 3: Supervision span and final recovery. Both settings use the same 37 checkpoint families and three seeds, for 111 outcomes. First-action supervision recovers 49/111 cases and complete-recovery supervision 101/111, a gain of 52 successful endpoints or 46.85 percentage points. Both rates measure final recovery.

Table 3: Recovery under held-out conditions.

Factor	Families	Goal	Full	Δ (pp)
Scene	4	4/12	10/12	+50.00
Object	7	7/21	17/21	+47.62
Template	3	8/9	9/9	+11.11
Error factor	26	35/78	26/78	−11.54

Each condition uses three seeds. Counts are successful recoveries / evaluations; differences are Full minus Goal.

The sealed controller comparison tests the correspondence between prediction and repair. Predicted programs recover 90/108 cases (83.33%); program-only, placebo, and shuffled controls each recover 40/108 (37.04%). Simply supplying a program is therefore insufficient in this control setting. Its diagnostic conditions must match the observed failure. The 108-record control, 90-record memory comparison, and 111-record supervision comparison address complementary components; their settings are reported separately in Appendix D.

5.4 Generalization and Training Scale

Held-out conditions. Table 3 reports all four holdouts. Full exceeds Goal on scenes, objects, and templates: 10/12 versus 4/12, 17/21 versus 7/21, and 9/9 versus 8/9, respectively. The gains are 50.00, 47.62, and 11.11 percentage points. The error-factor condition removes `container_state` content and leaves 80 training records from 12 families; Full recovers 26/78, compared with 35/78 for Goal. This contrasts transfer across layouts, entities, and templates with transfer to an error factor removed from training.

Training scale. Figure 4 uses the actual token count of each training view. Recovery is 72/111 at 26,066 tokens, 71/111 at 50,992, 102/111 at 100,234, and 101/111 at 609,793. The largest increase occurs between the approximately 50k and 100k views. The 100k and full views differ by one recovery, indicating that the main gain is already present at 100k tokens in this experiment. Table E1 in the supplement reports the corresponding record counts, family coverage, and mean steps.

Cross-backbone adaptation. For Qwen3-8B (Yang et al., 2025), the adaptation configuration is selected on development data and frozen before evaluation. Full recovers 104/111 cases (93.69%) versus 99/111 (89.19%) for Goal; mean steps are 8.53 and 9.08. The pairs contain 95 shared successes, nine Full-only successes, four Goal-only successes, and three shared failures. Full adds

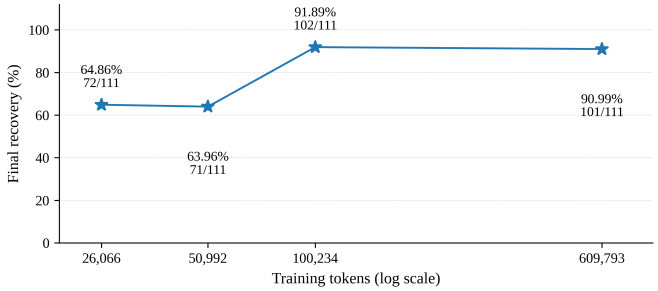


Figure 4: Recovery versus training-token budget. The four views contain 26,066, 50,992, 100,234, and 609,793 training tokens and recover 72, 71, 102, and 101 of 111 checkpoint–seed outcomes. The horizontal axis is logarithmic; lines connect the measured conditions. Record counts and family coverage are reported in Appendix Table E1.

Table 4: Task success on the frozen external benchmarks.

Method	OSWorld-361 (%)	WebArena-678 (%)
Standard	18.19 ± 0.85	18.73 ± 0.44
State-Chain	20.96 ± 0.58	21.58 ± 0.52
Reflection	22.25 ± 0.70	24.04 ± 0.59
AWM-offline	25.02 ± 0.70	26.79 ± 0.45
EvoWorld-Full	$\mathbf{28.25 \pm 0.83}$	$\mathbf{30.14 \pm 0.96}$
Full − AWM (pp)	$+3.23$	$+3.34$

Mean ± sample standard deviation across seeds 17, 29, and 43. All methods use OpenCUA-7B, with 20 steps for OSWorld and 30 for WebArena. Gains use unrounded rates.

five net recoveries (4.50 points), with a harmful-correction rate of 4/111 (3.60%). Details are in Appendices C.1 and E.3.

5.5 Browser and desktop validation

Full achieves the highest mean task success in both external environments using frozen OpenCUA-7B (Table 4). It reaches 28.25% on OSWorld and 30.14% on WebArena. AWM-offline reaches 25.02% and 26.79%, giving Full gains of 3.23 and 3.34 points; relative to Reflection, the gains are 6.00 and 6.10 points.

Full also exceeds AWM-offline in every seed: OSWorld is 102/99/105 versus 90/93/88, and WebArena is 204/198/211 versus 181/185/179. Workflows and verified repairs both reuse prior experience but organize it differently, so Full improves over the strongest experience-reuse baseline as well as direct, state-chain, and reflection policies.

Desktop uses screenshots and accessibility information, whereas browser interaction uses page candidates refreshed after changes. The shared state, expected effects, bounded continuation, and visible checks connect these interfaces; Appendices F–G give the protocol and trajectories.

6 Discussion and Conclusion

EvoWorld treats recovery as state correction supported by interaction feedback. The complete system achieves 91.35% recovery, full-recovery supervision improves over first-action supervision by 46.85 points, Qwen3-8B adaptation reaches 93.69%, and frozen OpenCUA-7B Full exceeds all four baselines on both external environments.

The experiments support three roles: localizing the falsified expectation, accepting a scoped correction from visible evidence, and carrying it through subsequent decisions. Held-out scenes, objects, and templates show transfer, while the error-factor condition identifies where more experience is needed. Reliable recovery depends on the full continuation, not only the first correction.

REPRODUCIBILITY STATEMENT

All corpus construction and family-disjoint split rules are documented in Appendix A. Appendix B specifies the repair state, objective, execution, and observable-effect validation. Appendix C gives the training, inference, method-freezing, and external configuration details. Appendix D reports the primary recovery, supervision, memory, control, and fresh-recovery results. Appendix E reports training-scale, holdout, and cross-backbone analyses. Appendix F gives the external benchmark scope, task accounting, and per-method results. Appendix G provides the retained OSWorld and WebArena interaction traces. Appendix H defines the paired gains, family-bootstrap intervals, exact tests, and cross-references. The manuscript uses fixed task lists, seeds, action limits, and terminal records; the appendix preserves the corresponding audit tables and figure evidence.

AI USE STATEMENT

OpenAI Codex was used solely to assist with LaTeX typesetting and document formatting.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. Agent S: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, 2025a. URL https://arxiv.org/abs/2410.08164.

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S2: A compositional generalist-specialist framework for computer use agents. In *Conference on Language Modeling*, 2025b. URL https://arxiv.org/abs/2504.00906.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, et al. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.13232.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. AutoManual: Constructing Instruction Manuals by LLM Agents via Interactive Environmental Learning. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0019. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/0142921fad7ef9192bd87229cdafa9d4-Abstract-Conference.html.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, et al. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, pp. 11642–11662. PMLR, 2024.

Dayuan Fu, Keqing He, Yejie Wang, et al. AgentRefine: Enhancing agent generalization through refinement tuning. In *International Conference on Learning Representations*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, et al. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.11738.

Yifei He, Pranit Chawla, Yaser Souri, Subhojit Som, and Xia Song. WebSTAR: Scalable data synthesis for computer use agents with step-level filtering. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 516–533, 2026.

Edward J. Hu, Yelong Shen, Phillip Wallis, et al. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2106.09685.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, et al. The BrowserGym ecosystem for web agent research. *Transactions on Machine Learning Research*, 2025.

Seungjoon Lee, Suhwan Kim, Minhyeon Oh, Youngsik Yoon, and Jungseul Ok. Experience-based knowledge correction for robust planning in Minecraft. In *International Conference on Learning Representations*, 2026.

Zhengxi Lu, Fei Tang, Guangyi Liu, et al. UI-Copilot: Advancing long-horizon GUI automation via tool-integrated policy optimization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pp. 19741–19762, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, et al. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Siru Ouyang, Jun Yan, I-Hung Hsu, et al. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM Web Agents via Self-Evolving Online Curriculum Reinforcement Learning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c66e1fcc9691aae706250638f36f681b-Abstract-Conference.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Arik. Learn-by-interact: A Data-Centric Framework for Self-Adaptive Agents in Realistic Environments. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/dd745a0c4f91fe91866fe6788be9cc28-Abstract-Conference.html.

Xinyuan Wang, Bowen Wang, Dunjie Lu, et al. OpenCUA: Open foundations for computer-use agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025a.

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

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, Si Qin, Lars Liden, Qingwei Lin, Huan Zhang, Tong Zhang, Jianbing Zhang, Dongmei Zhang, and Jianfeng Gao. GUI-Actor: Coordinate-Free Visual Grounding for GUI Agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0508. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/16130af940e9dabb43c726119bd3b42e-Abstract-Conference.html.

Tianbao Xie, Danyang Zhang, Jixuan Chen, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1650.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025a.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. AgentTrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, 2025b.

Yiheng Xu, Zekun Wang, Junli Wang, et al. Aguvis: Unified pure vision agents for autonomous GUI interaction. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 69772–69805. PMLR, 2025c.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Agentic plan caching: Test-time memory for fast and cost-efficient LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

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

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *International Conference on Learning Representations*, 2024.

Shuyan Zhou, Frank F. Xu, Hao Zhu, et al. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

A WED DATA VIEWS AND SPLITS

A.1 RAW INTERACTIONS AND RECOVERY VIEWS

WED organizes ALFWorld interactions into views for learning and evaluating recovery (Shridhar et al., 2021). An episode contains the observations and actions of one execution; a transition pairs an action with its preceding and following state. The raw corpus contains 10,000 episodes and 162,756 transitions. The canonical pilot contains 2,000 episodes and 30,766 transitions, and the recovery views contain 714 training records and 245 development records. Table A1 reports these processing views and their uses.

Table A1: Data views and uses

Data view	Scale	Unit and use
Initial smoke	100 / 1,462	episodes / transitions; interface and record checks
Canonical pilot	2,000 / 30,766	episodes / transitions; normalized view
Pilot training split	21,481	transitions
Pilot development split	4,546	transitions
Pilot evaluation split	4,739	transitions
Expanded raw	10,000 / 162,756	episodes / transitions; source corpus
Recovery training	714; 36 families	recovery samples; parameter learning
Recovery development	245; 9 families	recovery samples; configuration selection
Primary recovery evaluation	37 families $\times$ 5 seeds	185 checkpoint–seed pairs

A.2 FAMILY-DISJOINT PARTITIONS

For the recovery views, a family is identified by the source ALFWorld task/trial key. All checkpoints, recovery groups, and action records derived from that instance retain its `family_id`. Distinct trials can therefore belong to different source families even when they share a task template or object class. The recovery training, development, and evaluation views contain 36, 9, and 37 families, respectively, with the empty intersections reported in Table A2. The upstream `package_train` view provides an additional source-intersection check. Scene, object, template, and error-factor holdouts are defined separately in Section E.2.

Table A2: Task-family audit

Set or intersection	Families	Meaning
Training	36	Complete recovery training view
Development	9	Model/configuration selection
Primary evaluation	37	Frozen recovery checkpoints
Training $\cap$ development	0	No shared `family_id`
Training $\cap$ evaluation	0	No shared `family_id`
Development $\cap$ evaluation	0	No shared `family_id`
`package_train` $\cap$ evaluation	0	Additional source audit

Within the source groups assigned to recovery-data construction, the training/development assignment is deterministic. We hash the concatenation of `v70-train-split:` and the family identifier with SHA-256, interpret the first eight hexadecimal digits as an integer, and take the remainder modulo 10. Values 0 and 1 assign a family to development; the remaining values assign it to training. All retained records from that family inherit this assignment.

A.3 RECOVERY RECORDS AND SUPERVISION VIEWS

The recovery-view builder reads source groups marked for training-data construction, restores their recorded failure checkpoints, and replays the stored `certified_recovery_plan`. At each step, the target command must match a currently admissible environment command. The builder excludes `help`, `look`, `inventory`, and consecutive duplicate commands. It stores a retained step when

execution produces semantic progress, completes a stage, or completes the task. Each record contains the current observation, recent transitions, candidate commands, target index, source family, and recovery-group identifier. The family assignment in Section A.2 determines whether the record enters the training or development view.

A `group_id` links the records from one recovery process, and the step suffix in `example_id` determines their order. Each record identifies its task type, candidate actions, and target action or target index (Table A3).

Table A3: Fields used in recovery samples

Field	Role
`example_id`	Unique recovery step; suffix gives group order
`family_id` / `group_id`	Family split / same recovery process
`task_type`	ALFWorld task type
`candidate_actions`	Candidate actions in the record
`target_action` / `target_index`	Target action and its index

The first-action view retains the earliest corrective record in each recovery group, yielding 60 training and 15 development samples. The complete-recovery view retains the subsequent recovery steps, yielding 714 and 245 samples. Both views retain 36 training and nine development families. Table A4 makes the supervision-unit difference explicit: the complete view includes decisions encountered after the initial correction. Both variants are evaluated by continuing to the task endpoint or the same action-budget limit.

Table A4: Supervision-view composition

View	Training samples	Development samples	Train / dev. families
First-action	60	15	36 / 9
Complete-recovery	714	245	36 / 9

A.4 Checkpoints and observable inputs

A checkpoint contains the environment state and observable context at a failed execution point. For checkpoint i, the continuation budget B_i is read from its recorded `matched_step_budget`. Each compared method receives an independent restored copy, the same task and checkpoint context, and the same B_i. Every executed continuation action counts toward this limit. Evaluation ends when the task succeeds, the environment terminates, or the recorded action budget is exhausted.

The source checkpoint registry assigns 30 families to the design subset and seven to the confirmation subset. Candidate development used the design subset, and the five-seed summary includes both subsets. These registry roles are distinct from the parameter-training and development partitions reported in Table A2.

B Repair Algorithm and Learning Objective

B.1 Task-local state and online repair

AgentState stores the task identifiers, active and completed subgoals, unresolved information, and the most recent eight actions and observation fingerprints. It also records the diagnosis, repair scope, repair program, expected effect, and recovery counters. Table B1 specifies when these fields are updated.

The controller compares the preceding action's expected effect with the new observation. A contradiction or repeated lack of progress triggers a scoped repair. Otherwise, execution continues toward the current subgoal. A repair advances one executable step at a time, with another observation before the next decision. The following pseudocode summarizes this interaction sequence.

Table B1: AgentState fields and update timing

Field group	Content	Update timing
Task identity	`goal`, `method`, `environment`	Episode initialization
Short history	`recent_actions`, `observation_fingerprints`	After each action and observation; at most eight
Goal state	`completed_subgoals`, `active_subgoal`	When target-related evidence is visible
Recovery state	`diagnosis`, `repair_scope`, `repair_program`	On contradiction or persistent no progress
Check basis	`expected_effect`, `last_model_state`	When an action is generated and feedback is read
Control counters	`loop_count`, `recovery_events`, `verified_repairs_used`	When the corresponding event occurs

Algorithm B1: Online verified repair.
Input: task goal g, initial observation o, frozen development experience M, and environment-action budget B.
Initialize: task-local state, short history, and repair counters.

1. While execution continues and the action budget remains:
   (a) Update visible state and subgoal status from the current observation.
   (b) Compare the previous expected effect with the new observation.
   (c) If a contradiction or persistent lack of progress is detected, form a diagnosis, a scope, and a short repair program.
   (d) Select the next task action or the current repair step and apply the environment-specific action contract.
   (e) Execute one accepted action and obtain a new observation.
   (f) Check its target-related effect, then advance, re-localize, resume task execution, or terminate under the visible checks and bounded control rules.
2. Evaluate the terminal task outcome after interaction.

Output: terminal outcome and interaction record.

B.2 Verified experience construction

Development interactions supply candidate repair records. For each candidate, we inspect the pre-action expectation, the observed mismatch, the executed repair, and its target-related visible effect. Acceptance requires an observation-supported trigger, a defined repair scope, and evidence that the correction preserves completed progress. Accepted records are stored as `scope`, `signal`, `repair`, and `verification`.

Table B2 illustrates these fields for opening a target dialog. The verification condition specifies the observation that completes the repair; locating the control supports the next opening step. External evaluation reads at most the first 12 records in their fixed file order. Task-local state changes with interaction, while this cross-task experience file remains frozen between episodes.

Table B2: Illustrative verified-memory entry

Field	Example
scope	Target localization
signal	Expected dialog absent / no progress
repair	Locate the target; open it; inspect the resulting view
verification	Target dialog is visible in the new observation

B.3 RECOVERY-ACTION SCORING

For record i, the scorer assigns a value to each candidate action under the observable context h_i. The target action is compared with at most five negative candidates, prioritizing candidates with the same action verb. The ranking term uses margin 0.65. The per-record objective combines this term with the target-action score at weights 0.80 and 0.20, respectively.

$$L_{\mathrm{rank},i} = \mathrm{mean}_{a^- \in N_i} \,\mathrm{softplus}\left[0.65 - s_\theta(h_i, a_i^+) + s_\theta(h_i, a^-)\right], \tag{B.1}$$

$$L_i = w_i \left(0.80 L_{\mathrm{rank},i} + 0.20[-s_\theta(h_i, a_i^+)]\right). \tag{B.2}$$

The task-type multiplier is 1.35 for heating, 1.15 for cleaning and cooling, 1.10 for two-object tasks, and 1.00 otherwise. First-action training applies the objective to the first corrective record of each group; complete-recovery training applies it throughout the retained successful suffix. The terminal evaluation is shared, so the experimental comparison concerns recovery across the subsequent decisions.

B.4 EXECUTION AND EFFECT VALIDATION

The action parser establishes whether a proposed operation is executable. The visible checker tests its target-related effect, using changes in windows, text, focus, URLs, scroll position, or visible entities. Screenshot fingerprints provide a change cue. Terminal evaluation measures completion of the task after interaction.

External runs separately record effect checks, observed changes, WAIT, format retries, and executed environment actions. Visible completion evidence permits DONE. Desktop recovery permits at most three bounded relocations when seeking an executable alternative; executed recovery actions use the original environment budget. Environment-acceptance operations and model-generated trajectories are recorded separately.

C MODELS, TRAINING, AND INFERENCE

C.1 CORE TRAINING AND DEVELOPMENT SELECTION

Core ALFWorld recovery uses OpenCUA-7B with LoRA and AdamW. Recovery-specific optimization starts from the existing policy checkpoint for the corresponding seed and performs 480 parameter updates. Each update uses one recovery record together with its candidate actions; the complete-recovery training and development views contain 714 and 245 records.

Checkpoints are evaluated before recovery-specific optimization and after updates 240 and 480. Selection first compares development action accuracy on heating, cleaning, and cooling tasks, then overall development action accuracy. A checkpoint replaces the current selection only when this ordered pair strictly improves; ties retain the earlier selection. Table C1 gives the remaining optimization settings.

For Qwen3-8B (Yang et al., 2025), three adaptation configurations are compared on the development view (Table C2). The selected `mid_short` configuration uses 320 updates, learning rate 10^{-5}, and rank 16 with alpha 32. Its development action accuracy is 167/245 (68.16%). This configuration is then fixed for the three-seed recovery comparison reported in Section E.3.

The first-action baseline is trained for 480 steps per seed. Development selection retains the checkpoint at step 240 for seed 17 and the initial checkpoint for seeds 29 and 43. These selections achieve 100% action accuracy on the 15-record development view; their recovery continuations produce 49 successes in 111 evaluations. Development action accuracy and terminal recovery are reported separately. The initial checkpoint is the inherited policy checkpoint at update 0. Update 0 denotes the policy checkpoint before the recovery-specific optimization.

C.2 FROZEN EXTERNAL CONFIGURATION

Desktop and browser experiments use frozen OpenCUA-7B weights (Wang et al., 2025a), temperature 0, and seeds 17, 29, and 43 on eight A100 GPUs. OSWorld allows 20 environment steps and

Table C1: Core recovery-training configuration

Item	Setting
Core backbone	OpenCUA-7B
Initialization	Corresponding-seed policy checkpoint
Training seeds	17, 29, 43, 71, 101
Updates per training seed	480
Checkpoint checks	Updates 0, 240, and 480
Selection criterion	Transformation-task accuracy, then overall accuracy
Tie rule	Retain the earlier checkpoint
Recovery action limit	Checkpoint-specific `matched_step_budget`
Optimizer	AdamW; weight_decay=0; foreach=False
Default learning rate	1.5×10^{-5}
Maximum encoding length	2,048 tokens
Update unit	One recovery sample and candidate actions
Negative candidates	At most 5; same verb preferred
Loss	0.80 ranking + 0.20 target-action
Ranking margin	0.65
Gradient clipping	Maximum norm 1.0
Precision	bfloat16 when supported, else float16; gradient checkpointing
New LoRA	rank 16, alpha 32, dropout 0, bias=none
Target modules	Existing q/k/v/o and gate/up/down projections

Table C2: Qwen3-8B development selection

Configuration	Steps	Learning rate	Rank / alpha	Dev. acc. (%)	Decision
low_short	240	5e-6	8/16	44.49%	Not used
mid_short	320	1e-5	16/32	68.16%	Used
low_full	480	5e-6	8/16	60.82%	Not used

WebArena 30. Both interfaces use a default 512-token output limit and a 240-second model-request timeout. Table C3 lists the shared settings and the environment-specific observations and action parsers.

Table C3: External evaluation settings

Setting	OSWorld	WebArena
Model	OpenCUA-7B	OpenCUA-7B
Seeds	17, 29, 43	17, 29, 43
Maximum environment steps	20	30
Output limit	512 tokens	512 tokens; local retry may be lower
Request timeout	240 s	240 s
Temperature	0.0	0.0
Observation	Screenshot and available a11y	Screenshot and visible DOM candidates
Action interface	PyAutoGUI whitelist parser	BrowserGym function calls
Format repair	One contract repair	At most one format retry
New parameter training	None	None
Cross-task experience	Frozen development JSON (Full)	Frozen development JSON (Full)

C.3 External comparison methods

The five external methods share the backbone, task scope, observation interface, executable-action adapter, and terminal evaluator. Standard selects an action from the current observation. State-Chain additionally maintains location, visible entities, and subgoal status. Reflection uses recent failure or repetition to produce a short reflection. AWM-offline supplies reusable workflow context (Wang

et al., 2025b), whereas Full supplies verified repair context and uses the expected-effect, scoped-repair, and visible-check loop. Table C4 summarizes these decision and experience units.

Table C4: Differences among external methods

Method	Task-local control	Cross-task context
Standard	Next action from current observation	None
State-Chain	Location, entities, completed and active subgoals	None
Reflection	Short reflection after recent failure/repetition	None
AWM-offline	Reusable workflow guides actions	Offline workflow
EvoWorld-Full	Expected–observation comparison, scoped repair, visible verification	scope/signal/repair/verification

C.4 Observation and action contracts

OSWorld actions use a whitelisted desktop parser; WebArena actions use BrowserGym function calls (Le Sellier de Chezelles et al., 2025). Desktop coordinates are mapped once according to the screenshot protocol. Browser element references are regenerated after page changes. Each executable action is associated with the current observation and expected effect. Invalid responses receive the bounded format-repair procedure; WAIT, rejected outputs, and executed actions have separate counters.

C.5 Configuration freezing and repeated runs

Development data select the training checkpoint or inference configuration. Formal evaluations reuse the selected settings on the fixed task list. Core training seeds index repeated training runs; external seeds index repeated evaluations with the same frozen model and temperature-zero decoding. Results are aligned by benchmark, task or checkpoint, method, and seed before aggregation.

Core ALFWorld methods. The three EvoWorld variants in the primary recovery table differ in their continuation policy. RH uses receding-horizon repair with a persistent program and an execution ledger. Goal organizes continuation around the remaining goal conditions and observed task-stage progress. Full uses the complete-recovery specialist to rank stage-compatible candidate actions, retaining the observable-state controller and semantic action priorities. Table C5 links these policies to the primary comparison. All use the same checkpoint-level action limits and terminal task criterion.

Table C5: Core ALFWorld recovery variants.

Method	Continuation policy	Main control state
EvoWorld-RH	Receding-horizon repair using the existing policy	Persistent repair program and execution ledger
EvoWorld-Goal	Goal-state-guided action selection	Completed stages, current stage, and remaining goal conditions
EvoWorld-Full	Complete-recovery specialist with stage-compatible action selection	Observable task state, completed progress, and semantic action priorities

D Expanded ALFWorld Results

D.1 Primary comparison and seed stability

The primary comparison evaluates 37 checkpoint families with seeds 17, 29, 43, 71, and 101. Full recovers 32, 34, 35, 35, and 33 cases, respectively, for 169/185 overall. Table D1 shows that all five runs achieve at least 86.49% recovery.

Table D1: Full recovery by seed

Seed	Successes / total	Recovery (%)
17	32/37	86.49
29	34/37	91.89
43	35/37	94.59
71	35/37	94.59
101	33/37	89.19
Total	169/185	91.35

Table D2 places these results alongside the eight comparison methods. Full has both the highest recovery rate and the lowest mean step count. Paired with Goal, it produces 67 Full-only successes and seven Goal-only successes, a net gain of 60. Section H.2 reports the family-level interval and the paired calculation.

Table D2: All primary recovery comparisons

Method	Successes / total	Recovery (%)	Mean steps
No repair	32/185	17.30	16.79
ReAct	43/185	23.24	16.12
Generic reflection	39/185	21.08	16.29
State chain	40/185	21.62	16.23
Retrieved insight	36/185	19.46	16.78
Raw memory	51/185	27.57	16.21
EvoWorld-RH	73/185	39.46	16.04
EvoWorld-Goal	109/185	58.92	13.85
EvoWorld-Full	169/185	91.35	10.52

D.2 First-action and complete-recovery supervision

Table D3 reports the supervision comparison on the same 37 checkpoint families with seeds 17, 29, and 43. Complete-recovery supervision is higher in every run. The aggregate difference is 52 successful endpoints, or 46.85 percentage points.

Table D3: First-action versus complete-recovery supervision

Setting	Seed 17	Seed 29	Seed 43	Total / recovery
First-action	16/37	18/37	15/37	49/111; 44.14%
Complete-recovery	32/37	34/37	35/37	101/111; 90.99%

The complete-recovery view exposes the learner to later decisions after the first corrective action. Evaluation continues to the terminal criterion for both methods. The result therefore measures completion of the recovery suffix, while the record counts in Table A4 describe the corresponding difference in training supervision.

D.3 Verified memory and error recurrence

The leave-family-out memory control evaluates 30 families with three seeds. No memory and stale memory each recover 29/90 cases, while verified memory recovers 39/90. Table D4 also reports the acceptance process: three candidate signatures yield one accepted record and two rejections, with verified experience activated in 15 interactions. This control records no stale-experience false acceptance or negative-transfer event.

Figure D1 compares recovery under the three memory conditions. The recurrence analysis records 306 repeated errors in 715 opportunities for the earlier controller, versus 18 in 591 for verified repair. Figure D2 shows the corresponding rates, 42.80% and 3.05%. Each rate uses its own run's recorded

Table D4: Verified memory and experience acceptance

Item	Value
No-memory recovery	29/90; 32.22%
Stale-memory recovery	29/90; 32.22%
Verified-memory recovery	39/90; 43.33%
Candidate / accepted / rejected signatures	3 / 1 / 2
Verified experiences activated	15
Stale experience falsely accepted	0
Negative-transfer records	0

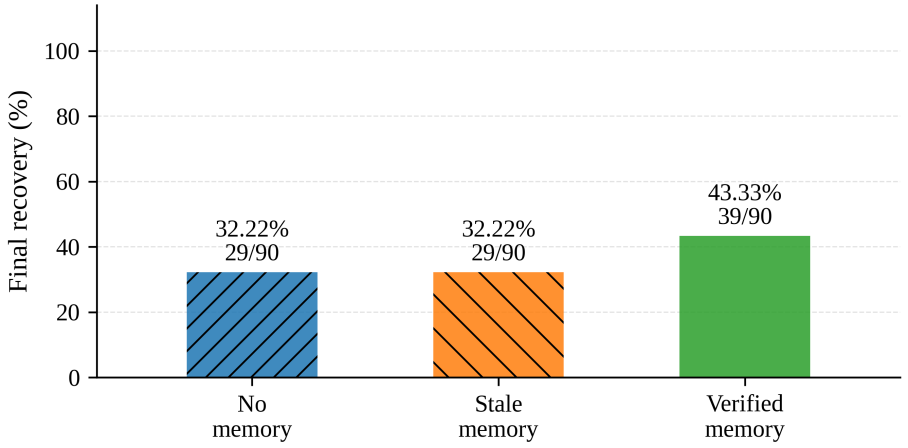


Figure D1: Recovery under three memory conditions on 90 paired checkpoint–seed outcomes. Verified memory recovers 39 cases, compared with 29 for each of the no-memory and stale-memory controls.

opportunities. Together, the comparisons characterize recovery and recurrence under verified experience reuse.

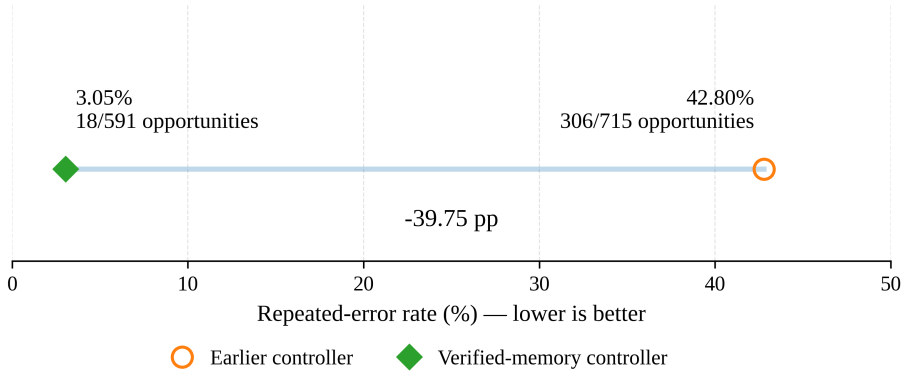


Figure D2: Repeated-error opportunity rate. The earlier controller repeats 306 errors in 715 opportunities; verified repair repeats 18 in 591. Each rate uses the opportunities recorded for its own run.

D.4 Sealed repair and action-condition controls

The independent sealed comparison evaluates 36 families with three seeds. Predicted repair programs recover 90/108 cases; program-only, placebo, and shuffled-program controls each recover 40/108 (Table D5). Figure D3 visualizes the gap. The comparison tests the relation between the supplied diagnostic conditions and the repair program under the same checkpoint protocol.

Table D5: Independent sealed controller confirmation

Condition	Successes / total	Recovery (%)
Predicted program	90/108	83.33
Program-only	40/108	37.04
Placebo program	40/108	37.04
Shuffled program	40/108	37.04

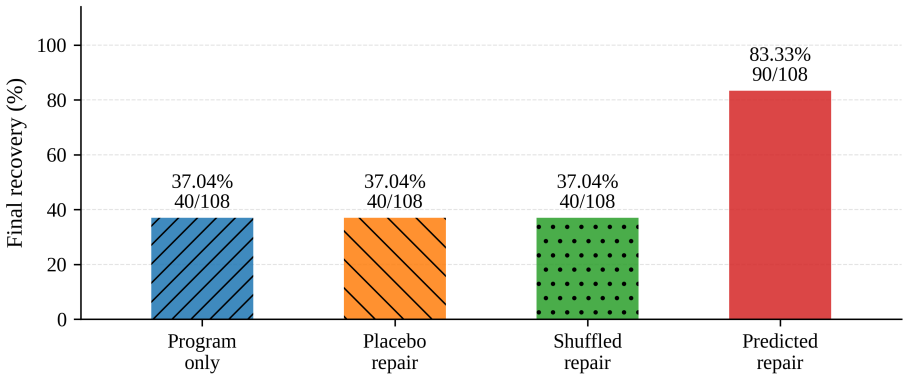


Figure D3: Independent sealed repair controls on 108 checkpoint–seed outcomes. Predicted programs recover 90 cases, whereas program-only, placebo, and shuffled-program controls each recover 40.

A separate structured-interface audit compares true action-conditioned input with shuffled-action and state-only input. The recorded F1 values are 1.00, 0, and 0, respectively. This audit measures action-conditioned correspondence in the structured interface; Table D5 measures terminal recovery after a program is executed.

D.5 Fresh natural errors and task slices

The fresh natural-error comparison evaluates 30 checkpoints over three seeds. Goal recovers 28, 27, and 27 cases; Full recovers 28 in every run (Table D6). Their totals are 82/90 and 84/90. Mean steps decrease from 17.64 to 12.73, and mean prompt tokens from 20,563.04 to 15,157.28. Figure D4 displays final recovery, while the table retains the per-seed and efficiency measures.

Table D6: Fresh natural-error recovery and efficiency

Method	Seed 17	Seed 29	Seed 43	Total	Mean steps	Mean prompt tokens
Goal	28/30	27/30	27/30	82/90	17.64	20,563.04
Full	28/30	28/30	28/30	84/90	12.73	15,157.28

Table D7 reports additional conditions. The held-out heating-training comparison gives 35/42 recoveries for the reported strategy and 15/42 for its control. The long-horizon and two-object slices give 35/35 and 10/10, respectively. These slices retain their own family and seed counts rather than adding new observations to the primary 185-result total.

D.6 Initial-state task execution

Table D8 evaluates execution from the initial task state rather than from a failure checkpoint. The table contains 120 in-distribution outcomes and 180 out-of-distribution outcomes. The explicit-working-memory controller is the separately evaluated structured controller shown in the table. Its initial-state results and the checkpoint-recovery results address different execution settings.

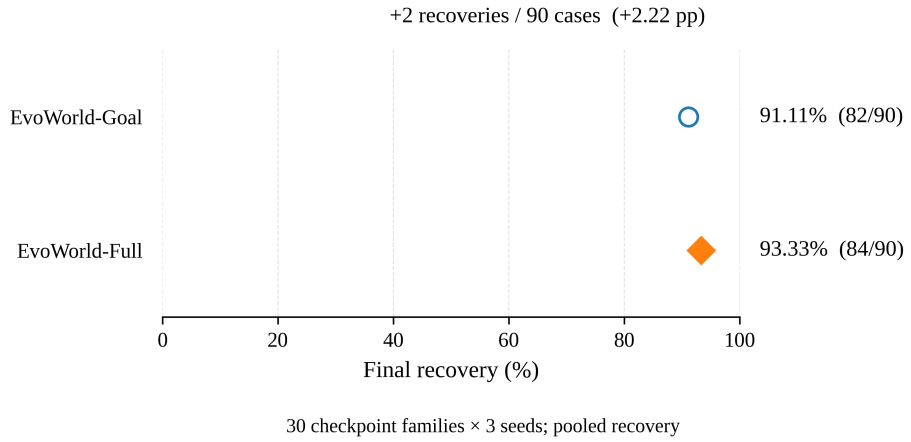


Figure D4: Recovery on fresh natural-error checkpoints. Full and Goal recover 84/90 and 82/90 cases, respectively, across three seeds. Table D6 reports the seed-level and efficiency measures.

Table D7: Task slices and extra recovery conditions

Condition	Families × seeds	Full / strategy	Control
Held-out heating training	14×3	35/42; 83.33%	15/42; 35.71%
Long-horizon slice	7×5	35/35; 100%	–
Two-object slice	2×5	10/10; 100%	–

Table D8: Independent initial-state evaluation

Method	ID: success / 120	OOD: success / 180
Direct	11 (9.17%)	30 (16.67%)
Generic reflection	20 (16.67%)	46 (25.56%)
Raw memory	27 (22.50%)	55 (30.56%)
ReAct	18 (15.00%)	38 (21.11%)
Retrieved insight	23 (19.17%)	40 (22.22%)
State chain	23 (19.17%)	44 (24.44%)
Explicit working memory controller	120 (100%)	180 (100%)

E Training Scale, Generalization, and Cross-Backbone Adaptation

E.1 Training-view size and coverage

The four training views contain 26,066, 50,992, 100,234, and 609,793 tokens. Their record and family counts are 31/31, 60/36, 122/36, and 714/36, respectively (Table E1). All are evaluated on 37 checkpoint families and three seeds. Recovery reaches 72/111, 71/111, 102/111, and 101/111, with mean steps of 12.70, 13.31, 10.73, and 10.59.

Table E1: Training scale and final recovery

Budget	Actual tokens	Samples / families	Success / 111	Recovery (%)	Mean steps
25k	26,066	31/31	72	64.86	12.70
50k	50,992	60/36	71	63.96	13.31
100k	100,234	122/36	102	91.89	10.73
Full	609,793	714/36	101	90.99	10.59

The change from roughly 50k to 100k tokens adds 31 recoveries. The 100k and full views differ by one recovery, so the four observed budgets show most of the improvement by 100k tokens. The smallest view also covers fewer training families; Table E1 retains this coverage information alongside token count. Figure 4 in the main paper plots the measured budgets.

E.2 HELD-OUT FACTORS

Each holdout removes the specified content from the training and development views before evaluating the corresponding held-out families. Table E2 records the keys and resulting view sizes: layout 11 for the scene condition, `mug` for object class, `pick_cool_then_place_in_recep|cabinet` for the template condition, and `container_state` for the critical-error-factor condition.

Table E2: Held-out views

Condition	Held-out key	Train records / families	Dev. records / families	Eval. families
Scene	layout_key / 11	688/35	219/8	4
Object	object_class / mug	649/34	245/9	7
Template	`template_key: pick_cool_then_place_in_recep|cabinet`	694/35	230/8	3
Error factor	critical_fact_type / container_state	80/12	18/2	26

Full recovers 10/12, 17/21, and 9/9 cases under the first three conditions, compared with 4/12, 7/21, and 8/9 for Goal. Under the error-factor holdout, Full recovers 26/78 and Goal 35/78. This condition leaves 80 training records from 12 families and 18 development records from two families. Figure E1 presents all four comparisons. The observed contrast is between transfer across layouts, entities, and templates and transfer after a critical error factor is removed from the recovery data.

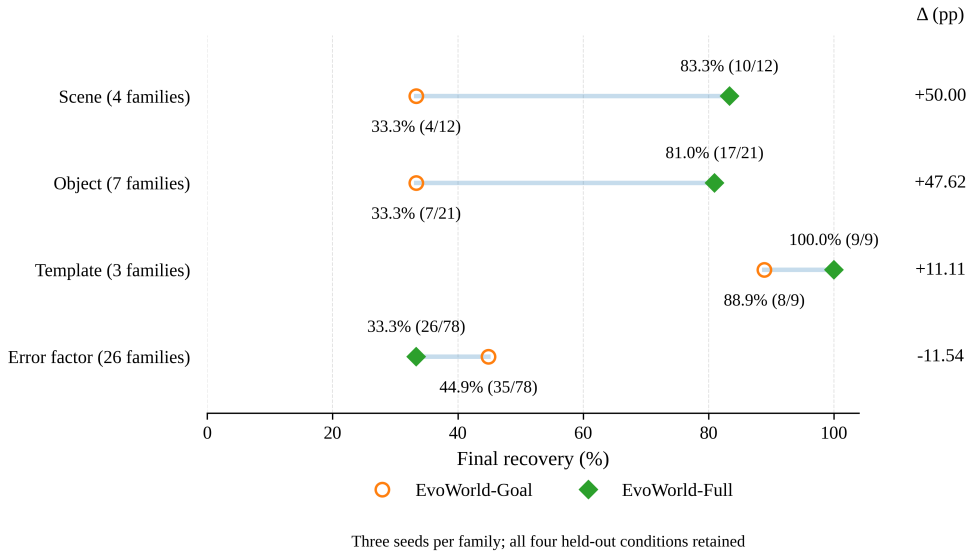


Figure E1: Recovery under four held-out factors. Full and Goal use the same checkpoints and seeds within each condition. The scene, object, template, and error-factor comparisons retain their respective denominators.

E.3 CROSS-BACKBONE RECOVERY

The Qwen3-8B configuration selected in Section C.1 is frozen before evaluation on 37 families and three seeds. Full recovers 104/111 cases (93.69%), compared with 99/111 (89.19%) for Goal. Mean steps are 8.53 and 9.08.

Table E3 gives the complete pairing: 95 shared successes, nine Full-only successes, four Goal-only successes, and three shared failures. The net gain is five recoveries, or 4.50 percentage points. Harmful correction is the Goal-only category and constitutes 4/111 (3.60%) of all paired outcomes, matching the cross-backbone result in the abstract.

Table E3: Qwen3-8B paired terminal outcomes

Outcome	Count	Share of 111 pairs
Both succeed	95	85.59%
Full succeeds, Goal fails	9	8.11%
Goal succeeds, Full fails	4	3.60%
Both fail	3	2.70%

E.4 Additional controls

Hierarchical skill fallback and the observable compiler/verifier each obtain 32/37 recoveries at seed 17, equal to the corresponding base strategy. They are not included in the final Full configuration. The no-budget control and complete configuration both recover 101/111 across seeds 17, 29, and 43 (Table E4). These comparisons retain the base strategy as the final configuration without adding components that produced no extra recoveries in these runs.

Table E4: Additional controls not used in final Full

Condition	Seed range	Result	Base result
Hierarchical skill fallback	17	32/37	32/37
Observable compiler/verifier	17	32/37	32/37
No-budget	17, 29, 43	101/111	101/111

F Desktop and Browser Evaluation

F.1 Task scope and terminal scoring

External evaluation starts from the initial task state in fixed OSWorld and WebArena task views (Xie et al., 2024; Zhou et al., 2024). Removing eight Google Drive/OAuth-dependent OSWorld tasks leaves 361 tasks. Removing Map- or Wikipedia-dependent WebArena tasks leaves 678. Five methods and three seeds yield 5,415 and 10,170 task–method–seed keys, respectively (Table F1).

Table F1: External task scope and repeated-run scale

Item	OSWorld	WebArena
Source tasks	369	812
Final tasks	361	678
Exclusion rule	8 Google Drive/OAuth tasks	Map or Wikipedia dependency
Methods	5	5
Seeds per method	3	3
Evaluation keys	5,415	10,170
Maximum environment steps	20	30
Final metric	Programmatic task success	Programmatic task success

The terminal evaluator assigns programmatic task success. Valid task failures remain in the fixed task set. Environment anomalies and missing records have separate entries; aggregation aligns the benchmark, task identifier, method, and seed before computing each rate.

F.2 Results across seeds

Table F2 reports OSWorld outcomes for the five methods. Full ranks first in every seed, followed by AWM-offline; its mean task success is 28.25%. The table retains the success counts and the common denominator of 361 tasks.

Table F2: OSWorld five-method results

Method	Seed 17	Seed 29	Seed 43	Mean ± SD (%)
Standard	65/361	69/361	63/361	18.19 ± 0.85
State-Chain	74/361	78/361	75/361	20.96 ± 0.58
Reflection	80/361	83/361	78/361	22.25 ± 0.70
AWM-offline	90/361	93/361	88/361	25.02 ± 0.70
Full	102/361	99/361	105/361	28.25 ± 0.83

Table F3 gives the corresponding results on 678 WebArena tasks. Full again ranks first in every seed, with a mean success rate of 30.14%. Both tables summarize seed-level success rates using the mean and sample standard deviation. The comparisons against each baseline are expanded in Table F4.

Table F3: WebArena five-method results

Method	Seed 17	Seed 29	Seed 43	Mean ± SD (%)
Standard	124/678	130/678	127/678	18.73 ± 0.44
State-Chain	146/678	150/678	143/678	21.58 ± 0.52
Reflection	163/678	167/678	159/678	24.04 ± 0.59
AWM-offline	181/678	185/678	179/678	26.79 ± 0.45
Full	204/678	198/678	211/678	30.14 ± 0.96

F.3 Comparisons with the baselines

Table F4 reports Full minus each baseline within each seed. Against AWM-offline, the OSWorld gains range from 1.66 to 4.71 percentage points and the WebArena gains from 1.92 to 4.72. The mean differences are 3.23 and 3.34 points. Full also exceeds Standard, State-Chain, and Reflection in every reported seed. The comparison therefore retains its direction against the strongest evaluated workflow-memory baseline as well as the task-local alternatives.

Table F4: Full improvement over each baseline (percentage points)

Environment	Baseline	Seed 17	Seed 29	Seed 43	Mean
OSWorld	Standard	+10.25	+8.31	+11.63	+10.06
OSWorld	State-Chain	+7.76	+5.82	+8.31	+7.29
OSWorld	Reflection	+6.09	+4.43	+7.48	+6.00
OSWorld	AWM-offline	+3.32	+1.66	+4.71	+3.23
WebArena	Standard	+11.80	+10.03	+12.39	+11.41
WebArena	State-Chain	+8.55	+7.08	+10.03	+8.55
WebArena	Reflection	+6.05	+4.57	+7.67	+6.10
WebArena	AWM-offline	+3.39	+1.92	+4.72	+3.34

F.4 Interfaces and frozen experience

Desktop interaction involves windows, focus, controls, and document or file state. Browser interaction uses current page content and visible element references, including lists, sorting, item details, and cross-site transitions. Table F5 maps these observable objects to their repair checks. The common state, repair, and verification loop connects the two interfaces.

The development experience file is fixed between formal episodes, while task-local state changes with observations. Interface debugging and candidate comparisons are recorded as development use. Cross-server aggregation uses the same benchmark–task–method–seed key and configuration, preserving the common task range for each method.

Table F5: Observable objects and repair focus

Environment object	Main visible information	Repair or check focus
Desktop application	Screenshot, window/focus, available a11y	Target control, dialog, document or file state
Shopping Admin	Page fields, menus, save state	Object localization and target-field modification
Shopping	Search, sort, list, detail	Target item and page state
Reddit	Forum, sort, post, page text	Whether target appears after sorting and detail matches
GitLab	Project, repository, page, item	Target object and page state
Cross-app / cross-site	Context before and after switching	State inheritance and goal preservation

G Interaction Traces

G.1 Desktop observations

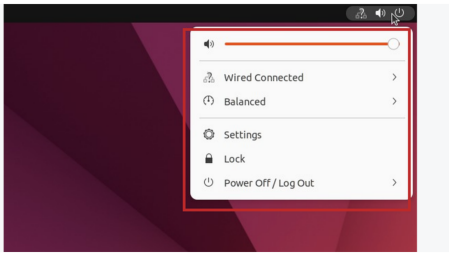
Figure G1 shows the supplied OSWorld observations at steps 1, 2, 3, and 20. The first two frames show the system tray and the opened system menu with a visible volume slider; the later frames show the Home icon and the final desktop observation. The episode carries an offline terminal score of 1.0. The figure preserves the actual step numbers and visible interface states.

a Step 1 | Initial attempt



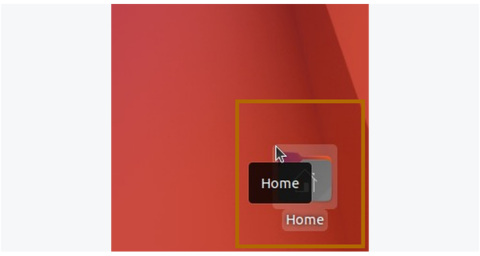
System-tray region in the initial observation.

b Step 2 | System menu opened



The volume slider is visible in the system menu.

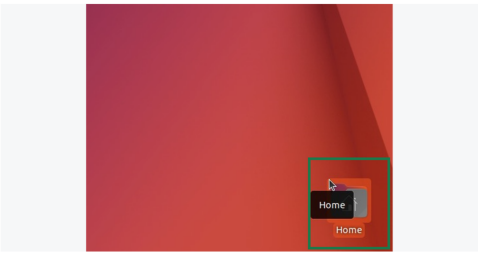
c Step 3 | Home icon observed



Home icon in the observed desktop region.

Offline final score: 1.0, as reported in the supplied case annotation.

d Step 20 | Final observed state



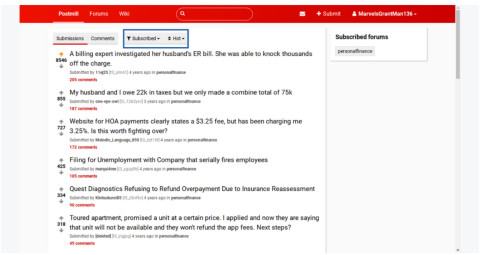
Home icon in the final supplied frame.

Figure G1: OSWorld observations at steps 1, 2, 3, and 20. The system menu contains a volume slider; later frames show Home. The offline terminal score is 1.0.

G.2 Browser navigation

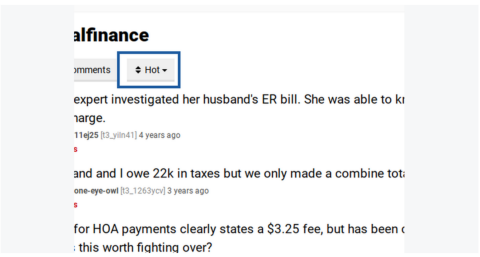
Figure G2 shows steps 1, 2, 4, and 5: Hot-to-New sorting reveals the target post, followed by its detail page. The offline terminal score is 1.0.

a Step 1 | Initial ranking (Hot)



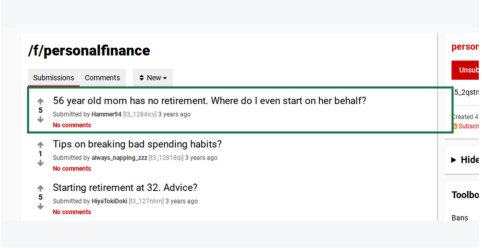
Forum feed in the initial Hot order.

b Step 2 | Inspect sorting



Sorting control in the supplied observation.

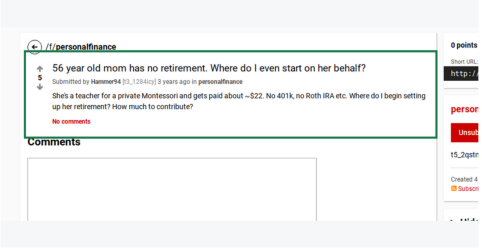
c Step 4 | Target found (New)



Target post visible in the New order.

Offline final score: 1.0, as reported in the supplied case annotation.

d Step 5 | Target post opened



The selected post is open in its detail view.

Figure G2: WebArena navigation at steps 1, 2, 4, and 5. Hot-to-New sorting reveals the target post, followed by its detail page. The offline terminal score is 1.0.

H Statistical Definitions and Paired Results

H.1 Success rates and seed variation

For each method and seed s, let S_s be the number of successful outcomes among N fixed starting points. We report the mean of the K seed-level rates and their sample standard deviation. Because each seed uses the same task denominator, the mean rate equals the pooled success proportion for that comparison. Percentage-point gains are computed from the unrounded rates.

$$p_s = 100\frac{S_s}{N}, \qquad \bar{p} = \frac{1}{K}\sum_{s=1}^{K} p_s, \qquad \mathrm{SD} = \sqrt{\frac{1}{K-1}\sum_{s=1}^{K}(p_s - \bar{p})^2}. \tag{H.1}$$

H.2 Paired gains and family resampling

Let R count paired outcomes in which the evaluated method succeeds and its comparator fails, and H count the reverse outcomes. For N matched checkpoint–seed outcomes, the gain is $100(R-H)/N$ percentage points and the harmful-correction rate is $100H/N$. Each comparison is paired on the source checkpoint family and seed.

We summarize uncertainty with checkpoint-family bootstrap intervals. Each resample draws the same number of families with replacement and retains all seed-level paired outcomes from every sampled family. The success-rate difference is recomputed from these paired outcomes. We report the 2.5th and 97.5th percentile endpoints of 2,500 bootstrap differences as a 95% interval. Table H1 retains the paired counts, gains, intervals, and pair-level exact calculations.

Table H1: Paired gains, family-bootstrap intervals, and pair-level exact tests.

Comparison	N	R/H	Δ (pp)	95% family-bootstrap interval (pp)	Pair-level exact p
Primary Full − Goal	185	67 / 7	+32.43	[+18.38, +45.95]	2.12e-13
Complete − first-step supervision	111	54 / 2	+46.85	[+31.53, +61.26]	4.433e-14
Fresh Full − Goal	90	2 / 0	+2.22	[0.00, +5.56]	0.5
Scene holdout	12	7 / 1	+50.00	[0.00, +100.00]	0.07031
Object holdout	21	10 / 0	+47.62	[+14.29, +76.19]	0.001953
Template holdout	9	1 / 0	+11.11	[0.00, +33.33]	1
Error-factor holdout	78	3 / 12	-11.54	[-23.08, -1.28]	0.03516
Qwen3 adaptation	111	9 / 4	+4.50	[-5.41, +15.32]	0.2668

Within the primary five-seed comparison, bootstrap random seed 7001 is used. The first-action and fresh-recovery comparisons use seed 7101, and the held-out-factor and Qwen3 comparisons use seed 7201. These are resampling seeds, distinct from the training and evaluation seeds. The primary implementation indexes clusters by `group_id`; for its 37 checkpoints, group and family identifiers are in one-to-one correspondence. The interval endpoints use zero-based sorted positions $\lfloor 0.025(B-1) \rfloor$ and $\lfloor 0.975(B-1) \rfloor$, with $B = 2500$.

The pair-level exact calculation applies a two-sided binomial test with probability $1/2$ to the $R + H$ discordant outcomes. Its p-values are unadjusted and do not account for dependence between seeds from the same family. Statistical interpretation therefore uses the family-bootstrap intervals; the exact column reports the accompanying discordant-pair calculation.

The intervals describe variation across checkpoint families under the reported evaluation protocol. The primary recovery and supervision comparisons have positive lower endpoints. Fresh recovery, template transfer, and Qwen3 adaptation are reported through their observed recovery rates, paired gains, and corresponding family intervals. External experiments retain the seed-level means and sample standard deviations in Tables F2–F3.

Intervals resample checkpoint families with all within-family seed records retained; exact p values use discordant pairs and are unadjusted.

H.3 Result cross-references

Tables D1–D8 detail recovery and controls; Tables C2 and E3 link development selection to cross-backbone recovery. Tables E1–E2 cover scale and holdouts, Tables F2–F4 report external runs, and Figures G1–G2 show the trajectories.