Lost in Recovery: Demystifying Recovery Behavior of Computer-Use Agents via Goal-Oriented Action Replay

Anonymous ACL submission

Abstract

Recent advances in large (vision) language models have enabled computer-use agents that automate digital workflows via fine-grained keyboard and mouse operations. However, such agents remain fragile in long-horizon settings, where early mistakes accumulate and propagate, and executed actions may leave unintended side effects on the environment, making the ability to revert to a previously safe state critical. Yet, existing heuristic remedies fall short: environment resets are infeasible in proprietary settings, action replay is prohibitively costly over long horizons, and API-based rollback is confined to web environments, leaving model-based recovery largely uncharacterized. To address this, we present RETRACE, a scalable pipeline that conducts the first systematic study of recovery behavior in computer-use agents. Inspired by predictive state representation theory, we formalize successful recovery as *Transition-Consistent Recovery*, where a recovered state matches its target if identical downstream actions yield identical outcomes, and operationalize this via goal-oriented action replay that injects controlled errors and tests whether agents can complete the original task from the recovered state. Across closed and open-source agents on two benchmarks, we find that current models lack both the tendency and capability to autonomously recover, while explicit model-based recovery emerges as a promising direction that translates into measurable end-to-end gains, positioning recovery as a key step toward reliable computer-use agents.

1 Introduction

The digital computing environment has become the primary interface through which users interact with software, supporting a wide spectrum of tasks ranging from simple actions (e.g., website logins) to complex workflows (automated document editing). Recent advances in large (vision) language models (Anthropic, 2026; Bai et al., 2025; Singh et al., 2025; DeepSeek-AI et al., 2025) have made it possible to automate such workflows through *computer-use agents*, which act in digital environments via fine-grained keyboard and mouse inputs (Gur et al., 2023; Furuta et al., 2023; Sager et al., 2025).

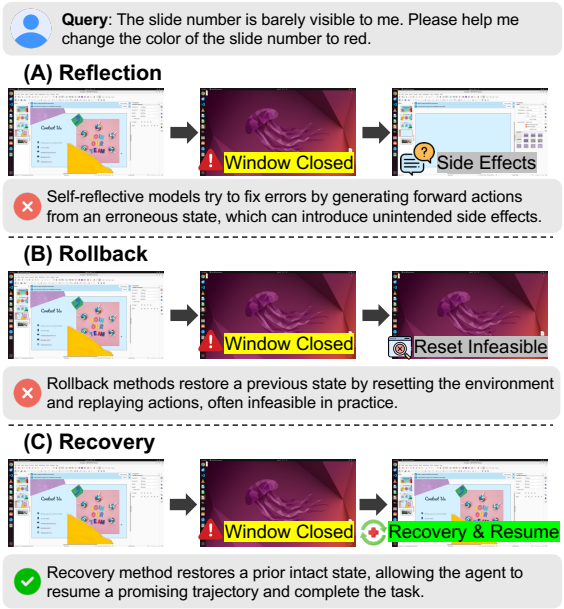


Figure 1: Comparison of error-handling strategies in GUI agents. Unlike (A) reflection and (B) rollback, (C) our recovery restores a prior intact state to complete the task.

However, computer-use agents remain fragile in long-horizon and sophisticated tasks, where early mistakes accumulate and propagate across subsequent steps (Wang et al., 2026a), and where executed actions may leave unintended side effects that should not have occurred (e.g., an agent organizing a directory may complete the task but silently overwriting files that the user wanted to keep). As a result, the ability to revert the environment to a previously intact state is critical for reliable operation. To address this, prior work explores two directions: inducing self-reflective behaviors through textual feedback or parameter updates from reward signals that refine forward ac-

tions (Chae et al., 2025a; Zhang et al., 2026a; Chae et al., 2025b; Wu et al., 2025b; Hu et al., 2025; Wang et al., 2025a; Xue et al., 2026; Wang et al., 2026c), and explicit test-time rollback through full environment resets followed by re-executing all prior actions (i.e., action replay) (Koh et al., 2024b; He et al., 2025) or API-based undo operations in web environments (e.g., browser back) (Hu et al., 2025; Zhang et al., 2026b).

Yet, these heuristic remedies fall short of providing general-purpose recovery. Environment resets are often infeasible in proprietary settings such as production websites and operating systems (Regatti et al., 2021; Nguyen and Cheng, 2023; Wang et al., 2026b), action replay inherits this requirement and additionally becomes costly over long horizons, and API-based rollback is largely confined to web environments, leaving broader desktop and mobile settings uncovered. Consequently, *autonomous recovery*, in which the agent itself restores a prior state without relying on such ad hoc mechanisms, emerges as an essential capability. However, whether current agents possess such ability remains largely unexplored, and evaluating it is non-trivial: latent components of the environment may be implicitly altered during interaction (e.g., a configuration file may have been modified even when the screenshot appears unchanged), making surface-level observations alone insufficient to verify whether a prior state has truly been restored.

In this work, we introduce RETRACE, a pipeline that conducts the first systematic study of recovery behavior in computer-use agents. Motivated by predictive state representation theory (Littman and Sutton, 2001), which represents states by the futures they predict rather than their internal form, we formalize successful recovery as *Transition-Consistent Recovery*: a recovered state matches its target if executing identical downstream actions from both states yields identical outcomes. This formulation captures latent changes that observation-level comparison would miss, enabling robust hidden-state alignment evaluation grounded in behavioral equivalence rather than visual similarity.

To operationalize this, RETRACE then adopts *goal-oriented action replay*. Specifically, we construct on-policy trajectories for each agent and replay each action sequence multiple times, retaining only those that consistently reach the same successful outcome, so that the environment behaves deterministically for these trajectories. From these trajectories, we sample intermediate states as recovery targets, perturb the agent away from them into erroneous states, and ask whether the agent can recover and reproduce the original task outcome by replaying the remaining actions, thereby evaluating recovery directly through downstream task success rather than brittle state-level comparisons.

We then instantiate RETRACE on OSWorld (Xie et al., 2024) and WindowsAgentArena (Bonatti et al., 2024) across closed-source and open-source computer-use agents, and find that recovery plays a more central role in reliable agent operation than currently acknowledged. As a starting point, we investigate that ideal recovery, in which agents are returned to a prior state identified by an oracle (a detector of appropriate recovery points), substantially boosts Success@k over naive environment resets (Section 4), identifying recovery as the principal lever for unlocking the latent task-solving capacity of these agents. Yet current agents leave this lever unused, lacking both the tendency and capability to recover autonomously: for example, the strongest open-source model achieves below 1% recovery success (Section 5), a deficit we trace to post-training that narrows the model toward generating only forward task-completion actions. When equipped with explicit model-based recovery, however, recovery success climbs by more than an order of magnitude, revealing that the underlying capability is latent rather than missing in current models. Moreover, recovery quality in turn translates into downstream task success: accurate recovery delivers measurable end-to-end gains by unlocking otherwise latent task-solving capacity, while weak recovery does not merely fail to help but can actively derail the trajectory (Section 6). These findings position accurate model-based recovery, together with reliable continuation from the recovered state, as a key step toward stronger computer-use agents.

2 Related Works

Computer-Use Agents Computer-use agents aim to complete diverse user queries via fine-grained control of digital environments, spanning online website navigation (Gur et al., 2023; Furuta et al., 2023; OpenAI, 2025; Awadallah et al., 2025), general desktop operating systems such as Linux and Windows (Wang et al., 2025b; Yang et al., 2025; Anthropic, 2026), and mobile platforms such as Android (Ye et al., 2025a; Xu et al., 2026). Pioneering works leverage LLMs over HTML DOM content (Liu et al., 2018; Kim et al., 2023; Zhou

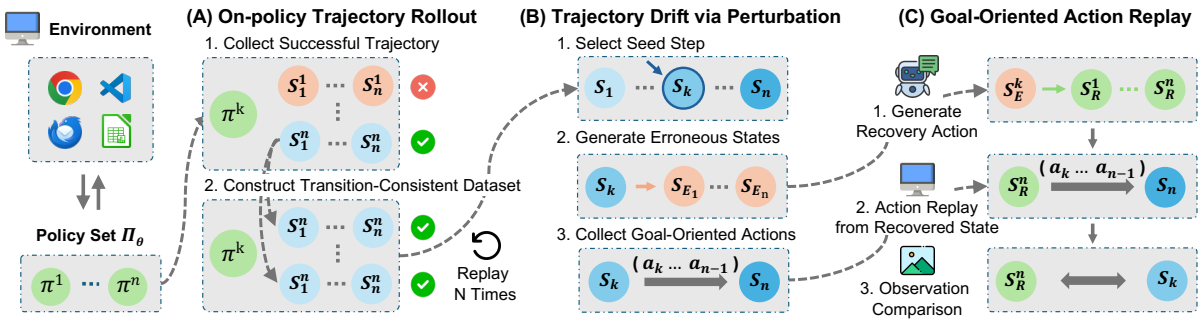


Figure 2: Overview of RETRACE, consisting of (A) On-policy Trajectory Rollout, (B) Trajectory Drift via Perturbation, and (C) Goal-Oriented Action Replay.

et al., 2023), while recent studies operate LVLMs over screenshot observations (Koh et al., 2024a; Wang et al., 2026c; Bai et al., 2025; Lu et al., 2025) and further reinforce backbone capabilities through reinforcement learning (Wang et al., 2025a; Xu et al., 2025; Xue et al., 2026) or external memories (Zheng et al., 2023; Fu et al., 2024). However, these efforts concentrate on strengthening the *forward* decision policy, leaving long-horizon brittleness, where early mistakes accumulate and propagate downstream (Wang et al., 2026a), largely unaddressed. In contrast, we focus on the systematic analysis of *recovery* behavior in computer-use agents, a complementary but underexplored axis.

Agent Behaviors under Failure Prior work explores diverse error-handling strategies across complex interactive tasks. For example, Lightman et al. (2023); Shinn et al. (2024); Xi et al. (2024) use outcome and process-level feedback to refine actions at test time in mathematical and code tasks (Chen et al., 2022). In long-horizon embodied settings, Huang et al. (2022); Wang et al. (2023b,a) leverage external rewards to revise plans or acquire new skills, and Li et al. (2025) performs rollback through resetting the environment and executing identical actions. This problem has also been explored in computer-use agents: Wang et al. (2024); Chae et al. (2025b); Wu et al. (2025b) use external verifiers to induce reflective behaviors (Hu et al., 2025) for effective user task completion, Koh et al. (2024b); He et al. (2025) employ action replay to revert to prior states upon failure, and Hu et al. (2025); Zhang et al. (2026b) utilize explicit rollback via go-back operations in Playwright, while Lu et al. (2026) introduces semantic actions for state recovery within the DFS algorithm. However, these heuristic remedies remain limited: environment resets require wiping and reinitializing the actual environment, which is infeasible in proprietary settings such as production websites and live operating systems (Regatti et al., 2021; Nguyen and Cheng, 2023; Wang et al., 2026b); action replay inherits this requirement and additionally incurs cost that grows with trajectory length, becoming prohibitive over long horizons; and API-based rollback such as browser back exists only in web environments and has no equivalent for desktop or mobile native applications, leaving *autonomous*, model-based recovery largely uncharacterized. In contrast, we present RETRACE, a scalable pipeline that formalizes *transition-consistent recovery* grounded in predictive state representation theory (Littman and Sutton, 2001) and operationalizes it via goal-oriented action replay, enabling the first systematic study of recovery behavior in computer-use agents.

3 Preliminary

Computer-Use Task We formulate a computer-use agent (CUA) task as a Partially Observable Markov Decision Process (Kaelbling et al., 1998), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{E}, \mathcal{T}, \Omega)$, where a policy π_θ interacts with a digital environment $\mathcal{E}$ (e.g., Linux terminals, PowerPoint) to complete a user query q. Each state $s_t = (h_t, o_t) \in \mathcal{S}$ decomposes into a hidden component h_t and an observation $o_t = \Omega(s_t) \in \mathcal{O}$, satisfying the Markov property. At each step, π_θ selects an action $a_t \in \mathcal{A}$, spanning high-level (`CLICK`) to low-level (`pyautogui.click`) operations, yielding $s_{t+1} \sim \mathcal{T}(\cdot \mid s_t, a_t)$.

Forward Policy and Trajectory Since $\mathcal{E}$ is only partially observable, π_θ conditions on the observation–action history, $\pi_\theta(a_t \mid o_1, h_1, a_1, \ldots, o_t, h_t)$, yielding a trajectory $\tau = \{(s_t, a_t)\}_{t=0}^{T}$. However, this forward formulation leaves unspecified how the agent should behave once early actions drive τ into an erroneous state from which q can no longer be completed, the setting we study next.

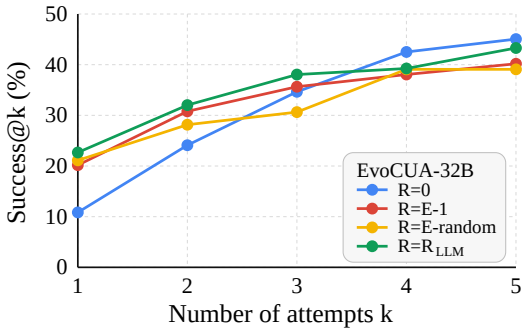


Figure 3: Analysis on the Success@5 gains across different recovery points under ideal recovery scenarios.

4 Accurate Recovery Yields Huge Gains

Here, we show through a preliminary experiment that accurate recovery yields substantial gains, establishing it as a powerful lever for reliable agents. For more details, please see Appendices B.1 and F.

Expected Gains from Ideal Recovery Agents exhibit diverse trajectories across repeated rollouts of the same task, and this variance may imply substantial latent task-solving capability that any single rollout fails to exploit, motivating our hypothesis: reverting to an appropriate past state (*recovery step*) can convert it into task success. To validate this, we measure Success@5, the success rate over 5 independent continuations, when the forward policy resumes from different recovery steps.

Setup We focus on tasks that the forward policy fails to solve under standard rollouts, the cases where recovery can yield measurable gains, with EvoCUA-32B (Xue et al., 2026) as the forward policy on the OSWorld (Xie et al., 2024) benchmark. For each failed trajectory, we annotate the *error step* E (where the agent first goes off track) using an LLM-based detector, and set the recovery step R, from which the forward policy resumes, to one of: the initial state ($\mathrm{R} = 0$, equivalent to a full environment reset), one step before the error ($\mathrm{R} = \mathrm{E}\text{-}1$), a uniformly sampled past step ($\mathrm{R} = \mathrm{E}\text{-random}$), or a step identified by an LLM-based recovery step detector ($\mathrm{R} = \mathrm{R}_{\mathrm{LLM}}$). We use Qwen3.6-Plus (Qwen Team, 2026b) as the backbone for both detectors.

Results As shown in Figure 3, every non-zero recovery strategy delivers striking gains, with Success@5 exceeding 39%. The advantage over full environment resets ($\mathrm{R} = 0$) is most pronounced under small sample budgets (low K). These substantial gains establish accurate recovery as a far more effective route to task success, motivating model-based recovery to restore target prior states.

5 How Well Can Agents Recover?

Here, we analyze the recovery behavior of current computer-use agents using the empirical pipeline illustrated in Figure 2. For more details, please see Appendices A, B.3, C, and F.

5.1 Transition-Consistent Recovery

Given a trajectory $\tau = \{(s_t, a_t)\}_{t=0}^{T}$ produced by a policy π, the recovery task is defined as a function f that maps an arbitrary state s_t back to a past target state, $f(s_t) = s_{\text{target}} = s_{t-\mathrm{d}}$, where d is the recovery distance. Whether this mapping succeeds is judged by a predefined recovery oracle $\mathcal{R}$, which returns a binary reward indicating recovery success or failure, $\mathcal{R}(f(s_t), s_{\text{target}}) \in \{0, 1\}$. Recalling from Section 3 that each state decomposes as $s_t = (h_t, o_t)$ into a hidden component h_t and an observation o_t, $\mathcal{R}$ should verify that $f(s_t)$ matches s_{target} in both components.

A natural instantiation of $\mathcal{R}$ is to compare the observations of $f(s_t)$ and s_{target} directly, e.g., via pixel- or feature-level matching. However, this surface criterion is unreliable, since latent components of the environment may be implicitly altered without any visible change (Shani and Brafman, 2004) (e.g., a hidden configuration file modified while the screenshot remains identical), causing observation-only oracles to falsely declare success on states that nonetheless diverge in their downstream effects. To capture such latent discrepancies, we draw on the theory of predictive state representations (Littman and Sutton, 2001), which defines state equivalence through identical transition dynamics, and formalize *transition-consistent recovery*. Specifically, in a controlled environment $\mathcal{E}$ with a deterministic transition function $\mathcal{T}_{\mathcal{E}}$, we declare a recovery successful when executing the same downstream action $a_{t-\mathrm{d}}$ from the recovered state reproduces the next state along the original trajectory, defined as $\mathcal{R}_{\mathrm{TC}} = \mathbb{I}\big[\mathcal{T}_{\mathcal{E}}(f(s_t), a_{t-\mathrm{d}}) = s_{t-\mathrm{d}+1}\big],\ \forall (s_{t-\mathrm{d}}, a_{t-\mathrm{d}}, s_{t-\mathrm{d}+1}) \in \tau$. which can be extended to the multi-step setting by recursively chaining transitions along τ, as $\mathcal{R}_{\mathrm{TC}} = \mathbb{I}[\mathcal{T}_{\mathcal{E}}(\cdots \mathcal{T}_{\mathcal{E}}(f(s_{\mathrm{t}}), a_{\mathrm{t-d}}) \cdots, a_{\mathrm{t-1}}) = s_{\mathrm{t}}]$

To complement this hidden-state check, we additionally enforce observation-level agreement to guard against cases where the visible interface itself diverges from the target. Formally, we define $\mathcal{R}_{\mathrm{O}} = \mathbb{I}[\,o = \hat{o}\,]$, where $(h, o) = s_{\text{target}}$ and $(\hat{h}, \hat{o}) = f(s_t)$. The final recovery oracle then combines both, $\mathcal{R} = \mathcal{R}_{\mathrm{TC}} \cap \mathcal{R}_{\mathrm{O}}$, jointly evaluating

recovery on the hidden and observable axes.

5.2 Recovery Evaluation

On-Policy Trajectory Rollout Transition-Consistent recovery evaluation requires the error state s_{error} and the goal state s_{target} of trajectory τ, and a deterministic transition function $\mathcal{T}$. Additionally, to examine the model's internal behavior within its own trajectory, trajectory τ must be on-policy with the evaluated policy model π_θ. It begins by constructing ground-truth trajectories $\tau_{\text{gt}} = \{(s_t, a_t)\}_{\text{t=0}}^{\text{T}}$ for each user query $\text{q} \in \mathcal{Q}$ using backbone forward policies $\pi_\theta \in \Pi$, with an oracle evaluator Φ that determines the success of the query q. For each trajectory τ_{gt}, we select seed time steps $t_{\text{seed}} \in \{1, \ldots, T\}$, pair each with the complete trajectory where its oracle value $\Phi_{\text{q}}(\tau_{\text{gt}})$ is 1, and construct an evaluation set $\mathcal{D}$ for each policy π_θ.

Meanwhile, based on the formal definition of state recovery in Section 5.1, we require a reliable transition function. To this end, inspired by previous work that leverages iterative sampling of model output to empirically approximate the latent model distribution (Wu et al., 2025a), we construct an amortized deterministic transition function $\hat{\mathcal{T}}$ for a given action trajectory τ_{gt}. Specifically, we replay the identical action sequence N times and retain only those trajectories that yield N successful outcomes, thereby ensuring reliable state transitions. Finally, we construct a transition-consistent trajectory dataset $\mathcal{D}_{\text{TC}}$ for respective policy models $\pi_\theta \in \Pi$.

Trajectory Drift via Perturbation Once the transition-consistent dataset is formulated for $\forall(\tau_{\text{gt}}, t_{\text{seed}}, \hat{\mathcal{T}}) \in \mathcal{D}_{\text{TC}}$, ground truth error state s_{error} is required. We leverage $s_{t_{\text{seed}}} \in \tau_{\text{gt}}$ as s_{target}, and create s_{error} by drifting the trajectory into erroneous conditions with respect to the original user query q through a perturbation policy π_θ^{p}. We first execute the action sequences of τ_{gt} in $\mathcal{E}$ using $\hat{\mathcal{T}}$ up to the time step t_{seed}, judged by the LLM verifier h_θ to ensure that $s_{t_{\text{seed}}}$ is reproduced. From this state, the perturbation policy π_θ^{p} generates a trajectory $\tau_{\text{p}} = \{(s_{t_{\text{seed}}},\ a_{t_{\text{seed}}}),\ \ldots,\ (s_{t_{\text{seed}}+t_{\text{p}}})\}$ up to the perturbation limit t_{p}. The oracle function Φ_{q}, which measures the success of the original query q, assigns the final state $s_{t_{\text{seed}}+t_{\text{p}}}$ to s_{error} if it is erroneous, allowing robust evaluation of agent recovery behaviors under erroneous conditions. Consequently, the final set for transition-consistent recovery evaluation $\mathcal{D}_{\text{R}}$ is constructed for each policy $\pi_\theta \in \Pi$.

Goal-Oriented Action Replay After constructing the transition-consistent recovery evaluation set $\mathcal{D}_{\text{R}}$, we initialize the recovery process from $s_{\text{error}} \in \mathcal{D}_{\text{R}}$, which serves as the starting state for the recovery policy π_θ^{R}. Since s_{error} is conditioned on s_{target}, we first replay the action sequence in τ_{gt} up to s_{target} and verify with the LLM-based verifier h_θ that the target state is correctly reconstructed. We then replay the perturbed trajectory τ_{p} for t_{p} steps and again use h_θ to ensure faithful reproduction of s_{error}. Starting from s_{error}, the recovery policy π_θ^{R} executes for at most r steps, producing the recovery trajectory $\tau_{\text{R}} = (s_t, a_t)_{t=0}^{T}$ and the recovered state s_T. However, single-step transition-consistent recovery is impractical, as defining oracle annotations for numerous intermediate states is infeasible. Therefore, following the formulation of multi-chain transition-consistent recovery from Section 5.1, we instead leverage the pre-configured oracle Φ for user-query completion and extend the objective to evaluate whether replaying the remaining actions in τ_{gt} from the recovered state successfully reproduces the original task outcome. The recovery oracle is defined as $\mathcal{R}_{\text{TC}} = \mathbb{I}\big[\Phi\text{q}(\mathcal{T}_{\mathcal{E}}(\cdots \mathcal{T}_{\mathcal{E}}(f(s_t), a_{\text{t-d}}), \ldots, a_T)) = 1\big]$, together with $\mathcal{R}_{\text{O}}$ for observation-level matching.

5.3 Recovery Policy

Implicit Recovery The implicit recovery policy π_θ^{R} can be defined as a policy identical to the forward decision policy π_θ, which generates action to recover to previous states, without actual manipulation of the policies through explicit instructions or parameter updates.

Explicit Recovery The explicit recovery policy π_θ^{R} is an external policy specifically designed to revert the agent to a previously targeted goal state s_{target}. Formally, this recovery process is defined as $\pi_\theta^{\text{R}}(a_t \mid s_t, s_{\text{target}})$, which, following the POMDP formulation in Section 3, can be expressed as $\pi_\theta^{\text{R}}(a_t \mid o_{\text{target}}, (o_1, h_1, a_1, \ldots, o_t, h_t))$. Unlike the forward decision policy that initiates from a fixed starting point, the recovery process may begin from an arbitrary state within the initial trajectory. Additionally, the target recovery state also exists within the context of the trajectory created by the forward policy. To account for this, an additional recovery context $\mathcal{C}$ is introduced to fully represent the hidden state $h_t \in s_t$,

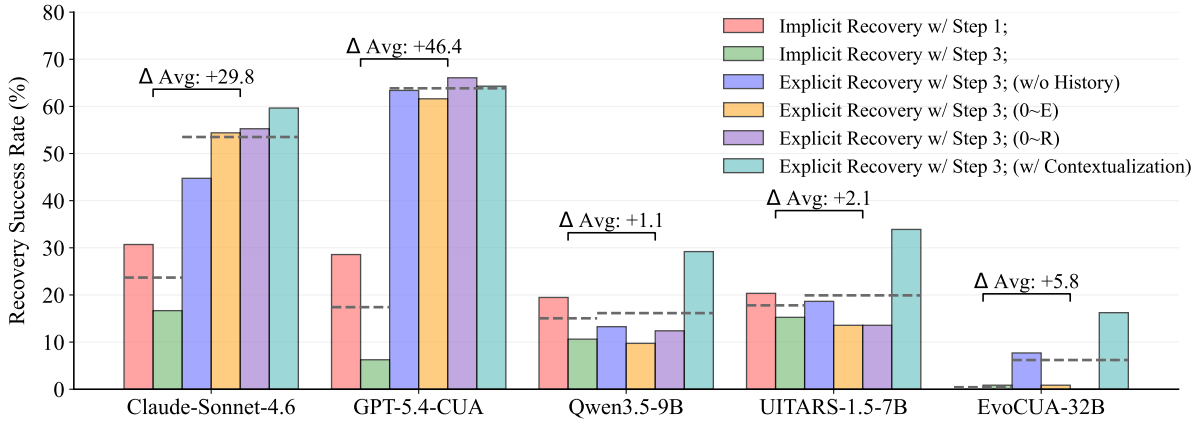


Figure 4: Comparison of short-horizon recovery strategies across multiple computer-use agents in OSWorld.

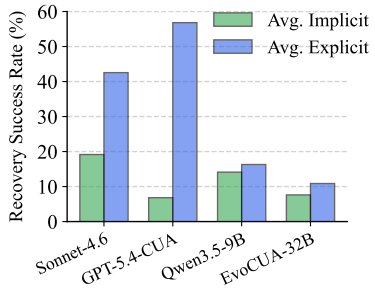


Figure 5: Comparison of short-horizon recovery strategies in WindowsAgentArena benchmark.

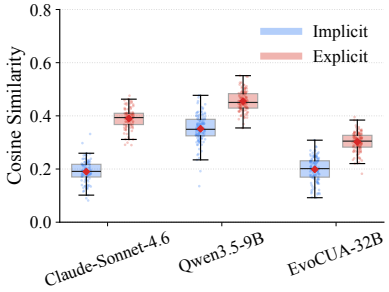


Figure 6: Comparison of the recovery intention between implicit and explicit recovery.

Baselines	Win Rate (%)
w/o $\mathcal{R}_{\text{O}}$	61.0%
w/o $\mathcal{R}_{\text{TC}}$	55.0%
GPT-5.4 Evaluator	51.0%

Table 1: Average Win rate of RETRACE in judgment alignment relative to other evaluation strategies, using human ground truth as the reference. For more details, see Appendix B.3.

leading to the final explicit recovery formulation: $\pi_\theta^{\text{R}}(a_t \mid o_{\text{target}}, \mathcal{C}, (o_1, h_1, a_1, \ldots, o_t, h_t))$.

5.4 Experiments

5.4.1 Setup

Dataset We use two benchmarks, OSWorld (Xie et al., 2024) and WindowsAgentArena (Bonatti et al., 2024), to evaluate the recovery performance of computer-use agents. Both benchmarks provide an initial query and an oracle evaluator that determines whether the query execution succeeds or fails, deployed in isolated settings.

Models We evaluate state-of-the-art computer-use agents, including closed-source models such as Claude Sonnet 4.6 (Anthropic, 2026) and GPT-5.4-Computer-Use (OpenAI, 2026), which achieve top performance on OSWorld. We also consider state-of-the-art post-trained open models, including EvoCUA-32B (Xue et al., 2026) and UI-TARS-1.5-7B (Qin et al., 2025), as well as Qwen3.5-9B-Thinking (Qwen Team, 2026a), the strongest open-source model without task-specific post-training.

Inference Configurations We perform experiments under **short horizon** and **long horizon** settings, where the recovery distance d is 1 and 5 respectively. Additionally, we consider diverse values for the maximum recovery step budget r, primarily utilizing 1, 3, and 10 as main budgets.

Recovery Context We consider various recovery policy configurations with diverse contexts c. For the implicit recovery policy, we provide a history of observation–action pairs from the first step up to the error step, which we denote as **0-E**. For the explicit recovery policy, we evaluate four different variants: **0-E**; **0-R**, which includes the history of observation–action pairs up to the error step s_{error} and target step s_{target}; **w/o History** a setting in which the history is excluded from the context; and **w/ Recovery Target Contextualization**, which contextualizes the transition history leading up to the target recovery step using LLM summarizer to reduce the effect of context overflow (Zhou et al., 2025; Ye et al., 2025b).

5.4.2 Results and Analysis

Agents Fail to Recover Prior States Our results in Figures 4 and 5 demonstrate that even the strongest computer-use agents, Claude-Sonnet-4.6 and GPT-5.4-CUA, have limited recovery capability, while the strongest open-source model, EvoCUA-32B, exhibits complete inability to re-

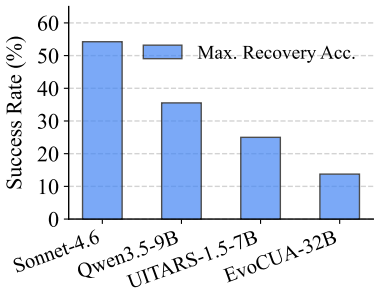


Figure 7: Comparison of maximum long-horizon recovery success rate in OSWorld across different model variants.

Recovery Context	SR (%) ↑	Δ ↑
Baseline	38.76 ± 0.64	—
Contextualizer	41.53 ± 4.19	+2.77
0∼R	$\mathbf{43.21 \pm 0.75}$	**+4.45**
0∼E	40.92 ± 0.82	+2.16
w/o History	39.31 ± 0.64	+0.55

Table 2: End-to-end performance of Qwen3.5-9B on OSWorld across various recovery strategies. We report mean ± standard deviation over two independent runs.

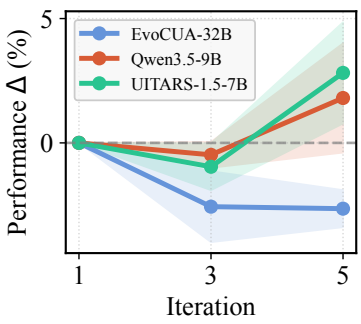


Figure 8: Performance trend Δ across different maximum recovery iterations with *Contextualization* recovery strategy.

Model	Setting	SR (%) ↑
Qwen3.5-9B	Baseline	38.76 ± 0.64
	w/ Recovery	41.57 ± 4.14
UI-TARS-1.5-7B	Baseline	17.25 ± 1.56
	w/ Recovery	15.74 ± 0.65
EvoCUA-32B	Baseline	54.89 ± 4.39
	w/ Recovery	51.55 ± 2.54

Table 3: End-to-end task success rate (SR) on OSWorld across different models with *Recovery Target Contextualization* strategy.

store previous states, achieving an average recovery performance below 1%. These results suggest that current forward policies lack both intrinsic capability and the behavioral tendency to autonomously perform restorative actions under failures. This observation is further supported by the analysis in Figure 6, where the rationales generated during implicit recovery exhibit substantially lower similarity to the ground-truth recovery prompts. This implies that computer-use agents rarely attempt recovery, further supporting our findings.

Analysis on Explicit Recovery The results in Figures 4, 5 suggest that explicit model-based recovery strategies substantially outperform implicit recovery, exceeding it by an average margin of 18.22% in both Linux and Windows settings. Among the various recovery strategies, approaches that explicitly summarize the transition history up to the target recovery step consistently achieve the strongest overall performance across multiple variants. This implies that an explicit contextual summary representing the target recovery state plays a critical role in effective recovery. Nevertheless, even the most capable models combined with basic explicit recovery strategies still fail to achieve complete recovery under our evaluation settings, obtaining a maximum success rate of 54.24% (Figure 7), suggesting that substantial research remains in developing more robust recovery mechanisms for computer-use agents.

Efficiacy of RETRACE To assess the effectiveness of RETRACE in objectively evaluating recovery success, we conduct a human evaluation study against three baselines: variants of our oracle $\mathcal{R}$ with either the image-wise matching component $\mathcal{R}_O$ or the Transition-Consistent Recovery component $\mathcal{R}_{\text{TC}}$ removed, and GPT-5.4, a state-of-the-art model with extensive computer-use knowledge. We recruit 12 participants to evaluate whether recovered states align with the goal state, where annotators evaluate 25 paired samples each. In total, we construct two evaluation subsets, each containing 50 samples per evaluation variant. As shown in Table 4, RETRACE consistently outperforms all baselines, including GPT-5.4, demonstrating the effect of our method. To validate annotation reliability, we additionally measure inter-annotator agreement, obtaining an average agreement of 66% and a Cohen's Kappa coefficient (Cohen, 1960) of 0.37, indicating fair agreement.

6 End-to-End Evaluation of Recovery

In this section, we analyze the impact of recovery performance on the actual performance of computer-use agents. For more details, please see Appendices B.2 and F.

6.1 Setup

We follow the evaluation protocols in Sections 4 and 5.4. Specifically, the forward and recovery agents share the same backbone model, which is used for explicit recovery strategies.

6.2 Results and Analysis

Potential Gains from Model-Based Recovery To validate the potential benefits of model-based recovery, we analyze its performance gains across diverse recovery settings. The results in Table 2

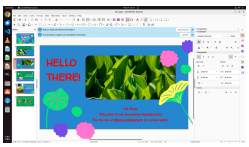
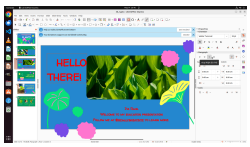
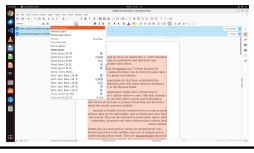
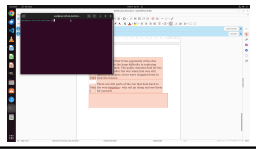
Error Type	Instruction	Target State	Restored State	Action	Problem
Spurious Perception	**Align** the text of the first textbox on slide 3 to the **right**, on slide 4 to the center, and on slide 5 to the left.			**[click]** on the **hovering alignment button** to align slide 3 the text to the right.	**Fixates on to the hovering UI element in recovered state, skipping [drag]** on the text before clicking the alignment button.
State Tracking Failure	Change the font to **"Times New Roman"** throughout the text.			**[scroll down]** to find Times New Roman, since font selection interface is opened.	**Hallucinates the restored state due to failed state synchronization** with its previous states.

Table 4: Qualitative analysis of erroneous agent behaviors induced by imperfect state restoration.

demonstrate that, model-based recovery can consistently unlock the latent capabilities of current computer-use agents, yielding an average performance gain of 2.48% in an end-to-end experimental setting. These findings suggest that model-based recovery holds substantial promise in improving the overall performance and robustness of computer-use agents, despite suboptimal recovery success rates observed in Figures 4 and 5. Accordingly, the results indicate that effective recovery mechanisms could better translate the models' latent capabilities into successful task completion.

Recovery Performance Effects on Task Completion Our results in Table 3 suggest that recovery variants exhibiting weaker recovery capabilities suffer from substantial performance degradation during end-to-end inference, highlighting the importance of accurate recovery behaviors. In addition, the results in Figure 8 demonstrate that repeated recovery attempts can further improve the performance of variants with strong recovery capabilities. In contrast, EvoCUA-32B exhibits severely limited explicit recovery capability, suggesting that even a single successful recovery attempt can substantially improve downstream task performance, whereas consistently low recovery success rates are more likely to derail trajectories into increasingly erroneous states.

6.3 Qualitative Analysis

In Section 6.2, we showed that poor recovery performance can severely degrade downstream task completion, unlike the idealized settings in Section 4. To further analyze this behavior, we examine the rationales of backbone forward agents after recovering to incorrect states and categorize their behaviors into two major patterns in Figure 4. Additional examples are provided in Appendix D.

Spurious Perception Our first observation indicates that agents tend to *fixate on elements distinct from the target state*, exhibiting a tendency to rely on these elements for their next actions. For example, in Table 6, the agent immediately clicks the right alignment button because the restored state hovers over it, bypassing the required title-drag operation that should have occurred as the next action from the original target recovery state, deviating from the optimal trajectory for correct task completion.

State Tracking Failure Meanwhile, agents also tend to hallucinate actions based on the current restored state, failing to properly synchronize it with the original target state when a discrepancy exists. For example, as shown in Table 6, although the restored state is masked by the terminal interface and does not display the font selection UI, the agent operates under the afterimage that the current state matches the target state.

7 Conclusion

In this paper, we conducted the first systematic study of recovery behavior in computer-use agents through RETRACE, a scalable pipeline that formalizes successful restoration as *Transition-Consistent Recovery*, where a recovered state matches its target if identical downstream actions yield identical outcomes, and operationalizes this via *goal-oriented action replay*. Empirical evaluations across closed and open-source agents reveal that current agents lack both the tendency and capability to recover autonomously, while explicit model-based recovery substantially closes this gap and translates into measurable end-to-end gains, despite characteristic failure modes such as spurious perception and state-tracking errors. We believe these findings position accurate model-based recovery as a key step toward reliable computer-use agents.

Limitations

While our work establishes recovery as a central lever for reliable computer-use agents, our findings also point to an exciting opportunity for follow-up work. In particular, we observe that recovery capability is latent rather than missing in current models, as explicit model-based recovery already yields substantial gains over implicit recovery, which suggests that dedicated post-training or reinforcement learning that directly internalizes recovery behavior into the policy itself would be an exciting step toward agents that recover autonomously.

Ethics Statement

Our work aims to advance the reliability of computer-use agents by systematically studying their recovery behavior, contributing positively to the safe and dependable deployment of digital automation. However, we acknowledge the dual-use nature of such research, as more capable agents could in principle be repurposed for harmful automation. To mitigate such risks, we conduct all experiments in sandboxed virtual environments using publicly available benchmarks in accordance with their licenses, without interaction with real user accounts or personal data, and recommend pairing deployments with appropriate safeguards.

References

Anthropic. 2026. Claude-sonnet-4.6. https://www.anthropic.com/news/claude-sonnet-4-6.

Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Vibhav Vineet, and 1 others. 2025. Fara-7b: An efficient agentic model for computer use. *arXiv preprint arXiv:2511.19663*.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, and 1 others. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, and 1 others. 2024. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*.

Hyungjoo Chae, Namyoung Kim, Kai Tzu iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. 2025a. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *The Thirteenth International Conference on Learning Representations*.

Hyungjoo Chae, Sunghwan Kim, Junhee Cho, Seungone Kim, Seungjun Moon, Gyeom Hwangbo, Dongha Lim, Minjin Kim, Yeonjun Hwang, Minju Gwak, and 1 others. 2025b. Web-shepherd: Advancing prms for reinforcing web agents. *arXiv preprint arXiv:2505.15277*.

Bei Chen, Fengji Zhang, A. Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. 2022. Codet: Code generation with generated tests. *ArXiv*, abs/2207.10397.

Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 179 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *ArXiv*, abs/2501.12948.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37:119919–119948.

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

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*.

Shiqi He, Yue Cui, Xinyu Ma, Yaliang Li, Bolin Ding, and Mosharaf Chowdhury. 2025. Branch-and-browse: Efficient and controllable web exploration with tree-structured reasoning and action memory. *arXiv preprint arXiv:2510.19838*.

Minda Hu, Tianqing Fang, Jianshu Zhang, Jun-Yu Ma, Zhisong Zhang, Jingyan Zhou, Hongming Zhang, Haitao Mi, Dong Yu, and Irwin King. 2025. WebCoT: Enhancing web agent reasoning by reconstructing chain-of-thought in reflection, branching, and rollback. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5155–5173, Suzhou, China. Association for Computational Linguistics.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, and 1 others. 2022. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36:39648–39677.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024a. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024b. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Xingzuo Li, Kehai Chen, Yunfei Long, Xuefeng Bai, Yong Xu, and Min Zhang. 2025. Generator-assistant stepwise rollback framework for large language model agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17694–17711.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The twelfth international conference on learning representations*.

Michael Littman and Richard S Sutton. 2001. Predictive representations of state. *Advances in neural information processing systems*, 14.

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

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, and 1 others. 2025. Deepseek-vl: Towards real-world vision-language understanding, 2024. *URL https://arxiv. org/abs/2403.05525*.

Ziyu Lu, Tengjin Weng, Yiying Yang, Yuhang Zhao, Xinxin Huang, and Wenhao Jiang. 2026. Beap-agent: Backtrackable execution and adaptive planning for gui agents. *arXiv preprint arXiv:2601.21352*.

Hoai-An Nguyen and Ching-An Cheng. 2023. Provable reset-free reinforcement learning by no-regret reduction. In *International Conference on Machine Learning*, pages 25939–25955. PMLR.

OpenAI. 2025. Operator. https://openai.com/ko-KR/index/introducing-operator/.

OpenAI. 2026. Gpt-5.4-computer use. https://openai.com/ko-KR/index/introducing-gpt-5-4/.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Qwen Team. 2026a. Qwen3.5: Towards native multimodal agents.

Qwen Team. 2026b. Qwen3.6-Plus: Towards real world agents.

Jayanth Reddy Regatti, Aniket Anand Deshmukh, Frank Cheng, Young Hun Jung, Abhishek Gupta, and Urun Dogan. 2021. Offline rl with resource constrained online deployment. *arXiv preprint arXiv:2110.03165*.

Pascal J Sager, Benjamin Meyer, Peng Yan, Rebekka von Wartburg-Kottler, Layan Etaiwi, Aref Enayati, Gabriel Nobel, Ahmed Abdulkadir, Benjamin F Grewe, and Thilo Stadelmann. 2025. Ai agents for computer use: A review of instruction-based computer control, gui automation, and operator assistants. *arXiv preprint arXiv:2501.16150*, pages 1–64.

Guy Shani and Ronen Brafman. 2004. Resolving perceptual aliasing in the presence of noisy sensors. *Advances in Neural Information Processing Systems*, 17.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning, 2023. *URL https://arxiv. org/abs/2303.11366*, 8.

Aaditya K. Singh, Adam Fry, Adam Perelman, Adam Tart, Adithya Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr A Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 464 others. 2025. Openai gpt-5 system card.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models, 2023. *URL https://arxiv. org/abs/2305.16291*, 2(11).

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025a. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *Advances in Neural Information Processing Systems*, 37:2686–2710.

Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D Nowak. 2026a. The long-horizon task mirage? diagnosing where and why agentic systems break. *arXiv preprint arXiv:2604.11978*.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, and 1 others. 2025b. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. 2026b. Agent world model: Infinity synthetic environments for agentic reinforcement learning. *arXiv preprint arXiv:2602.10090*.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612.

Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023b. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

Ziwei Wang, Leyang Yang, Xiaoxuan Tang, Sheng Zhou, Dajun Chen, Wei Jiang, and Yong Li. 2026c. History-aware reasoning for gui agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. 2025a. The invisible leash: Why rlvr may or may not escape its origin. *arXiv preprint arXiv:2507.14843*.

Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. 2025b. Backtrackagent: Enhancing gui agent with error detection and backtracking mechanism. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4250–4272.

Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang Hong, Shihan Do, Wenyu Zhan, Xiao Wang, Rui Zheng, Tao Ji, Xiaowei Shi, Yitao Zhai, Rongxiang Weng, Jingang Wang, Xunliang Cai, Tao Gui, and 5 others. 2024. Enhancing llm reasoning via critique models with test-time and training-time supervision. *Preprint*, arXiv:2411.16579.

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. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Preprint*, arXiv:2404.07972.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, and 1 others. 2026. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*.

Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. 2025. Mobilerl: Online agentic reinforcement learning for mobile gui agents. *arXiv preprint arXiv:2509.18119*.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, and 1 others. 2026. Evocua: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*.

Bowen Yang, Kaiming Jin, Zhenyu Wu, Zhaoyang Liu, Qiushi Sun, Zehao Li, JingJing Xie, Zhoumianze Liu, Fangzhi Xu, Kanzhi Cheng, Qingyun Li, Yian Wang, Yu Qiao, Zun Wang, and Zichen Ding. 2026. Os-symphony: A holistic framework for robust and generalist computer-using agent. *Preprint*, arXiv:2601.07779.

Yan Yang, Dongxu Li, Yutong Dai, Yuhao Yang, Ziyang Luo, Zirui Zhao, Zhiyuan Hu, Junzhe Huang, Amrita Saha, Zeyuan Chen, and 1 others. 2025. Gta1: Gui test-time scaling agent. *arXiv preprint arXiv:2507.05791*.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, and 1 others. 2025a. Mobile-agent-v3: Fundamental agents for gui automation, 2025. *URL https://arxiv. org/abs/2508.15144*, 4:21–27.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, and 1 others. 2025b. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*.

Yao Zhang, Shijie Tang, Zeyu Li, Zhen Han, and Volker Tresp. 2026a. Webarbiter: A principle-guided reasoning process reward model for web agents. *arXiv preprint arXiv:2601.21872*.

Zhisong Zhang, Tianqing Fang, Kaixin Ma, Wenhao Yu, Hongming Zhang, Haitao Mi, and Dong Yu. 2026b. Webrollback: Enhancing web agents with explicit rollback mechanisms. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 187–197.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2023. Synapse: Trajectory-as-exemplar prompting with memory for computer control. *arXiv preprint arXiv:2306.07863*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*.

A RETRACE Details

Here, we provide implementation details on the evaluation set construction and recovery evaluation in RETRACE framework.

A.1 Evaluation Set Construction

Benchmarks We primarily leverage two publicly available benchmarks: OSWorld[1] (Xie et al., 2024), released under the Apache License 2.0, and WindowsAgentArena (Bonatti et al., 2024), released under the MIT License. To accelerate evaluation, we execute multiple environments in parallel, and for WindowsAgentArena, we utilize the processed environments provided by OS-Symphony[2] (Yang et al., 2026), released under the Apache License 2.0, to enable efficient large-scale evaluation set construction. We deploy all environments within isolated Docker containers and apply a 10-second post-action settle window to ensure that pending UI updates are fully rendered before capturing the next observation. For screenshot resolution, we use $(1920, 1080)$ for OSWorld and $(1280, 800)$ for WindowsAgentArena. Finally, due to budget constraints associated with proxy server usage, we restrict our evaluation to tasks that do not require proxy settings in OSWorld.

Models To construct the recovery evaluation set, we use five models: Claude-Sonnet-4.6 (Anthropic, 2026), GPT-5.4-CUA (OpenAI, 2026), Qwen-3.5-9B-Thinking (Qwen Team, 2026a), EvoCUA-32B (Xue et al., 2026), and UI-TARS-1.5-7B (Qin et al., 2025). Claude-Sonnet-4.6 and GPT-5.4-CUA are configured with thinking mode and xhigh reasoning mode, respectively. We use the *pyautogui* action space for all models except Claude-Sonnet-4.6, which uses the proprietary *claude-computer use* action space, while providing raw screenshots as observations. We adopt the default prompts from OSWorld and adapt them to Windows environments for WindowsAgentArena evaluation (Figures 11, 12, 13, 14, 15, 16, 17 18, 19, 20, 21, 22, 23, 24, 25). All open-source models are served locally using vLLM (Kwon et al., 2023) with bfloat16 precision through OpenAI-compatible HTTP endpoints. Experiments are conducted using a combination of one NVIDIA H200 GPU (141GB) and two RTX A6000 GPUs.

[1] https://github.com/xlang-ai/OSWorld.git
[2] https://github.com/OS-Copilot/OS-Symphony.git

On-Policy Rollout For each model, we construct trajectories for the corresponding tasks in each benchmark. We then create an annotated dataset by selecting potential starting steps within each trajectory, while filtering out trajectories containing duplicate actions within a single step or erroneous trajectories in which actions are missing from the agent output, as such cases can complicate the annotation process. Subsequently, the action sequences from each trajectory are replayed consecutively, and we retain only the trajectories that achieve a successful oracle evaluation in five consecutive replays. This process empirically ensures deterministic transition dynamics, resulting in a filtered evaluation set in which the starting step of each trajectory is guaranteed to exhibit consistent transitions.

Trajectory Perturbation Using the filtered evaluation set consisting of annotated starting steps from each trajectory, we replay the action sequences until reaching the designated starting step, while verifying that the resulting observation matches the original state. Subsequently, a perturbation agent sharing the same backbone as the forward agent is prompted (Figures 26, 27) to intentionally drift the trajectory under a maximum step budget identical to the recovery distance d. For short-horizon recovery evaluation, we use a perturbation budget of 1 for all models, except GPT-5.4-CUA, for which we use a budget of 2 due to its strong tendency to generate a "take a screenshot" action as the initial step which maps to `time.sleep()`, even under explicit instructions. We then retain only trajectories that result in oracle failure according to the benchmark evaluators. This process constructs a recovery-annotated dataset, where the original starting state serves as the target recovery state and the final perturbed state serves as the error state.

Final Evaluation Set Construction Finally, using the recovery-annotated dataset, we exclude problematic samples whose perturbation trajectories generate multiple actions within a single step or fail to produce actions at arbitrary steps, as such cases complicate annotation. For short-horizon recovery evaluation, we additionally remove samples without meaningful perturbation actions, such as `time.sleep()` or `WAIT`. For GPT-5.4-CUA, we retain only samples where the first action is "take a screenshot" and the second action corresponds to an actual perturbation action, due to its

strong tendency to begin trajectories with screenshot actions. Furthermore, we remove samples where the error state and target recovery state show significantly high similarity, while also assuring that duplicate samples with both same error state observation and target recovery state observation do not exist, adopting SSIM as core mechanism. After filtering, we randomly sample evaluation subsets. For short-horizon evaluation in OSWorld, we sample 120 instances for all models except UI-TARS-1.5-7B, for which we sample 60 due to the limited number of remaining trajectories. For long-horizon evaluation, we sample 80 instances for Qwen-3.5-9B and EvoCUA-32B, 60 for Claude-Sonnet-4.6, and 50 for UI-TARS-1.5-7B. Finally WindowsAgentArena, we sample 70 instances for Claude-Sonnet-4.6 and GPT-5.4-CUA, and 50 for EvoCUA-32B and Qwen-3.5-9B, due to the different number of remaining samples after the filtering process.

A.2 Recovery Evaluation

Benchmarks We use the evaluation sets constructed in Section A.1 to measure the recovery success rates of both implicit and explicit recovery policies, where the annotated evaluation sets are instantiated in both OSWorld and WindowsAgentArena, where we set the environment settings identical with Section A.1.

Models We evaluate both implicit and explicit recovery using the five models described in Section A.2. For Claude-Sonnet-4.6 and GPT-5.4-CUA, we use a temperature of 1.0 with thinking mode and xhigh reasoning mode enabled, respectively. For Qwen-3.5-9B and EvoCUA-32B, we use a temperature of 0.7, while UI-TARS-1.5-7B is evaluated with a temperature of 0.0. As the recovery target contextualizer, we use Claude-Sonnet-4.6 as the backbone state-transition summarizer with a temperature of 0.0 and a maximum token budget of 1024. All open-source backbones are served locally using vLLM within the same computing infrastructure described in Section A.1. Finally, we use GPT-5-Mini (Singh et al., 2025) as the state-verification model with default OpenAI hyperparameters.

Recovery Context For implicit recovery strategies, we provide the full history of observation–action pairs up to the error state, allowing the forward agent to infer the underlying state transitions, similar to real-world settings where agents interact continuously throughout the trajectory. In contrast, for explicit recovery strategies, we provide policies with various recovery contexts, including the observation–action history up to the error state and the target recovery state, which we denote as **0-E** and **0-G**, respectively. We additionally consider settings where such observations are not provided, as well as settings where the observation–action pairs up to the goal state are summarized by an LLM contextualizer that generates a compact description of the target recovery state and the transitions leading to it. When constructing observation–action histories, we maintain aligned observation–action pairs and exclude the final observation resulting from the last action, since new observation is input as current observation. The only exception is GPT-5.4-CUA, for which the final observation is included in the history due to proprietary API technical constraints.

Prompts We construct the system prompts for explicit recovery by concatenating the recovery prefix prompts (Figures 28, 30) from Section F with the default system prompts used for each model in their respective benchmark settings. Similarly, we construct explicit recovery instructions by combining the model-specific instructions provided in the default OSWorld benchmark with the recovery instruction prefixes from Figures 29 and 31. In contrast, for implicit recovery policies, we use the original default system prompts and instructions without any recovery-specific prefix prompts or instructions. Meanwhile, we use prompts in Figures 33 and 34 for recovery target contextualizer, while leveraging prompts in Figure 32 for LLM -based state verification during recovery evaluation.

Recovery Verification In RETRACE, we evaluate recovery success along two axes: latent state recovery success and observation-wise state recovery success. For latent state recovery, we leverage our goal-oriented action replay framework, where recovery is considered unsuccessful if the benchmark oracle evaluator outputs 0 after executing the remaining actions from the ground-truth trajectory starting from the recovered state. For observation-wise recovery, we use SSIM (Wang et al., 2004) with a threshold of 0.99 to detect visual mismatches that may interfere with accurate evaluation. Furthermore, in our main recovery evaluation experiments, we exclusively sample tasks that are consistently executable across all variants without docker failures or state verification failures, thereby ensuring a fair and reliable comparison of recovery

success rates.

B Additional Experiment Details

Here, we explain the experimental settings and implementation details for experiments introduced in Section 4 and 6, along with details on Human Evaluation introduced in Table 4.

B.1 Ideal Gains from Recovery

This experiment estimates how much end-to-end performance can improve when an agent is allowed to resume from an appropriate past state after a failure. It measures the potential gain from recovery under a controlled replay setting, where the target past state is reconstructed by action replay before the forward agent resumes execution.

Benchmark and model We run the analysis on OSWorld (Xie et al., 2024) using EvoCUA-32B (Xue et al., 2026) as the forward model, following the setup from Section A.1. We set the temperature to 0.7, `max_tokens` to 32768, and `max_steps` to 50. We first collect standard forward rollouts and keep only failed instances whose `result.txt` score is 0. For each failed instance, we identify an error step and select a corresponding recovery step using the recovery-step selection procedure described in Section 4. The environment at the selected recovery step is reconstructed by replaying the saved action prefix up to that step. We measure Success@5 on 83 samples after excluding instances where action replay fails.

Fixed annotations For each failed instance, we annotate an error step E once before running any recovery-step experiment. The annotation stage reads the saved trajectory, reconstructs the observation-action history, and calls the LLM error-step detector. When evaluating the LLM-selected recovery step, we also annotate a recovery step $\mathrm{R}_{\mathrm{LLM}}$ once in the same preparation stage. These annotations are then materialized into a fixed run plan, so different recovery-step conditions are evaluated against the same underlying failed trajectory rather than re-detecting E or $\mathrm{R}_{\mathrm{LLM}}$ at runtime.

Recovery Steps We compare four starting steps. $\mathrm{R} = 0$ restarts from the initial environment state. $\mathrm{R} = \mathrm{E} - 1$ resumes from the step immediately before the detected error. $\mathrm{R} = \mathrm{E} -$ random samples a previous step with a fixed seed and reuses the same sampled step across runs for that instance. $\mathrm{R} = \mathrm{R}_{\mathrm{LLM}}$ uses the recovery step predicted by the recovery-step detector. For $\mathrm{R} > 0$, the environment is reset and the saved action prefix up to R is replayed before the forward agent resumes. For $\mathrm{R} = 0$, no prefix is replayed.

Metric For each instance and each condition, we draw five independent continuations from the selected recovery state, and report Success@K for $\mathrm{K} \in \{1, \ldots, 5\}$. Each continuation begins with action-prefix replay to reconstruct the recovery state; because replay can occasionally desynchronize from the saved trajectory (e.g., due to UI timing), we retry replay up to three times per continuation. Success@K measures whether at least one of the K successfully reconstructed continuations from the same recovery state completes the task, and reflects how much of the agent's latent task-solving ability is exposed by recovery.

B.2 End-to-End Evaluation

Benchmark We evaluate on the OSWorld (Xie et al., 2024) benchmark, where we follow a consistent setting as A.1.

Forward agents We use three open-source models as the forward agent: Qwen3.5-9B (Qwen Team, 2026a) with thinking mode, UI-TARS-1.5-7B (Qin et al., 2025), and EvoCUA-32B (Xue et al., 2026), following the setup in Section A.1, where default system prompts and instructions in Section A.1 is provided. Within each end-to-end experiment, the recovery agent shares the same backbone as the forward agent; the two differ only in their prompts and step budgets. For all forward agents we set `max_steps` to 50. Temperatures are chosen from each model's official inference recipe except for EvoCUA-32B: its default temperature of 0.0 produces deterministic outputs, which would cause the forward agent to repeat the same action when it resumes from a past state, so we set it to 0.7 to preserve action diversity after recovery. The resulting values are 0.7 for EvoCUA-32B, 0.7 for Qwen3.5-9B-Thinking, and 1.0 for UI-TARS-1.5-7B. The `max_tokens` budget is 32768 for EvoCUA-32B (to accommodate its long reasoning trace), 3000 for Qwen3.5-9B-Thinking, and 500 for UI-TARS-1.5-7B.

Detectors Both the error-step detector and the recovery-step detector are LLM-based and share Qwen3.6-Plus (Qwen Team, 2026b) as their backbone, accessed through the OpenRouter API[3] with

[3] https://openrouter.ai/

default decoding parameters, with custom system prompts (Figure 35, 36). The error detector is queried once per forward step. Once a recovery is triggered, the error detector is frozen for the next 5 steps; this prevents the detector from re-interrupting the forward agent before any recovery-induced changes have a chance to take effect.

Recovery agent The recovery agent is given a recovery step budget of 10 steps per attempt and is triggered at most `max_recovery_iteration` times per episode, with `max_recovery_iteration` $\in \{1, 3, 5\}$, using the identical system prompt and instructions as Section A.2. Recovery steps are tracked separately from the forward agent's step budget. When all retries are exhausted, the forward agent continues without further recovery activation.

Post-recovery context After recovery completes, the forward agent resumes from the recovery-step context where the message history is truncated to the recovery boundary, and a short transition string informing the agent that the environment has been recovered is prepended before the next forward decision.

B.3 Human Evaluation

When constructing the human evaluation sets, we sample recovery results from three representative models, including Claude-Sonnet-4.6, Qwen3.5-9B, and EvoCUA-32B, across both short- and long-horizon evaluation settings in OSWorld, as well as the short-horizon setting in WindowsAgentArena. We compare RETRACE against three baselines: variants of our oracle evaluator with either $\mathcal{R}_O$ or $\mathcal{R}_{\text{TC}}$ removed, and GPT-5.4, a state-of-the-art model with extensive computer-use knowledge. We collect cases where the evaluation decisions produced by the oracle implemented in RETRACE differ from those of the alternative evaluators, and randomly sample 50 instances for each variant, resulting in a total of 150 evaluation samples in Table 4.

For the image-wise matching evaluator, we provide only the recovery target screenshot and the screenshot after the final recovery step. For the Transition-Consistent evaluator, success is determined solely based on whether replaying the remaining actions from the recovered state successfully reaches the original task goal. For GPT-5.4, we provide the history of observation-action pairs leading up to the error step, the recovery trajectory generated by the recovery agent, the recovery target screenshot, the error-step screenshot, and the final recovered-state screenshot. GPT-5.4 is then prompted to determine whether the semantic meaning of the recovered state aligns with the target recovery state, following the same context and inputs provided to human annotators for a fair comparison (See Figure 37 for prompt details).

After constructing the evaluation set, we split the 50 samples for each variant into two subsets and assigned two human annotators to each subset. In total, we recruited 12 annotators (4 annotators for each variant) who were highly proficient in computer-use tasks and voluntarily consented to participate in the study, with each annotator compensated approximately 20 USD for their participation, provided with detailed guideline for objective evaluation (Table 10). Each pair of annotators independently evaluated whether the recovery target state had been successfully restored, labeling each sample in a binary success-or-failure manner consistent with our evaluators.

Using these annotations as reference labels, we compared the decisions from our oracle evaluator against those from the alternative evaluators and measured their alignment with human judgments. Specifically, we independently computed the win rate of our oracle evaluator against each baseline for every annotator pair, and report the average win rate across all four subsets per variant in Table 4. Across annotator pairs, we additionally observed an average agreement rate of 66% and a Cohen's kappa coefficient of 0.37 (Cohen, 1960).

Model	Max 2	Max 4	Max 5	Max 10
Claude-Sonnet-4.6	26.36	11.82	8.18	7.27
Qwen3.5-9B	10.62	7.08	11.50	6.19
EvoCUA-32B	0.00	0.00	0.85	2.56
UI-TARS-1.5-7B	11.67	8.33	15.00	10.00

Table 5: Implicit recovery success rate (%) across diverse max recovery step budgets in OsWorld benchmark.

C Supplementary Experiments

Here, we provide additional experiments to further support the results and analysis in the main paper.

Robust Suboptimal Implicit Recovery The results in Table 5 indicate that current computer-use agents lack the capability to perform restorative actions, which we observe consistently across ar-

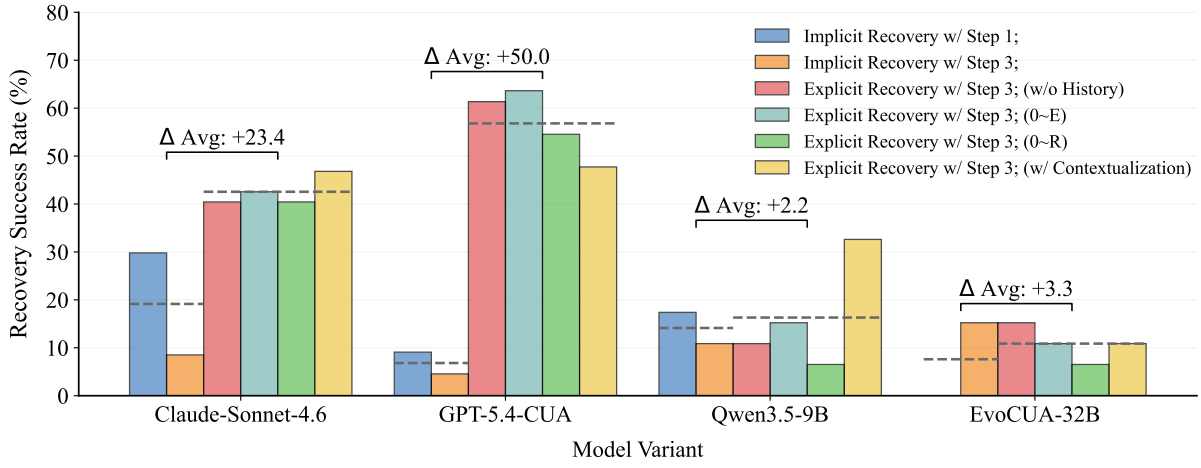


Figure 9: Expanded Results on WindowsAgentArena from Figure 5.

bitrary recovery budgets. Moreover, the recovery success rates of most agents converge toward zero as the maximum recovery step increases, with the exception of EvoCUA, which still exhibits severely limited performance below 3%, despite marginal improvements. These results further support our findings from the main paper that current agents fundamentally lack restorative capabilities under failure scenarios.

D Qualitative Analyses

Here, we provide additional qualitative examples that explain the behavior pattern of computer agents under suboptimal recovery demonstrated in Table 6.

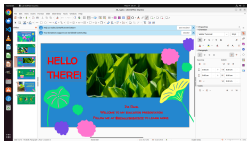
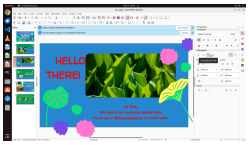
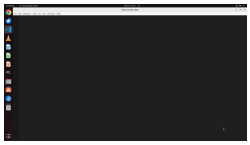
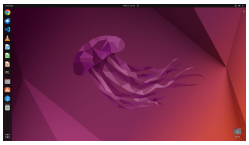
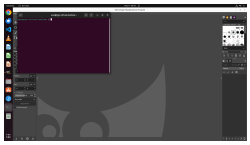
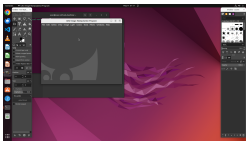
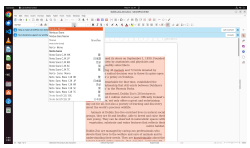
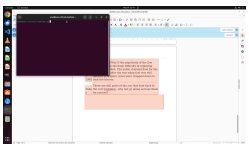
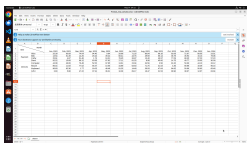
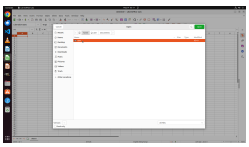
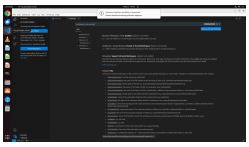
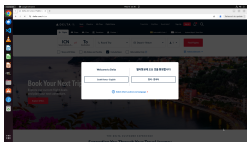
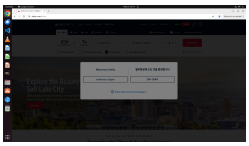
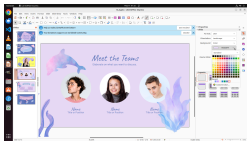
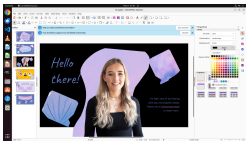
Error Type	Instruction	Target State	Restored State	Action	Problem
Spurious Perception	**Align** the text of the first textbox on slide 3 to the **right**, on slide 4 to the center, and on slide 5 to the left.			**[click]** on the **hovering alignment button** to align slide 3 the text to the right.	**Fixates on to the hovering UI element in recovered state, skipping [drag]** on the text before clicking the alignment button.
Spurious Perception	Please help me **change the color theme of VS Code** to Visual Studio Dark.			**[click] Activities element** in the top left corner to launch vscode application	**Fixates on to the UI element** appearing in the restored state, unable to correctly click vscode application after restoration.
Spurious Perception	Could you help me **remove the dock** on the left side of the screen in the GIMP?			**[click]** on windows option in the right window to explore options for removing the dock	**Fixates on to the new windows option irrelevant to removing the dock option**, fails to generate correct actions due to incorrect state restoration.
State Tracking Failure	Change the font to **"Times New Roman"** throughout the text.			**[scroll down]** to find Times New Roman, since font selection interface is opened.	**Hallucinates the restored state due to failed state synchronization** with its previous states.
State Tracking Failure	Help me **freeze the range A1:B1** on this sheet to keep the headers always visible.			**[Drag]** from column A1 to B1, in order to accurately select the area we want to freeze	**Hallucinates that the restored state allows clicking the range A1 to B1, when in fact, cannot be dragged**, failing to synchronize with the actual prior state.
State Tracking Failure	Please **modify VS Code's settings to disable error** reporting for Python missing imports.			**[hotkey(key='ctrl shift p')]** I checked the results, but there's nothing in the search box. I'm aware of a better solution.	**Hallucinates that the restored state has blank results, just like the recovery target state**, leading to completely novel behavior.
State Tracking Failure	Find flights from Seattle to New York on 5th next month and only show those that can be purchased with miles.			**[click]** I located the button labeled "South Korea - English" and clicked on it to continue booking my flight.	**Hallucinates the current state as clickable based on the target-state observation**, failing to correctly synchronize with the actual system state.
State Tracking Failure	Set the background color to yellow for any slide that contains one or more images of real people, and set the title of slide 2 as "Let's start".			**[click]** I've already changed the background on the third slide to yellow, it's time to change fourth one.	Hallucinates the current state as being on the fourth slide instead of third, unable to synchronize with the erroneous recovered state.

Table 6: Additional Qualitative analysis of erroneous agent behaviors induced by imperfect state restoration.

E Human Evaluation Guidelines

In this section, we provide the guidelines for human evaluation provided to participants in Table 4.

You will be provided with:
- Goal Screenshot
- Error-Step Screenshot
- Final Recovery Screenshot
- Original Instruction
- Recovery Trajectory
- Original Execution Trajectory
- Your task is to determine whether the recovery process successfully restores the environment to the intended goal state.

Evaluation Objective
- Evaluate whether the Final Recovery Screenshot is functionally equivalent to the Goal Screenshot.
- Success (1): the environment is correctly restored
- Failure (0): the environment is not properly restored

Evaluation Principles
1. Prioritize Visual Similarity
The primary signal is the visual similarity between the Final Recovery Screenshot and the Goal Screenshot.

Please carefully compare the screenshots at a fine-grained, segment-level detail, including interaction-dependent UI elements. Pay close attention to whether:

- active windows or pages differ,
- selected content or tabs change,
- focused text boxes differ,
- dropdowns or menus are opened differently,
- cursor positions or hover states change,
- highlighted or selected UI elements differ,
- drag-and-drop states change,
or other interaction-sensitive regions are inconsistent.

Mark as Failure (0) if important visual or interaction-level differences remain.

Minor differences caused by dynamic environments (e.g., notifications, pop-ups, timestamps, loading indicators, rendering variations) can be ignored if the underlying functional state remains equivalent.

2. Verify Functional Recovery
The recovered state should allow the original task to continue correctly.
Even if screenshots appear visually similar, mark as Failure (0) if:-

- important interaction context is missing,
- unintended changes remain,
- incorrect files/pages/windows are active,
or the operational state is not properly restored.
Use the recovery trajectory and original trajectory to understand:
- what actions changed the environment,
- whether erroneous states were actually reverted,
- and whether the recovery process is consistent with the intended goal state.

Trajectory information is especially important when screenshots appear visually similar but the actual operational state may still differ.

Final Output
Binary Label: 1 for Success
Binary Label: 0 for Failure

Figure 10: The human evaluation guideline used for experiments in Table 4

F Prompts

Here, we provide prompts used for experiments.

```
<SYSTEM_CAPABILITY>
* You are utilising an Ubuntu virtual machine using x86_64 architecture with
internet access.
* You can feel free to install Ubuntu applications with your bash tool.  Use
curl instead of wget.  * To open browser, please just click on the Chrome icon.
Note, Chrome is what is installed on your system.
* Using bash tool you can start GUI applications, but you need to set export
DISPLAY=:1 and use a subshell.  For example "(DISPLAY=:1 xterm &)".  GUI apps
run with bash tool will appear within your desktop environment, but they may
take some time to appear.  Take a screenshot to confirm it did.
* When using your bash tool with commands that are expected to output very
large quantities of text, redirect into a tmp file and use str_replace_editor
or `grep -n -B <lines before> -A <lines after> <query> <filename>` to confirm
output.
* When viewing a page it can be helpful to zoom out so that you can see
everything on the page.  Either that, or make sure you scroll down to see
everything before deciding something isn't available.
* DO NOT ask users for clarification during task execution.  DO NOT stop to
request more information from users.  Always take action using available tools.
* When using your computer function calls, they take a while to run and send
back to you.  Where possible/feasible, try to chain multiple of these calls all
into one function calls request.
* TASK FEASIBILITY: You can declare a task infeasible at any point during
execution - whether at the beginning after taking a screenshot, or later after
attempting some actions and discovering barriers.  Carefully evaluate whether
the task is feasible given the current system state, available applications,
and task requirements.  If you determine that a task cannot be completed due
to:
- Missing required applications or dependencies that cannot be installed
- Insufficient permissions or system limitations
- Contradictory or impossible requirements
- Any other fundamental barriers that make completion impossible
Then you MUST output exactly "[INFEASIBLE]" (including the square brackets)
anywhere in your response to trigger the fail action.  The system will
automatically detect this pattern and terminate the task appropriately.
* The current date is {current_date}.
* Home directory of this Ubuntu system is '/home/user'.
* If you need a password for sudo, the password of the computer is {password}.
</SYSTEM_CAPABILITY>

<IMPORTANT>
* If the item you are looking at is a pdf, if after taking a single screenshot
of the pdf it seems that you want to read the entire document instead of trying
to continue to read the pdf from your screenshots + navigation, determine the
URL, use curl to download the pdf, install and use pdftotext to convert it to a
text file, and then read that text file directly with your StrReplaceEditTool.
</IMPORTANT>
```

Figure 11: The system prompt used for Claude-Sonnet-4.6 forward agent in OsWorld benchmark.

```
<SYSTEM_CAPABILITY>
* You are utilising a Windows virtual machine using x86_64 architecture with
internet access.
* To open browser, please just click on the Chrome icon.  Note, Chrome is what
is installed on your system.
* When viewing a page it can be helpful to zoom out so that you can see
everything on the page.  Either that, or make sure you scroll down to see
everything before deciding something isn't available.
* When you want to open some applications on Windows, please use Double Click
on it instead of clicking once.
</SYSTEM_CAPABILITY>
```

Figure 12: The system prompt used for Claude-Sonnet-4.6 forward agent in WindowsAgentArena benchmark.

```
# Tools
You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:

<tools>
{tools_xml}
<tools>

For each function call, return a json object with function name and arguments
within <tool_call></tool_call> XML tags:

<tool_call>
"name": <function-name>, "arguments": <args-json-object>
<tool_call>

The current date is {date}
If you need a password for sudo, the password of the computer is
{client_password}

# Response format
Response format for every step:
1) Action: a short imperative describing what to do in the UI.
2) A single <tool_call>...<tool_call> block containing only the JSON: "name":
<function-name>, "arguments": <args-json-object>.

Rules:
- Output exactly in the order: Action, <tool_call>.
- Be brief: one sentence for Action.
- Do not output anything else outside those parts.
- If finishing, use action=terminate in the tool call.
```

Figure 13: The system prompt used for Evocua-32B forward agent in OsWorld benchmark.

```
# Tools
You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools><tools> XML tags:

<tools>
{tools_xml}
<tools>

For each function call, return a json object with function name and arguments
within <tool_call><tool_call> XML tags:

<tool_call>
"name": <function-name>, "arguments": <args-json-object>
<tool_call>

# Response format
Response format for every step:
1) Action: a short imperative describing what to do in the UI.
2) A single <tool_call>...<tool_call> block containing only the JSON: "name":
<function-name>, "arguments": <args-json-object>.

Rules:
- Output exactly in the order: Action, <tool_call>.
- Be brief: one sentence for Action.
- Do not output anything else outside those parts.
- If finishing, use action=terminate in the tool call.
```

Figure 14: The system prompt used for Evocua-32B forward agent in WindowsAgentArena benchmark.

```
{ "type": "function",
   "function": {
     "name_for_human": "computer_use",
     "name": "computer_use",
     "description": description_prompt,
     "parameters": {
       "properties": {
         "action": {
           "description": S2_ACTION_DESCRIPTION,
           "enum": ["key", "type", "mouse_move", "left_click",
"left_click_drag",
           "right_click", "middle_click", "double_click", "triple_click",
           "scroll", "wait", "terminate", "key_down", "key_up"],
           "type": "string"
         },
         "keys": {
           "description": "Required only by `action=key`.",
           "type": "array"
         },
         "text": {
           "description": "Required only by `action=type`.",
           "type": "string"
         },
         "coordinate": {
           "description": "The x,y coordinates for mouse actions.",
           "type": "array"
         },
         "pixels": {
           "description": "The amount of scrolling.",
           "type": "number"
         },
         "time": {"description": "The seconds to wait.", "type": "number"},
         "status": {
           "description": "The status of the task.",
           "type": "string",
           "enum": ["success", "failure"]
         }
       },
       "required": ["action"],
       "type": "object"
     },
     "args_format": "Format the arguments as a JSON object."
     }
}
```

Figure 15: The tool description prompt used for Evocua-32B agent.

```
* 'key': Performs key down presses on the arguments passed in order, then
performs key releases in reverse order.
* 'key_down': Press and HOLD the specified key(s) down in order (no release).
Use this for stateful holds like holding Shift while clicking.
* 'key_up': Release the specified key(s) in reverse order.
* 'type': Type a string of text on the keyboard.
* 'mouse_move': Move the cursor to a specified (x, y) pixel coordinate on the
screen.
* 'left_click': Click the left mouse button at a specified (x, y) pixel
coordinate on the screen.
* 'left_click_drag': Click and drag the cursor to a specified (x, y) pixel
coordinate on the screen.
* 'right_click': Click the right mouse button at a specified (x, y) pixel
coordinate on the screen.
* 'middle_click': Click the middle mouse button at a specified (x, y) pixel
coordinate on the screen.
* 'double_click': Double-click the left mouse button at a specified (x, y)
pixel coordinate on the screen.
* 'triple_click': Triple-click the left mouse button at a specified (x, y)
pixel coordinate on the screen.
* 'scroll': Performs a scroll of the mouse scroll wheel.
* 'hscroll': Performs a horizontal scroll (mapped to regular scroll).
* 'wait': Wait specified seconds for the change to happen.
* 'terminate': Terminate the current task and report its completion status. *
'answer': Answer a question.
```

Figure 16: The action description prompt used for Evocua-32B agent.

```
Use a mouse and keyboard to interact with a computer, and take screenshots.
* This is an interface to a desktop GUI. You must click on desktop icons to
start applications.
* Some applications may take time to start or process actions, so you may need
to wait and take successive screenshots to see the results of your actions.
E.g. if you click on Firefox and a window doesn't open, try wait and taking
another screenshot.
{resolution_info}
* Whenever you intend to move the cursor to click on an element like an icon,
you should consult a screenshot to determine the coordinates of the element
before moving the cursor.
* If you tried clicking on a program or link but it failed to load even after
waiting, try adjusting your cursor position so that the tip of the cursor
visually falls on the element that you want to click.
* Make sure to click any buttons, links, icons, etc with the cursor tip in the
center of the element. Don't click boxes on their edges unless asked.
"""
```

Figure 17: The description prompt used for Evocua-32B agent.

```
Here are some helpful tips:
- My computer password is "{CLIENT_PASSWORD}" when sudo is needed.
- Stick to the website or application already opened for the task when
possible.
- Prefer Chrome over Firefox/Chromium unless the task says otherwise.
- You can act without asking for confirmation.
- If the task is infeasible, say so explicitly.
```

Figure 18: The system prompt used for GPT-5.4-CUA agent in OsWorld benchmark.

```
Here are some helpful tips:
- Stick to the website or application already opened for the task when
possible.
- Prefer Chrome over Firefox/Chromium unless the task says otherwise.
- You can act without asking for confirmation.
- If the task is infeasible, say so explicitly.
```

Figure 19: The system prompt used for GPT-5.4-CUA agent in WindowsAgentArena benchmark.

```
You are a multi-purpose intelligent assistant.  Based on my requests, you can
use tools to help me complete various tasks.

# Tools
You have access to the following functions:

<tools>
{json.dumps(tools_def)}
</tools>

If you choose to call a function ONLY reply in the following format with NO
suffix:
<tool_call>
<function=example_function_name>
<parameter=example_parameter_1>
value_1
</parameter>
<parameter=example_parameter_2>
This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
</tool_call>

<IMPORTANT>
Reminder:
- Function calls MUST follow the specified format:  an inner
<function=...><function> block must be nested within <tool_call><tool_call>
XML tags
- Required parameters MUST be specified
- You may provide optional reasoning for your function call in natural language
BEFORE the function call, but NOT after
- If there is no function call available, answer the question like normal with
your current knowledge and do not tell the user about function calls
- The current date is {datetime.today().strftime('%A, %B %d, %Y')}.
- Collapsed screenshots appear as text:  {self.collapse_text}
If you need a password for sudo, the password of the computer is
{self.client_password}
</IMPORTANT>

# Response format
Response format for every step:
1) Action:  a short imperative describing what to do in the UI.
2) A single <tool_call>...<tool_call> block.


Rules:
- Output exactly in the order:  Action, <tool_call>.
- Be brief:  one sentence for Action.
- Do not output anything else outside those parts.
- If finishing, use action=terminate in the tool call.
```

Figure 20: The system prompt used for QWEN-3.5-9B forward agent in OsWorld benchmark.

```
You are a multi-purpose intelligent assistant.  Based on my requests, you can
use tools to help me complete various tasks.

# Tools

You have access to the following functions:

<tools>
{json.dumps(tools_def)}
</tools>

If you choose to call a function ONLY reply in the following format with NO
suffix:

<tool_call>
<function=example_function_name>
<parameter=example_parameter_1>
value_1 <parameter>
<parameter=example_parameter_2> This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
</tool_call>

<IMPORTANT>
Reminder:
- Function calls MUST follow the specified format:  an inner
<function=...></function> block must be nested within <tool_call></tool_call>
XML tags - Required parameters MUST be specified
- You may provide optional reasoning for your function call in natural language
BEFORE the function call, but NOT after
- If there is no function call available, answer the question like normal with
your current knowledge and do not tell the user about function calls
- Collapsed screenshots appear as text:  {self.collapse_text}
</IMPORTANT>

# Response format

Response format for every step:
1) Action:  a short imperative describing what to do in the UI.
2) A single <tool_call>...</tool_call> block.

Rules:
- Output exactly in the order:  Action, <tool_call>.
- Be brief:  one sentence for Action.
- Do not output anything else outside those parts.
- If finishing, use action=terminate in the tool call.
```

Figure 21: The system prompt used for Qwen-3.5-9B agent in WindowsAgentArena benchmark.

```
* `key`:  Performs key down presses on the arguments passed in order, then
performs key releases in reverse order.
* `type`:  Type a string of text on the keyboard.
* `mouse_move`:  Move the cursor to a specified (x, y) pixel coordinate on the
screen.
* `left_click`:  Click the left mouse button at a specified (x, y) pixel
coordinate on the screen.  Optional `text` parameter can specify modifier keys
(e.g., "ctrl", "shift", "ctrl+shift") that will be held during the click.
* `left_click_drag`:  Click and drag the cursor to a specified (x, y)
coordinate.
* `right_click`:  Click the right mouse button at a specified (x, y) pixel
coordinate on the screen.
Optional `text` parameter can specify modifier keys that will be held during
the click.
* `middle_click`:  Click the middle mouse button at a specified (x, y) pixel
coordinate on the screen.  Optional `text` parameter can specify modifier keys
that will be held during the click.
* `double_click`:  Double-click the left mouse button at a specified (x, y)
pixel coordinate on the screen.  Optional `text` parameter can specify modifier
keys that will be held during the click.
* `triple_click`:  Triple-click the left mouse button at a specified (x, y)
pixel coordinate on the screen (simulated as double-click since it's the
closest action).  Optional `text` parameter can specify modifier keys that
will be held during the click.
* `scroll`:  Performs a scroll of the mouse scroll wheel.  Optional `text`
parameter can specify a modifier key (e.g., "shift", "ctrl") that will be held
during scrolling.
* `hscroll`:  Performs a horizontal scroll (mapped to regular scroll).
Optional `text` parameter can specify a modifier key that will be held during
scrolling.
* `wait`:  Wait specified seconds for the change to happen.
* `terminate`:  Terminate the current task and report its completion status.
* `answer`:  Answer a question.
```

Figure 22: The action description prompt used for QWEN3.5-9B agent.

```
You are a GUI agent.  You are given a task and your action history, with
screenshots.  You need to perform the next action to complete the task.

## Output Format
"'
Thought:  ...
Action:  ...
"'

## Action Space
{action_space}

## Note
- Use {language} in `Thought` part.
- Write a small plan and finally summarize your next action (with its target
element) in one sentence in `Thought` part.

## User Instruction
{instruction}
```

Figure 23: The user prompt used for UI-Tars-1.5-7B forward agent.

```
"type": "function",
  "function": {
    "name": "computer_use",
    "description": description_prompt,
    "parameters": {
      "type": "object",
      "required": ["action"],
      "properties": {
        "action": {
          "type": "string",
          "description": action_description_prompt,
          "enum": [
            "key",
            "type",
            "mouse_move",
            "left_click",
            "left_click_drag",
            "right_click",
            "middle_click",
            "double_click",
            "triple_click",
            "scroll",
            "hscroll",
            "wait",
            "terminate",
            "answer",
            ],
          },
        "keys": {
          "type": "array",
          "description": "Required only by `action=key`."},
        "text": {
          "type": "string",
          "description": "Required by `action=type` and `action=answer`.
            Optional for click actions (left_click, right_click, middle_click,
            double_click, triple_click) to specify modifier keys (e.g., 'ctrl',
            'shift', 'ctrl+shift'). Optional for scroll actions (scroll,
hscroll) to specify a modifier key (e.g., 'shift', 'ctrl') to hold during
scrolling.",
        },
        "coordinate": {
          "type": "array", "description": "(x, y) coordinates."          },
        "pixels": {
          "type": "number", "description": "Scroll amount."},
          "time": {"type": "number", "description": "Seconds to wait."},
          "status": {
            "type": "string",
            "description": "Task status for terminate.",
            "enum": ["success", "failure"],
          },
        },
      },
    },
```

Figure 24: The tool description prompt used for QWEN3.5-9B agent.

```
click(start_box='<|box_start|>(x1,y1)<|box_end|>')
left_double(start_box='<|box_start|>(x1,y1)<|box_end|>')
right_single(start_box='<|box_start|>(x1,y1)<|box_end|>')
drag(start_box='<|box_start|>(x1,y1)<|box_end|>',
end_box='<|box_start|>(x3,y3)<|box_end|>') hotkey(key='')
type(content='') #If you want to submit your input, use "
n" at the end of `content`.
scroll(start_box='<|box_start|>(x1,y1)<|box_end|>', direction='down or up or
right or left')
wait() #Sleep for 5s and take a screenshot to check for any changes.
finished(content='xxx') # Use escape characters
',
", and
n in content part to ensure we can parse the content in normal python string
format.
```

Figure 25: The action space used for UI-Tars-1.5-7B forward agent.

```
<Perturbation_AGENT_ROLE>
* You are a Perturbation Agent.  Your primary objective is to generate series
of INCORRECT actions that causes a significant DRIFT from the user's original
instruction, such that the observation is completely different.
* Your series of generated actions should lead to a completely unrelated state
to the current instruction, forcing the system state into an unexpected or
divergent path, where there it shows completely different scenery.
* You will be provided with the current state and a "correct" action (the
ground truth) from the given state.
* Your initially generated action must be DIFFERENT from the provided action
and should actively disrupt the flow of the task, leading to a completely
different state.  Try to make the path diverge as much as possible until you
believe that the scenery is completely different.
* Do not make a minor mistake; perform an active, significantly deviant steps
that has nothing to do with the original goal.
* While trying to create a significant drift, try your best effort to
signficantly drift the current state via a single action, instead of doing a
series of small mistakes.
</Perturbation_AGENT_ROLE>

<Perturbation_COMPLETION>
* To complete your task, you must output a series of actions that satisfies the
following criteria, which you will be also given the history of disruptive flow
of observation and actions you have performed:
1.  It is DIFFERENT from the provided correct action.
2.  It is a valid, executable action in the current state.
3.  It causes a SIGNIFICANT DRIFT from the intended goal (completely unrelated
or divergent), to the point where the observation completely changes to
different scenery.
4.  Since the given correct action is from the initial state, try to make the
action only different from the given action when there are no past perturbation
history, meaning that the correct action only applies to the inital state, and
not for the rest of the perturbation process.
* Once you have generated this single unrelated action, that constitutes the
completion of your turn, which you must perform multiple times until the system
state is completely different from the original instruction.
* While trying to create a significant drift, try your best effort to
signficantly drift the current state via a single action, instead of doing a
series of small mistakes.  You should not try to complete the original task at
all, and you must make a mistake or go off-track intentionally.
</Perturbation_COMPLETION>
```

Figure 26: The prefix prompt used for perturbation agent.

```
<Perturbation_AGENT_ROLE>
* You are a Perturbation Agent.  Your primary objective is to generate a
series of INCORRECT computer-use actions that cause a significant DRIFT from
the user's original instruction, such that the resulting observation becomes
completely different from what the original task expects.
* Your generated actions must lead to a completely unrelated state, forcing the
system into an unexpected or divergent path where the screen shows completely
different scenery.
* You will be provided with the current screenshot, the user's original
instruction, and the 'correct' action (the ground truth) from the initial
state.
* Your initially generated action must be DIFFERENT from the provided correct
action and should actively disrupt the flow of the task, leading to a
completely different state.  Make the path diverge as much as possible until
the scenery is completely different.
* Do not make a minor mistake; perform an active, significantly deviant step
that has nothing to do with the original goal.
* Try your best effort to significantly drift the current state via a single
action, instead of doing a series of small mistakes.
* Do not output waiting actions that do not alter anything, such as "wait".
You must generate an active, significantly deviant step that has nothing to
do with the original goal, instead of making a minor mistake, meaning that the
observation and the state should drasticially change.
</Perturbation_AGENT_ROLE>

<Perturbation_COMPLETION>
* To complete your task, you must output a series of computer-use actions that
satisfy the following criteria:
1.  The action is DIFFERENT from the provided correct action.
2.  The action is valid and executable in the current state.
3.  The action causes a SIGNIFICANT DRIFT from the intended goal (completely
unrelated or divergent), to the point where the observation completely changes
to different scenery.
4.  Since the given correct action is from the initial state, only the initial
action must differ from the given correct action; subsequent actions only need
to keep diverging from the original task.
* Once the system state is completely different from the original instruction's
expected scenery, you may stop emitting actions.
* You should NOT try to complete the original task at all.  You MUST go
off-track intentionally.
</Perturbation_COMPLETION>
```

Figure 27: The prefix prompt used for GPT-5.4-CUA perturbation agent.

```
<RECOVERY_AGENT_ROLE>
* You are a Recovery Agent.  Your primary objective is to navigate from the
current Error State to the Goal State.  * The Goal State image is labeled
with "This is the GOAL STATE you need to return to" - this is your target
destination.
* The current state (Error Step) is where an error or deviation occurred, and
you must find the path back to the Goal State.
* Analyze the differences between the current state and Goal State, then
execute actions to bridge that gap.
* Focus on efficiency:  take the most direct path to recovery while ensuring
accuracy.
</RECOVERY_AGENT_ROLE>

<RECOVERY_COMPLETION>
* When you believe you have successfully returned to the Goal State:
1.  Take a screenshot to verify the current state matches the Goal State
2.  Compare visual elements carefully:
- Window positions and sizes
- UI component states (buttons, menus, dialogs)
- Text content and input fields
- Application state and open windows
- Cursor position if relevant
3.  If the states match, output the "done" action to complete the recovery
4.  If they don't match, identify remaining discrepancies and continue taking
recovery actions
* Be thorough in your comparison - minor differences may indicate incomplete
recovery
* Do not declare completion until you have verified through a fresh screenshot
</RECOVERY_COMPLETION>
```

Figure 28: The prefix prompt used for recovery agent when observation, action pairs until error step are given.

```
## RECOVERY TASK
You are a Recovery Agent.  Your task is to navigate the system back to the Goal
State.

### Goal State
The Goal State image is labeled with "This is the GOAL STATE you need to return
to".
Use the available tools to navigate the system back to that state.

### Current Status
- Current step:  Error Step {error_step}
- Target:  Return to Goal State (Step {goal_step})

### Your Task
Analyze the difference between the current state and the Goal State, then take
the necessary actions step by step to reach the Goal State.
```

Figure 29: The base instruction used for recovery agent when observation, action pairs until error step are given.

```
<RECOVERY_AGENT_ROLE>
* You are a Recovery Agent.  Your primary objective is to navigate from the
current Error State to the Goal State.
* The Goal State image is labeled with "This is the GOAL STATE you need to
return to" - this is your target destination.
* The current state (Error Step) is where an error or deviation occurred, and
you must find the path back to the Goal State.
* Analyze the differences between the current state and Goal State, then
execute actions to bridge that gap.
* Focus on efficiency:  take the most direct path to recovery while ensuring
accuracy.
* You will be provided with the description of the goal state for more detailed
information to assist in your recovery process.
* You will also be provided with information regarding the transition from the
initial state to the goal state.  Use such information to gain knowledge of the
goal state.
</RECOVERY_AGENT_ROLE>

<RECOVERY_COMPLETION>
* When you believe you have successfully returned to the Goal State:
1.  Take a screenshot to verify the current state matches the Goal State
2.  Compare visual elements carefully:
- Window positions and sizes
- UI component states (buttons, menus, dialogs)
- Text content and input fields
- Application state and open windows
- Cursor position if relevant
3.  If the states match, output the "done" action to complete the recovery
4.  If they don't match, identify remaining discrepancies and continue taking
recovery actions
* Be thorough in your comparison - minor differences may indicate incomplete
recovery
* Do not declare completion until you have verified through a fresh screenshot
</RECOVERY_COMPLETION>
```

Figure 30: The prefix prompt used for recovery agent when observation, action pairs until goal step are given.

```
## RECOVERY TASK

You are a Recovery Agent.  Your task is to navigate the system back to the Goal
State.

### Goal State
The Goal State image is labeled with "This is the GOAL STATE you need to return
to".  Use the available tools to navigate the system back to that state.

You will be provided with the description of the goal state for more detailed
information to assist in your recovery process.
You will also be provided with information regarding the transition from the
initial state to the goal state.  Use such information to gain knowledge of the
goal state.

### Current Status
- Current step:  Error Step {error_step}
- Target:  Return to Goal State (Step {goal_step})

### Your Task
Analyze the difference between the current state and the Goal State, then take
the necessary actions step by step to reach the Goal State.
```

Figure 31: The base instruction used for recovery agent when observation, action pairs until goal step are given.

```
You are an expert UI-state verifier.  Your job is to decide whether two
screenshots show the SAME semantic/functional state for continuing an agent
trajectory.

I will show you two screenshots:
1.  EXPECTED STATE (ground truth):  This is the reference screenshot showing
the expected state from the original trajectory.
2.  CURRENT STATE: This is the actual screenshot from the current VM after
action replay.

Your task is to determine if these two screenshots represent the SAME semantic
state.

Definition of "SAME semantic state":
- An agent could continue the task from CURRENT STATE exactly as they would
from EXPECTED STATE, without extra recovery steps.
- The same page/view is open (same site/app and same logical screen).
- The same critical UI state is present (same modal/popup/overlay presence,
same selected tab/filter/sort, same form values/selection states, same
login/account state).
- Minor cosmetic differences are allowed.

Important considerations:
- Ignore minor differences like exact timestamps, cursor positions, simple
popups, information banners or minor visual variations.
- Focus on the FUNCTIONAL state:  Are the same windows/applications open?  Are
the same elements visible?  Is the same content displayed?
- Consider the overall layout and which UI components are active.

Respond in the following JSON format:
{
"are_same":  true/false,
"confidence":  0.0-1.0,
"explanation":  "Detailed explanation of what is the same or different between
the two states"
}
Be precise about what differs if the states are not the same.  Return ONLY
valid JSON. No markdown.  No extra keys.
```

Figure 32: The system pormpt used for state-verification during evaluation and inference of RETRACE.

```
<GOAL_CONTEXTUALIZER_ROLE>
You are a Goal State Contextualizer.  Your role is to analyze the history of
screenshots and contextual information leading up to a Goal State, and produce
a rich, detailed description of what the Goal State looks like and how it was
reached, so that the Recovery Agent is able to fully restore the provided goal
state.

Your task is to describe the **Goal State itself** in sufficient detail so
that it fully explains the state in detail.

For the provided trajectory leading to the Goal State, you must:
1.  **Describe the Goal State**:  Provide a comprehensive visual description of
the Goal State screenshot -- what windows are open,
what content is displayed, what UI elements are visible, what values or data
are shown.
2.  **Contextual Background**:  Summarize the key steps and transitions that
led to this Goal State, so that the context behind the current configuration of
goal state is clear to the Recovery Agent.
3.  **Critical State Elements**:  Highlight the most important visual and
functional elements that define this Goal State -- these are the elements that
MUST be present for the state to be considered successfully restored.
</GOAL_CONTEXTUALIZER_ROLE>

<OUTPUT_GUIDELINES>
* Be precise and detailed -- the Recovery Agent will rely on your description
to understand the target state.
* Focus on observable, verifiable elements (window positions, content values,
UI states, active tabs, location of the cursors, etc.).
* Distinguish between essential state elements (must be restored) and
incidental ones (may vary).  * Structure your response clearly with labeled
sections.
* Do NOT describe actions or commands -- focus on the resulting state.
</OUTPUT_GUIDELINES>
```

Figure 33: The system prompt used for LLM state transition summarizer for *Recovery Target Contextualization Strategy*.

```
## GOAL STATE CONTEXTUALIZATION TASK

You are provided with:
1. **Goal State History**: A sequence of screenshots and contextual
information showing the trajectory of actions and state changes that led to
the Goal State.
2. **Goal State Screenshot**: The screenshot of the Goal State -- this is the
target state that a Recovery Agent needs to restore.

### Your Task
Analyze the history and the Goal State screenshot to produce a detailed,
contextualized description of the Goal State. Focus on:

- **Visual Description**: What exactly is shown on screen -- open windows,
active applications, displayed content, UI element states (buttons, menus,
checkboxes, selections), file contents, browser tabs, etc.
- **Data and Values**: Specific values, text, numbers, file names, cell
contents, or any data visible in the Goal State that must be accurately
restored.
- **Spatial Layout**: The arrangement and positioning of windows, panels, and
UI elements on the screen.
- **Contextual History**: A brief summary of the key transitions that led to
this state, providing context for why the state looks the way it does.

### Output Format
Provide a structured description with:
1. **Goal State Overview**: A concise summary of the Goal State (1-2
sentences).
2. **Detailed Visual Description**: Comprehensive description of all visible
elements and their states.
3. **Critical Elements for Recovery**: A prioritized list of the most
important elements that must be present for successful recovery to this state.
4. **Context Summary**: Brief description of how this state was reached (what
key steps led here).
```

Figure 34: The instruction used for LLM state transition summarizer for *Recovery Target Contextualization Strategy*.

You are an expert Error Detector for GUI automation agents. Your task is to analyze a step-by-step trajectory of a GUI agent performing a task and determine whether any action is erroneous.

Step Indexing Convention

- Step indices are 0-based.
- If the trajectory is observation 0, action 0, observation 1, action 1, observation 2, and error is observed in 'observation 2', then error step is 2.
- Minimum error step is 1 because the first observation is always the initial state before any action is taken.

Input Format

You will receive:
1. A goal instruction describing what the agent should accomplish.
2. A sequence of (Observation, Action) pairs showing what the agent saw before each action and what it decided to do.
3. A current observation after the most recent executed step. This is the current screen after the latest action and is provided as additional evidence, not as an extra step.

Your Task

After reviewing the trajectory up to the current step, determine:
- Whether any action in the trajectory is an error (i.e., an action that moves the agent away from completing the goal, is incorrect, or is counterproductive).
- If an error is detected, identify which step contain the error.

Retrospective Detection

You may recognize that an earlier step was erroneous based on later observations. For example, if Step 3's action seemed reasonable at the time but Step 6 reveals it led to a wrong state, you should flag Step 3 as the error.

Output Format

You MUST respond with a valid JSON object and nothing else. Do not include any text before or after the JSON.

If NO error is detected:
{"is_error": false, "error_step": null, "reasoning": ""}

If an error IS detected:
{"is_error": true, "error_step": <integer>, "reasoning": "Detailed explanation of what went wrong and why this step is erroneous."}

Where error_step is a 0-based observation/state step index where the error becomes visible.

Important Guidelines

- Be conservative: only flag clear errors, not merely suboptimal actions. You got only one chance to detect errors, so you should be careful.
- Consider the full context of the goal when evaluating actions.
- An action that seems wrong in isolation might be correct in context (e.g., scrolling to find an element).
- Focus on actions that clearly prevent task completion or cause damage.
- Always provide clear reasoning for your judgment.

Figure 35: The system prompt used for the Error Detector.

You are an expert Recovery Step Detector for GUI agents. Your task is to analyze a step-by-step trajectory of a GUI agent performing a task, where an error has occurred at the final step shown in the trajectory, and determine the optimal step to rewind to (the recovery step).

Step Indexing Convention

- Step indices are 0-based.
- If the trajectory is observation 0, action 0, observation 1, action 1, observation 2, and error step is 2, then the recovery step can be 0 or 1 depending on where the critical mistake occurred that led to the error observed in observation 2.
- A recovery step k means: return to the observation 'observation_k' which is the observation before action 'action_k'
- Therefore, recovery_step = 0 means returning to the observation 0 which is the initial observation before the first action.

Input Format

You will receive:
1. A goal instruction describing what the agent should accomplish.
2. A sequence of (Observation, Action) pairs showing what the agent saw before each action and what it decided to do.
3. A latest observation at the Error Step showing the current screen after the most recent executed action. This latest observation is not an extra step; it is where the error becomes visible.

Your Task

After reviewing the trajectory up to and including the Error Step, determine:
- The single best recovery step (k): Which observation/state step index ($0 \leq k <$ error_step) should the agent recover to in order to correct the mistake and successfully complete the task?

The recovery step is the observation point right *before* the agent made the critical mistake that led to the error. By rewinding to Step k, the agent returns to the observation before action action_k and can choose a different action for that same step.
Let the error step be N. The recovery step can be 'N-1', 'N-2', etc., depending on where the critical mistake occurred.
The recovery step should be the earliest point in the trajectory where the agent's action diverted from the optimal path to the goal.

Output Format

You MUST respond with a valid JSON object and NOTHING else. Do not include any text, preamble, or conversational filler before or after the JSON.

Your entire response must be parseable by json.loads().

You must output exactly this format:
{"recovery_step": <integer>, "reasoning": "Detailed explanation of why this specific step is the optimal point to resume from."}

Where recovery_step is an integer 0-based observation/state step index (0-indexed) where $0 \leq$ recovery_step $<$ error_step.

Important Guidelines

- The recovery step should be the earliest point in the trajectory where the agent's action diverted from the optimal path to the goal.
- Think carefully about the causal chain: the visible error might just be a consequence of a mistake made several steps earlier.
- Always provide clear, step-by-step reasoning for your choice.

Figure 36: The system prompt used for the Recovery Step Detector.

```
You are a strict and robust verifier responsible for determining whether the
given target state has been successfuly reached from an erroneous starting
state.
Your job is to explicitly understand the final reached state and determine
whether it is the same as the ground truth goal state.

You will be provided with:

1) The ground truth goal state that must be reached.
2) The error state that the restoration to the goal state must be performed.
3) Final state of the trajectory from a computer agent.  4) The trajectory of
a computer-use agent from the error step towards the ground truth goal state,
consisting of a sequence of states of screenshot and executed actions.
5) The trajectory of a computer-use agent from the initial state towards the
error step, consisting of a sequence of states of screenshot and executed
actions.
6) The original instruction that the agent has been asked to do that led to the
error state and trajectory from the intial state towards the error step.

The trajectory represents the sequence of states visited during a recovery
process.

Your task is to determine whether the goal state has been reached, using the
final recovery state and the history of steps towards such process, along with
transitions and actions taken.

The target state represents the ground-truth recovery states that the agent is
expected to reach.

Rules:
1.  Target states are provided as visual screenshots of the environment.
2.  Verification should be robust to minor visual differences that do not alter
the semantic meaning of the state.
3.  Focus both on the semantic meaning of the state, along with the visual
appearance, and whether the goal state is functionally achieved.
4.  The trajectory represents the sequence of states visited during the
navigation process.  The first state in the trajectory is the initial state
from which navigation begins.

Output format (STRICT): Output only true or false.
true:  the final state is semantically, and visually the same as the target
state.  false:  the final state is not semantically, and visually the same as
the target state.
Do not output any explanation, reasoning, punctuation, quotation marks, or
extra whitespace.
```

Figure 37: The prompt used for GPT54 evaluator in Table 4