A History-Aware Visually Grounded Critic for Computer Use Agents

Jaewoo Lee[1] **Zaid Khan**[1] **Archiki Prasad**[1] **Justin Chih-Yao Chen**[1]
Supriyo Chakraborty[2] **Kartik Balasubramaniam**[2] **Sambit Sahu**[2]
Elias Stengel-Eskin[3] **Hyunji Lee**[1] **Mohit Bansal**[1]

University of North Carolina at Chapel Hill[1], Capital One[2], University of Texas at Austin[3]

Abstract

Various test-time interventions for Computer Use Agents (CUAs), including critic models, have been developed to improve performance through pre-execution action evaluation in complex Graphical User Interface (GUI) environments. However, existing critics suffer from two key limitations: they (1) focus primarily on *short-sighted decision loops* (e.g., forgetting earlier actions) and (2) *lack the visual grounding* needed to detect flawed actions (e.g., clicking wrong UI elements). To address these, we introduce HiViG, a **Hi**story-aware **Vis**ually **G**rounded test-time framework, built around a multimodal critic trained on real GUI trajectories to abstract past interactions into a compact record and to evaluate actions with visual grounding. At test time, HiViG integrates the critic into the policy decision loop to provide *macro-action history*, which summarizes the policy's completed achievements, and *visually grounded critique*, which verifies raw execution coordinates against the current screenshot to intercept errors before execution. Across web, mobile, and desktop benchmarks, HiViG consistently outperforms existing scalar and verbal critics, improving average success rates over the strongest baseline by 5.8% for Qwen3-VL-32B and 9.0% for Gemini-3-Flash, and demonstrates strong cross-platform generalization. Ablations show that macro-action history mitigates short-sighted planning and visually grounded critique reduces execution errors, with both components being critical for test-time scaling in long-horizon GUI tasks.[1]

1 Introduction

Computer Use Agents (CUAs) are widely used to automate long-horizon tasks in Graphical User Interface (GUI) environments (Deng et al., 2023; He et al., 2024; Koh et al., 2024). The underlying policy models inevitably make mistakes while operating directly on complex visual interfaces (Hong et al., 2024; Qin et al., 2025; Awadallah et al., 2025), such as selecting incorrect UI elements. Since many GUI actions are irreversible (Wu et al., 2025a), relying on post-execution correction is often unsafe and impractical, motivating the use of test-time intervention to prevent errors before execution. To provide such intervention, existing work employs reward models (Chae et al., 2025b; Xiong et al., 2025; Zhang et al., 2026) that offer *scalar feedback* to score candidate actions. However, in GUI environments with continuous parameter spaces (e.g., pixel coordinates), scoring alone is uninformative when all candidates are poor, trapping the policy in failure modes with no path to improvement (Luo et al., 2025; Ning et al., 2026). In contrast, *natural language (verbal) feedback* can explain why an action fails, enabling the policy to recover and progress toward task completion. Thus, critic models that provide this kind of verbal feedback have emerged as a promising alternative to scalar reward models for test-time intervention.

Despite this conceptual shift, effective test-time critique, which serves as a pre-execution action evaluation to intercept a policy's proposed action before the action alters the environment, requires two underexplored capabilities by existing approaches. First, a reliable critic must maintain **visual grounding**. Existing methods tend to rely on the policy's verbalized action intent (e.g., 'double-click on "SpecialProjects"' shown in Figure 1 (Right)). As a result, they may erroneously approve a logically sound intent that is visually misaligned, such as an intent targeting incorrect coordinates or hallucinating state-transitions (Chae et al., 2025a; Zheng et al., 2026). This risks allowing spatial or reasoning errors to bypass the pre-execution action evaluation. To help a policy maintain task progress, a critic needs to keep track of **historical state** (i.e., what has been accomplished and failed).

Our work aims to fill these gaps by introduc-

[1]Code available at https://github.com/G-JWLee/HiViG

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 45877–45908
October 24-29, 2026 ©2026 Association for Computational Linguistics

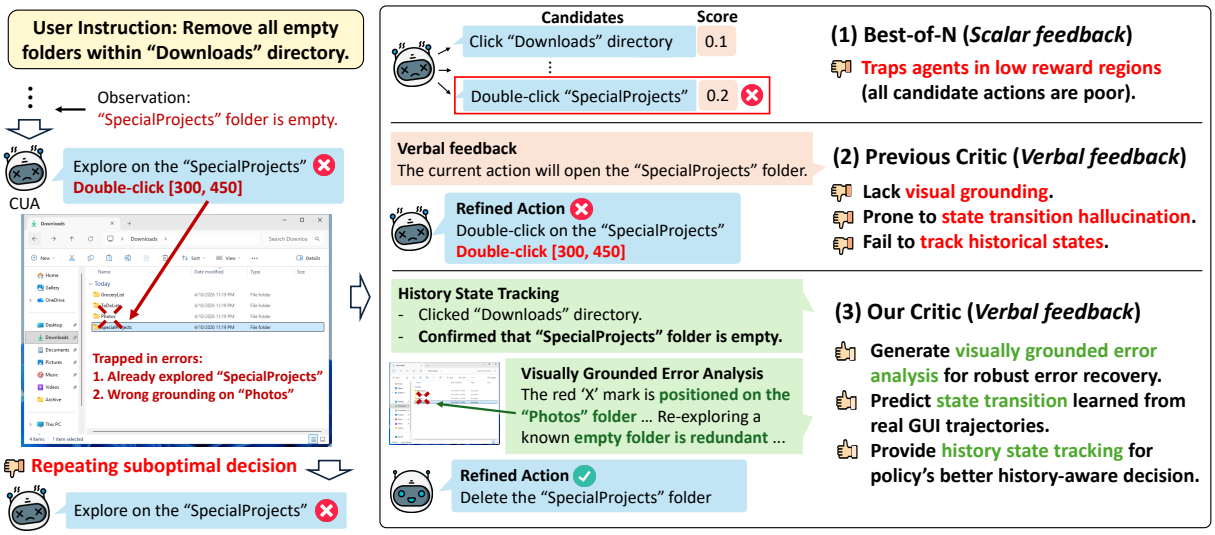


Figure 1: Comparison of test-time interventions for Computer Use Agents (CUAs). **Left**: Lacking historical awareness and proactive error-recovery, standard policies easily become trapped in short-sighted decision loops. **Right**: Existing approaches are limited. Scalar feedback (top) traps policies in low-reward trajectory regions when all candidate actions are suboptimal. Previous critics (middle) rely heavily on textual intent, missing spatial and reasoning errors, and fail to provide historical awareness. In contrast, our critic (bottom) verifies raw execution coordinates, predicts immediate visual state outcomes grounded in its learned state transition knowledge, and provides visually grounded error analysis that intercepts errors before execution. Furthermore, it equips agents with history state tracking, condensing past interactions to guide them toward the final task objective.

ing **His**tory-aware **Vis**ually **G**rounded (HIVIG) test-time intervention framework Figure 1 (Right-bottom). HIVIG resolves prior shortcomings by equipping CUAs with two core capabilities. First, HIVIG equips the policies with the ability to **track historical state**, enabling better history-aware planning over long horizons. To this end, HIVIG provides a macro-action history that summarizes past interactions to date. By recursively compressing past interactions into multi-step achieved goals (e.g., "Successfully opened the 'Downloads' directory and confirmed the 'SpecialProjects' folder is empty"), this helps the policy to track global task completion and avoid redundant decisions. Second, to address the lack of visual grounding, HIVIG performs **visually grounded error analysis**. Rather than overly relying on policy's textual intents, HIVIG verifies raw execution coordinates against actual visual states. If a proposed action is flawed, our framework identifies the error dimension (e.g., visual hallucination, termination misjudgment) to provide the policy with corrective guidance before execution. Inside our framework, we propose HIVIG-critic, a multimodal model trained to serve as a method for test-time intervention with these dual critique generation capabilities. To this end, we construct a training corpus derived from open-sourced, multi-domain GUI trajectories (Liu et al., 2026). First, to teach history state tracking, we train the critic to update a macro-action history by integrating the last visual change with the past macro-action history to track long-term goal achievement (Figure 3 (Top-right)). Second, to teach visually grounded error analysis, we train the critic to evaluate successful and flawed actions through a reasoning process: it verifies execution coordinates against the current screenshot for visual grounding, predicts the visual state-transition to assess the action's causal effect, and evaluates the action's relevance to the task (Figure 3 (Bottom-right)).

To validate the efficacy of HIVIG, we conduct extensive evaluations across three diverse, long-horizon GUI benchmarks: WebArenaLitev2 (web; Liu et al., 2026), AndroidLab (mobile; Xu et al., 2025b), and WindowsAgentArena (desktop; Bonatti et al., 2025). We evaluate our test-time intervention on two CUA policies: Qwen3-VL-32B-Thinking (Qwen Team, 2025) and Gemini-3-Flash (Gemini Team, 2025), representing open- and closed-weight models capable of navigating complex visual interfaces. Our empirical results Table 1 show that existing scalar rewards and visually ungrounded critics frequently degrade the performance of already highly-capable policies. In contrast, HIVIG improves Gemini-3-Flash's over-

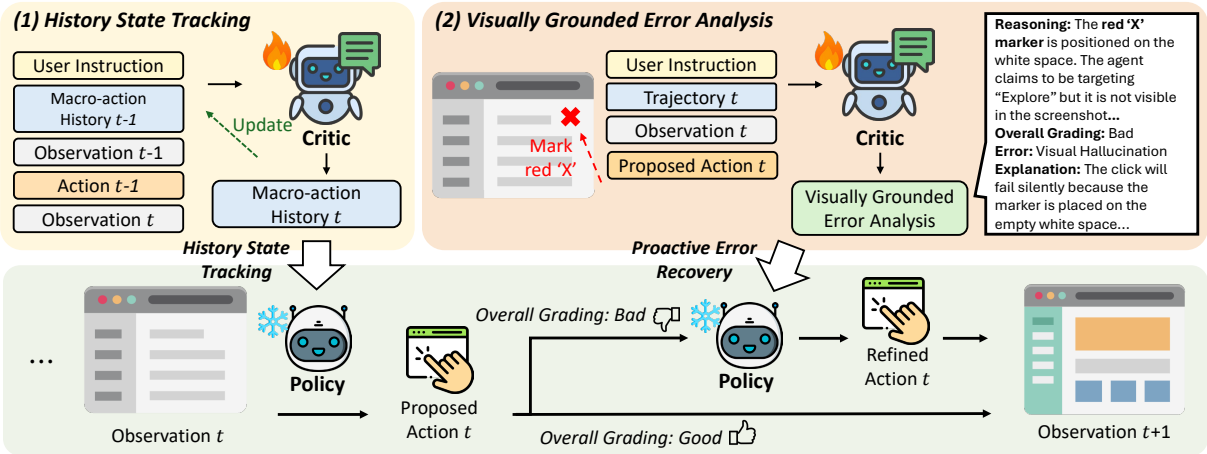


Figure 2: Overview of HiVig test-time intervention framework. **Top-left:** By analyzing visual changes between consecutive states, our critic updates the prior macro-action history, compressing micro-steps into macro-achievements. **Top-right:** our critic verifies raw pixel coordinates against the current visual state and predicts the impact of the proposed action, thereby catching spatial or reasoning errors before execution, outputting specific error dimensions to drive proactive error recovery. **Bottom:** At test time, the policy uses the macro-action history for better decision-making, and uses the visually grounded error analysis to refine the flawed action before execution.

all success rate by 15.0% (30.5% to 45.5%) on WebArenaLitev2 benchmark. HiVig also generalizes across different GUI environments, outperforming the strongest baseline critics by 9.0% and 5.8% when guiding Gemini-3-Flash and Qwen3-VL-32B-Thinking, respectively. Overall, HiVig is an test-time intervention framework that effectively guides strong policies to better complete long-horizon GUI tasks with history state tracking and visually grounded error analysis.

2 History-aware Visually Grounded Test-time Intervention (HiVig)

We first introduce the preliminaries of the CUA paradigm in Section 2.1. We then describe the data construction of two supervised fine-tuning (SFT) datasets jointly used to train the HiVig-critic in Section 2.2 for *history state tracking* and *visually grounded error analysis*. Finally, in Section 2.3, we present the overall HiVig framework (Figure 2) and describe how the trained critic is deployed at test time to guide policies.

2.1 Preliminary: Computer Use Agent (CUA)

To automate digital tasks, we adopt the standard CUA paradigm, where a Multimodal Large Language Model (MLLM) operates as a policy that generates actions to interact with a Graphical User Interface (GUI) environment to fulfill a user instruction I (Wang et al., 2025; Xu et al., 2026). As illustrated in Figure 2 (Bottom), at each timestep t, the environment provides a raw visual observation o_t (i.e., a GUI screenshot). The policy π then generates a subsequent action a_t conditioned on the instruction, the current observation o_t, and the execution history H_{t-1}, i.e., $a_t \sim \pi(\cdot|I,\ H_{t-1},\ o_t)$. The trajectory $H_{t-1} = (a_1,\ a_2,\ \ldots,\ o_{t-W+1},\ a_{t-W+1},\ \ldots,\ o_{t-1},\ a_{t-1})$ is a structured history that, to manage MLLM context limitations in long-horizon GUI tasks, retains the complete sequence of past actions together with only the most recent W visual observations (Qwen Team, 2025; Wang et al., 2025; Xu et al., 2026). The generated action a_t typically consists of verbalized intent (e.g., 'Click the browser's Back button in the top-left corner') and a pixel-based mouse or keyboard event (e.g., `{"action": "left_click", "coordinate": [10, 20]}`). The environment executes a_t and transitions to the next visual state o_{t+1}, repeating until the policy generates a termination action or exhausts its computational budget.

2.2 Critic Data Construction

We construct two distinct datasets to train HiVig-critic for two capabilities: **(1) history state tracking** and **(2) visually grounded error analysis**, as shown in Figure 3. Specifically, we leverage an off-the-shelf MLLM as an annotator to automatically derive data from open-sourced, multi-domain GUI trajectories from ScaleCUA (Liu et al., 2026) training corpus. By conditioning the annotator on the verified action execution outcomes present in the source corpus, we ensure the resulting data is grounded in environmental reality rather than the

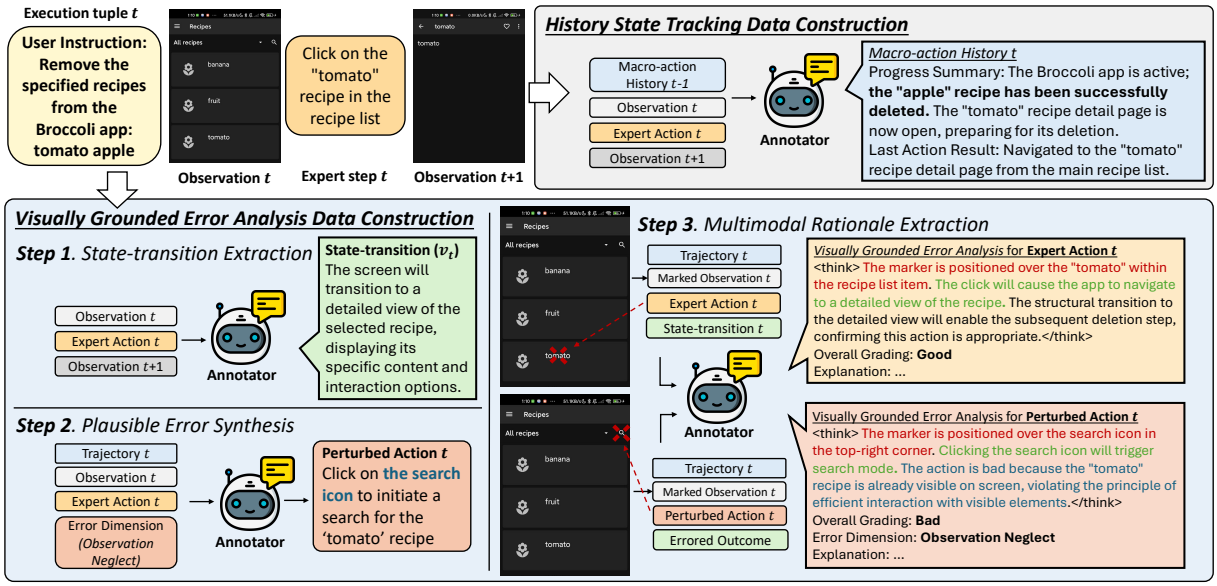


Figure 3: Data construction for HiVIG. From the execution tuple (top left), we construct two distinct SFT datasets. **Top**: For history state tracking, the annotator iteratively translates visual state changes into a compact macro-action history to track long-term goal progress. **Bottom**: For visually grounded error analysis, we extract ground-truth state-transitions (Step 1) and synthesize plausible errors (Step 2). In Step 3, the annotator generates a multi-stage rationale that leverages a visual marker (red 'X') for visual grounding and the extracted state-transitions. (Note: User instructions are omitted here for simplicity.)

annotator's parametric reasoning.

History State Tracking. To train our critic to track and supply the macro-action history, we construct a training dataset using an iterative compression mechanism (Figure 3 (Top-right)). First, given a successful trajectory, i.e., $(o_1, a_1, \ldots, o_t, a_t)$ from the source corpus, we segment the sequence into overlapping execution tuples consisting of the user instruction, the previous visual observation, the executed action, and the resulting visual observation: $\tau_t = (I, o_t, a_t, o_{t+1})$, as shown in Figure 3 (Top-left). At timestep t, the MLLM annotator takes the previous macro-action history H_{t-1} and the tuple τ_t as inputs. The annotator translates the visual difference between the two consecutive observations into text to identify valid UI updates or silent failures. Merging this analysis with the previous macro-action history H_{t-1}, it recursively compresses past atomic steps into an updated macro-action history H_t containing multi-step achieved goals, which our critic leverages to guide the policy to have better history state understanding during its next-step planning.

Visually Grounded Error Analysis. To enable our critic to penalize errors from policy actions, we construct a dataset that captures diverse failure modes using visually grounded rationales. The dataset construction process consists of *three* steps: (i) state-transition extraction, (ii) plausible error synthesis, and (iii) multimodal rationale extraction.

Step 1. State-transition Extraction. To ground error analysis on real GUI state changes, we represent the immediate visual impact of each successful action in natural language (Chae et al., 2025a; Mei et al., 2025). Given the execution tuple τ_t, we prompt the MLLM annotator to generate a verbalized state-transition v_t that describes the observed causal effect of the action. Because the annotator conditions on the actual resulting screenshot o_{t+1}, v_t reflects a visual change rather than inferred intent (Figure 3 (Bottom-left)). These verbalized transitions provide a compact and interpretable representation of short-term dynamics, acting as the basis for rationale generation in **Step 3**.

Step 2. Plausible Error Synthesis. To synthesize a comprehensive set of common GUI failure modes reported in recent CUA literature (Li et al., 2025; Wang et al., 2025; Liu et al., 2026; Jin et al., 2026), we prompt the annotator to systematically perturb the *expert* action a_t into a *perturbed* action $\hat{a}_t$ (Figure 3 (Bottom-left)). This generation is guided by a taxonomy of 12 diverse error dimensions observed in CUA literature (the complete taxonomy, the distribution of each error type, and representative examples are in Appendix B), including *Grounding Errors* (e.g., intent is correct but clicking adjacent non-interactive space), *Proce-*

dural Prerequisite Neglect (e.g., typing without focusing the search field), and *Visual Hallucinations* (e.g., targeting a non-existent UI element). Conditioned on the user instruction I, past interaction trajectory $h_t = (a_1, a_2, \ldots, a_{t-1})$, current observation o_t, and expert action a_t, the annotator selects the most appropriate error from the taxonomy to ensure these synthesized errors are plausible. To prevent hallucinations and guarantee the quality of these synthesized actions, we employ an automated double-blind filtering pipeline using a multimodal judge (Qwen3-VL-32B). A human validation study confirms our filtering protocol highly correlates with human consensus (76.6% agreement); we detail this study in Appendix D.

Step 3. Multimodal Rationale Extraction. Using the expert action a_t and perturbed action $\hat{a}_t$, we prompt the annotator to generate fine-grained, step-by-step rationales, as shown in Figure 3 (Bottom-right). To enforce spatial awareness lacking in prior critics, we apply two strategies during data construction. First, to break the model's over-reliance on text, we mask the policy's verbal intent in 30% of samples. Second, we render a visual marker (Yang et al., 2023) on the screenshot o_t at the exact proposed coordinates and instruct the annotator to reason based on the marker, extracting the supervisory signal needed to teach the target critic to perform action verification using this marker position at test time. Conditioned on the provided action causal effect, which is either the ground-truth state-transition v_t for a_t or the pre-defined failure outcome for $\hat{a}_t$ (e.g., clicking inactive space yields no UI update in Ground Error), and this marked screenshot, the annotator structures a rigorous reasoning process. Specifically, it first visually verifies the action by identifying the UI element directly beneath the marker, preventing it from overly relying on the textual intent. Next, incorporating this visual verification and provided action causal effect, the annotator evaluates the action's alignment with the user instruction I, and outputs verbal feedback that can include a corresponding error dimension and a corrective explanation to guide a policy. This pipeline yields diverse SFT samples of visually grounded rationales (Figure 3 (Right-bottom)), providing the optimal data to teach our critic precise action verification.

2.3 Test-time CUA Guidance

During deployment, HiViG framework operates alongside the underlying CUA policy and HiViG-critic provides continual history tracking and error recovery. As illustrated in Figure 2, our framework intervenes in two distinct phases at each timestep t. First, before the policy plans its next move, the critic updates the macro-action history. By processing the previous execution tuple $\tau_{t-1} = (I, o_{t-1}, a_{t-1}, o_t)$ and prior history H_{t-1}, it produces an updated history H_t. By articulating visual changes and mapping them to global task progress, macro-action history equips the policy with the history state tracking required to propose an informed initial action a_t. Once proposed, the critic verifies this action. To ground the critic's analysis in visual observation, we render the visual marker at the action's proposed coordinates on the screenshot. Given the user instruction I, the execution trajectory h_t, and this marked observation, the critic evaluates the proposed action using its learned reasoning process: visual verification, state-transition prediction, and instruction alignment. If graded "Good", the action is directly executed, advancing the GUI to the next observation o_{t+1}. If the action is rated "Bad", the critic classifies the failure into one of the predefined errors and generates a verbal explanation. This pre-execution evaluation serves as an constraint, forcing the policy to refine its action, and enables continuous progress toward the goal, preventing the compounding of silent failures.

3 Experiments

3.1 Setup

Computer Use Agents. To execute complex GUI tasks and evaluate test-time intervention frameworks, we use two distinct Multimodal Large Language Models (MLLMs) as Computer Use Agent (CUA) policies: Qwen3-VL-32B-Thinking (Qwen Team, 2025), an open-source model; and Gemini-3-Flash (Gemini Team, 2025), a closed-source frontier model. We employ a generalized pixel-based action space adapted from Qwen Team (2025), where policies interact with environments by generating structured JSON tool calls encompassing standard operations (e.g., click, type, scroll, swipe). The complete specifications for the desktop and mobile action spaces can be found in Appendix C.

Dataset. To train our critic, we construct SFT dataset using the ScaleCUA (Liu et al., 2026) training corpus which contains GUI trajectories spanning web, mobile, and desktop environments. Using Qwen3-VL-32B-Thinking as the MLLM annotator, we generate 52k mixed samples, includ-

Method	*Qwen3-VL-32B-Thinking*				*Gemini-3-Flash*			
	WALv2	**ALab**	**WAA**	**Avg.**	**WALv2**	**ALab**	**WAA**	**Avg.**
Base agent	13.0	44.2	35.7	31.0	30.5	58.0	35.8	41.4
OpenCUA	14.9	47.1	35.2	32.4	29.9	57.2	35.8	41.0
SE-WSM	11.7	45.7	31.6	29.7	30.5	57.2	35.1	40.9
CGI (32B)	16.9	46.9	33.7	32.5	29.9	55.8	37.9	41.2
CGI (8B)	12.3	44.2	31.0	29.2	26.6	55.1	37.9	39.9
GUI-Critic-R1	13.6	49.3	34.4	32.4	22.1	53.6	32.5	36.1
HiVig (Ours)	**25.3**	**51.5**	**38.0**	**38.3**	**45.5**	**61.6**	**44.2**	**50.4**

Table 1: Comparison of test-time interventions on Qwen3-VL-32B and Gemini-3-Flash across GUI benchmarks in diverse environments. The best and the second best overall success rate results are in **bold** and underline, respectively. HiVig delivers gains in all benchmarks. WALv2: WebArenaLitev2 for web, ALab: AndroidLab for mobile, WAA: WindowsAgentArena for desktop, and Avg.: average overall success rate. Detailed performances for each benchmark are in Table 8, Table 9, and Table 10.

ing 20k history state tracking and 32k visually grounded error analysis.

For evaluation, we conduct experiments across three representative computing ecosystems: web, mobile, and desktop. For the web environment, we use WebArenaLitev2 (Liu et al., 2026; Zhou et al., 2024), comprising vision-centric tasks performed in 5 diverse websites with a 15-step maximum horizon. To assess mobile GUI control, we employ AndroidLab (Xu et al., 2025b), evaluating agents across 9 mobile applications with task-specific step limits. Finally, for the desktop environment, we use WindowsAgentArena (Bonatti et al., 2025), comprising Windows 11 OS tasks grouped into 6 categories (Rivard et al., 2025) bounded by a 30-step horizon. Across all benchmarks, performance is measured by execution-based task success rates. Further details are in Appendix A.

Baselines. We compare our approach against three distinct categories of test-time intervention: **(1) Base agent:** the unguided CUA, serving as a reference that executes the policy without any test-time intervention. **(2) Scalar feedback (PRMs):** To compare against standard Process Reward Models (PRMs) operating in a Best-of-N search, we adapt established step-evaluation templates (Lin et al., 2025) from OpenCUA (Wang et al., 2025) and SE-WSM (Sun et al., 2025). We simplify these into standard PRM templates using the same backbone model (Qwen3-VL-8B-Thinking) as our critic, and set $N = 2$ to match the computational cost of verbal frameworks, which allow a maximum of two policy calls per step. **(3) Verbal feedback:** To benchmark against existing critics, we evaluate two base models (Qwen3-VL-8B-Thinking and Qwen3-VL-32B-Thinking) as a critic prompted with the zero-shot multidimensional critique prompt from CGI (Yang et al., 2025). We also compare against GUI-Critic-R1 (Wanyan et al., 2025), a specialized critic model trained for mobile GUI environments. Further details regarding prompt configurations and baseline implementations are provided in Appendix A.

Implementation Details. We initialize our critic with Qwen3-VL-8B-Thinking (Qwen Team, 2025) base model, which is trained for one epoch on the mixed SFT dataset. Further details regarding training hyperparameters, computing resources, and inference configurations are provided in Appendix A.

3.2 Results and Discussion

HiVig outperforms existing test-time interventions across diverse GUI platforms and policies. To evaluate the efficacy and cross-platform generalizability of HiVig, we benchmark test-time interventions across three distinct GUI environments: web (WebArenaLitev2), mobile (AndroidLab), and desktop (WindowsAgentArena). As shown in Table 1, HiVig consistently improves both the open-weight Qwen3-VL-32B-Thinking and the frontier closed-source Gemini-3-Flash policies, achieving average absolute gains in overall success rate of 7.3% and 9.0% across the three domains, respectively. These improvements are observed in all three environments, demonstrating strong cross-platform generalization. This strong generalization stems from HiVig relying solely on raw GUI screenshots and pixel-level visual understanding, without requiring platform-specific features such as DOM or accessibility tree.

HiVig consistently achieves performance gains over the Qwen3-VL-32B-Thinking as the base

policy, outperforming strong baselines. Specifically, our framework surpasses the base agent by 12.3%, 7.3%, and 2.3% in absolute overall success rate on WebArenaLitev2, AndroidLab, and WindowsAgentArena, respectively, and surpasses the best baseline on each benchmark by 5.8% average overall success rate. We additionally evaluate HiVIG using the highly optimized, frontier-class Gemini-3-Flash as the policy backbone, to demonstrate the versatility of our test-time intervention approach. Although Gemini-3-Flash is already strong, HiVIG improves its absolute success rate by 15.0%, 3.6%, and 8.4% on WebArenaLitev2, AndroidLab, and WindowsAgentArena, respectively. These results suggest that limited history state tracking and ineffective error recovery remain key challenges for CUAs, even when powered by frontier models.

We observe that HiVIG is particularly beneficial for challenging tasks where existing test-time interventions often fail. Without an abstractive macro-action history, policies frequently repeat ineffective commands, as they lack a compressed record of prior attempts that would allow them to avoid short-sighted decision loops. Without visually grounded critiques to verify raw execution coordinates, critics often fail to detect misaligned actions caused by complex UI elements. In contrast, HiVIG mitigates these failures by maintaining a macro-action history and providing visually grounded critiques, allowing policies to make decisions anchored in visual content and historical progress. As a result, it improves performance on tasks that otherwise remain very difficult, increasing Qwen3-VL-32B-Thinking's success rate on the WebArenaLitev2 Map category from 3.9% to 23.1% (Table 8), and Gemini-3-Flash's success rate on the WindowsAgentArena Office category from 4.7% to 23.3% (Table 10).

Lack of visual grounding and history conditioning hurts other critics. In Table 1, compared to our approach, baseline interventions often degrade frontier model performance, yielding at best a 2.1% gain on WindowsAgentArena and at worst an 8.4% drop on WebArenaLitev2 for Gemini-3-Flash. Notably, scalar feedback baselines (OpenCUA and SE-WSM) yield no performance improvement over the base agent. This shows a fundamental limit of scalar rewards: when all candidate actions lead to silent failures, scalar scores offer no guidance to improvement. Furthermore, while verbal feedback approaches (CGI, GUI-Critic-R1) offer more expressive signals, their improvements are highly inconsistent across domains. For instance, GUI-Critic-R1 achieves a 5.1% absolute overall success rate gain on AndroidLab, but only a marginal 0.6% on WebArenaLitev2, and degrades performance by 1.3% on WindowsAgentArena. This instability stems from two fundamental limitations: First, existing critics lack the visual grounding (see a qualitative example in Figure 10) needed to generalize across the diverse, dense layouts of web and desktop interfaces. Second, they fail to track historical state, limiting the policy on tasks requiring long execution trajectories (examples in Figure 1 and Figure 11). Further analysis on the lack of visual grounding and history state tracking of baseline critics are in Section E.1 and Section E.2.

VisAnalysis	HisTrack	*Qwen3-VL*	*G-3-Flash*
✗	✗	13.5	30.0
✓	✗	21.4	35.1
✗	✓	23.4	42.9
✓	✓	**25.3**	**45.5**

Table 2: **Impact of verbal feedback components.** Experiments with components of HiVIG on WebArenaLitev2 evaluated with Qwen3-VL-32B and Gemini-3-Flash as CUA. VisAnalysis: Visually-grounded error analysis, HisAnalysis: History state tracking. Combining two components leads to better performance.

3.3 Analysis and Ablations

Effectiveness of Individual Components in HiVIG Framework. To understand the individual contributions of visually grounded error analysis and history state tracking in our framework, we ablate these components on WebArenaLitev2, as shown in Table 2. Deploying either component alone outperforms all baselines. This is especially notable with the highly-capable Gemini-3-Flash policy. While prior baselines degrade its performance on WebArenaLitev2 (Table 1), applying our critic solely with visually grounded error analysis increases its success rate from 30.5% to 35.1%. Furthermore, deploying history state tracking alone achieves 23.4% and 42.9% overall success rates for Qwen3-VL-32B and Gemini-3-Flash, respectively. By tracking the task progress, this component prevents the policy from falling into short-sighted decision loops and needlessly re-exploring known states. This independent strength also shows the computational flexibility of our framework; while

Method	Policy Inference	Critic Inference	Latency (s)	WALv2 Avg. (%)
Base agent	1	0	**8.2**	13.0
OpenCUA	2	2	31.8	14.9
SE-WSM	2	2	41.7	11.7
CGI (32B)	1-2	1	37.9	16.9
CGI (8B)	1-2	1	21.5	12.3
GUI-Critic-R1	1-2	1	14.8	13.6
VisAnalysis (Ours)	1-2	1	15.2	21.4
HisAnalysis (Ours)	1	1	11.5	23.4
HiVig (Ours)	1-2	2	17.4	**25.3**

Table 3: **Performance-latency trade-off.** Average step latency (wall-clock time) and the average success rate on WebArenaLitev2. Policy Inference and Critic Inference indicate the number of policy and critic model inferences per step, respectively, at test-time. Latency is measured across 20 tasks using 4 H100 GPUs. HiVig demonstrates superior performance while maintaining a highly competitive runtime. VisAnalysis: Visually-grounded error analysis, HisAnalysis: History state tracking. Combining two components leads to better performance.

Intent Masking	Visual Marker	WALv2	ALab
✓	✗	20.8	46.4
✗	✓	**25.3**	47.1
✓	✓	**25.3**	**51.5**

Table 4: **Ablation of visual grounding strategies.** We evaluate the impact of masking the policy's verbal intent (30% of SFT samples) and injecting a visual marker (i.e., 'X' marker). Combining both strategies forces the critic to break its text reliance and evaluate the raw spatial coordinates, yielding visually accurate verbal feedback. WALv2: WebArenaLitev2, ALab: AndroidLab.

the full pipeline requires two critic inferences per step, a single inference of either capability is already sufficient to achieve better results. Combining these two within HiVig yields the highest overall success rates (25.3% for Qwen3-VL-32B-Thinking and 45.5% for Gemini-3-Flash), proving a strong synergistic effect of the two components that are essential to HiVig's superiority. More detailed analyses of these components are provided in Section E.3 and Section E.4.

Importance of Visual Grounding. Prior critics often struggle to enforce visual grounding effectively in their training mechanisms and erroneously approve spatially misaligned actions by over-relying on the policy's verbal intent. To overcome this, we enforce visual grounding during Multimodal Rationale Extraction (**Step 3**) in Section 2.2: masking the verbal intent and injecting a visual marker on the screenshot. Table 4 demonstrates the efficacy of combining both strategies. Training without intent masking degrades AndroidLab performance from 51.5% to 47.1%. This exposes a "shortcut" phenomenon; if verbal intent is consistently available during training, the model ignores the visual observation, missing critical spatial information. When trained without the visual marker, performance drops significantly (e.g., from 25.3% to 20.8% on WebArenaLitev2). This indicates that MLLMs struggle to interpret raw numerical coordinates in complex visual interfaces, and the marker can act as a reliable and effective spatial anchor to help comprehend the visual semantics of the action. Overall, having both strategies shows the best performance, where the intent masking breaks text-reliance to force analysis on the screenshot, while the visual marker provides spatial anchors needed to verify action execution. Additionally, we provide the analysis of the importance of state-transition knowledge in Section E.5.

Performance and Latency Trade-off. To better understand the computational efficiency of HiVig test-time intervention, we evaluate explicit wall-clock time against baseline methods. We measure the average step latency across 20 tasks in WebArenaLitev2 using 4 H100 GPUs, allocating 2 GPUs for the Qwen3-VL-32B policy and 2 for the critic models. As detailed in Table 3, HiVig allows for flexible trade-offs between accuracy and efficiency. Employing either the visually grounded error analysis or the history state tracking component alone requires only a single critic inference, achieving average latencies of 15.2s and 11.5s, respectively. This shows that HiVig's individual components are as computationally efficient as GUI-Critic-R1 (14.8s) while significantly outperforming it (21.4% and 23.4% vs. 13.6%). Furthermore, deploying the full HiVig pipeline with two critic inferences

requires 17.4s. This remains substantially faster than 32B-scale critics like CGI (37.9s) while delivering far superior performance (25.3% vs. 16.9%). Overall, HIVIG introduces a latency increase over the base agent (8.2s), but achieves the highest performance among all evaluated methods in Table 3.

4 Related work

Reward Models for Scalar Feedback. To mitigate compounding errors in long-horizon GUI tasks, recent test-time interventions employ reward models to score candidate actions (Xu et al., 2025a; Cheng et al., 2025; Mei et al., 2025; Chen et al., 2025). World-model-style reward models (Chae et al., 2025a; Bai et al., 2025) predict action outcomes and assign scalar scores to future states. However, reliably generating precise future states in complex GUIs is challenging (Cheng et al., 2025; Zheng et al., 2026). Moreover, evaluating these imperfect simulations with outcome-based scalar rewards risks compounding errors and provides no path to improvement when all candidate actions are poor. While some work augments reward models with external tutorials (Xu et al., 2025a; Mei et al., 2025) to improve planning, this reliance can limit generalizability, as such resources are rarely available for proprietary or diverse GUI environments. In contrast, HIVIG utilizes a computationally efficient 8B model and generalizes across diverse visual interfaces by relying purely on screenshot-level visual understanding of the current state.

Verbal Feedback Critics. In general, scalar feedback offers little guidance to the policy when all candidate actions are poor. To address this, verbal critics (Xiong et al., 2026) provide natural-language critiques (Luo et al., 2025; Zhong et al., 2024) that help policies refine their trajectories (Wu et al., 2025a; Tang et al., 2025; Yang et al., 2025). This refinement process involves executing an action, observing the environment outcome, and backtracking to correct mistakes. However, generating critiques after an action is executed is highly impractical in real-world GUIs, as many actions are irreversible (e.g., sending an email or deleting a file). GUI-Critic-R1 (Wanyan et al., 2025) addresses this by extending critique generation to the pre-execution setting, but lacks the visual grounding to verify spatial correctness and does not explicitly reason over historical interactions. Similarly, OS-Oracle (Wu et al., 2025b) proposes a framework for cross-platform GUI critics using taxonomy-guided error synthesis. However, OS-Oracle relies primarily on prompting frontier models (e.g., GPT-4o) for rationale generation, lacking strict mechanisms to enforce pixel-level spatial verification or explicit history state tracking. In contrast, HIVIG provides history-aware, visually grounded critiques by predicting the visual consequence of a proposed action from the current visual observation and grounding its assessment in past trajectories, enabling safer and more reliable test-time decision making.

5 Conclusion

In this paper, we introduce HIVIG, a test-time intervention framework that equips CUAs with history state tracking and visually grounded error analysis. HIVIG leverages a multimodal critic trained to summarize past interactions that enables policies to track long-term progress and a visually grounded critique that intercepts spatial errors before execution by verifying raw execution coordinates against the screenshot. Evaluations show HIVIG generalizes across web, mobile, and desktop environments, consistently enhancing frontier models and achieving average overall success rates by up to 9.0%. Our ablations confirm a strong synergy between the two components, proving that history-aware and visually grounded test-time intervention can push the performance limits of frontier policies without modifying their underlying weights.

Limitations

While our test-time intervention framework shows strong performance across diverse policies and GUI environments, our taxonomy remains bounded by the current landscape of digital interfaces. Although our 12 predefined dimensions capture the most prevalent GUI failure modes, the continuous evolution of digital interfaces and policy capabilities will likely give rise to new classes of spatial and reasoning errors, necessitating an iterative expansion of the existing taxonomy.

Acknowledgments

This work was supported by NSF-AI Engage Institute DRL2112635, NSF-CAREER Award 1846185, ARO Award W911NF2110220, ONR Grant N00014-23-1-2356, Capital One Research Award, Apple PhD Fellowship, NDSEG PhD Fellowship. The views contained in this article are those of the authors and not of the funding agency.

References

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

Hao Bai, Yifei Zhou, Li Erran Li, Sergey Levine, and Aviral Kumar. 2025. Digi-q: Learning VLM q-value functions for training device-control agents. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Keunho Jang, and Zheng Hui. 2025. Windows agent arena: Evaluating multi-modal OS agents at scale. In *Proceedings of the International Conference on Machine Learning (ICML)*.

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 *Proceedings of the International Conference on Learning Representations (ICLR)*.

Hyungjoo Chae, Sunghwan Kim, Junhee Cho, Seungone Kim, Seungjun Moon, Gyeom Hwangbo, Dongha Lim, Minjin Kim, Yeonjun Hwang, Minju Gwak, Dongwook Choi, Minseok Kang, Gwanhoon Im, ByeongUng Cho, Hyojun Kim, Jun Hee Han, Taeyoon Kwon, Minju Kim, Beong-woo Kwak, Dongjin Kang, and Jinyoung Yeo. 2025b. Webshepherd: Advancing prms for reinforcing web agents. *arXiv preprint arXiv:2505.15277*.

Cong Chen, Kaixiang Ji, Hao Zhong, Muzhi Zhu, Anzhou Li, Guo Gan, Ziyuan Huang, Cheng Zou, Jiajia Liu, Jingdong Chen, Hao Chen, and Chunhua Shen. 2025. Gui-shepherd: Reliable process reward and verification for long-sequence GUI tasks. *arXiv preprint arXiv:2509.23738*.

Jiali Cheng, Anjishnu Kumar, Roshan Lal, Rishi Rajasekaran, Hani Ramezani, Omar Zia Khan, Oleg Rokhlenko, Sunny Chiu-Webster, Gang Hua, and Hadi Amiri. 2025. Webatlas: An llm agent with experience-driven memory and action simulation. *Preprint*, arXiv:2510.22732.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samual Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Gemini Team. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*. Proceedings of the Association for Computational Linguistics (ACL).

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogagent: A visual language model for GUI agents. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*.

Chao Jin, Wenkui Yang, Hao Sun, Yuqi Liao, Qianyi Jiang, Kai Zhou, Jie Cao, Ran He, and Huaibo Huang. 2026. Halluclear: Diagnosing, evaluating and mitigating hallucinations in gui agents. *arXiv preprint arXiv:2604.17284*.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the Association for Computational Linguistics (ACL)*.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025. Screenspot-pro: GUI grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia, MM 2025, Dublin, Ireland, October 27-31, 2025*.

Haojia Lin, Xiaoyu Tan, Yulei Qin, Zihan Xu, Yuchen Shi, Zongyi Li, Gang Li, Shaofei Cai, Siqi Cai, Chaoyou Fu, Ke Li, and Xing Sun. 2025. Cuarewardbench: A benchmark for evaluating reward models on computer-using agent. *arXiv preprint arXiv:2510.18596*.

Zhaoyang Liu, JingJing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, Shenglong Ye, Qingyun Li, Xuan Dong, Yue Yu, Chenyu Lu, YunXiang Mo, Yao Yan, Zeyue Tian, Xiao Zhang, Yuan Huang, Yiqian Liu, Weijie Su, Gen Luo, Xiangyu Yue, Biqing Qi, Kai Chen, Bowen Zhou, Yu Qiao, Qifeng Chen, and Wenhai Wang. 2026. Scalecua: Scaling open-source computer use agents with cross-platform data. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Renjie Luo, Zichen Liu, Xiangyan Liu, Chao Du, Min Lin, Wenhu Chen, Wei Lu, and Tianyu Pang. 2025. Language models can learn from verbal feedback without scalar rewards. *arXiv preprint arXiv:2509.22638*.

Kai Mei, Jiang Guo, Shuaichen Chang, Mingwen Dong, Dongkyu Lee, Xing Niu, and Jiarong

Jiang. 2025. R-wom: Retrieval-augmented world model for computer-use agents. *arXiv preprint arXiv:2510.11892*.

Yuting Ning, Jaylen Jones, Zhehao Zhang, Chentao Ye, Weitong Ruan, Junyi Li, Rahul Gupta, and Huan Sun. 2026. When actions go off-task: Detecting and correcting misaligned actions in computer-use agents. *arXiv preprint arXiv:2602.08995*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. 2025. UI-TARS: pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Qwen Team. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E. Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P. Lillicrap, and Oriana Riva. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Luke Rivard, Sun Sun, Hongyu Guo, Wenhu Chen, and Yuntian Deng. 2025. Neuralos: Towards simulating operating systems via neural generative models. *arXiv preprint arXiv:2507.08800*.

Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. 2025. Seagent: Self-evolving computer use agent with autonomous learning from experience. *arXiv preprint arXiv:2508.04700*.

Qiaoyu Tang, Hao Xiang, Le Yu, Bowen Yu, Hongyu Lin, Yaojie Lu, Xianpei Han, Le Sun, and Junyang Lin. 2025. Refcritic: Training long chain-of-thought critic models with refinement feedback. *arXiv preprint arXiv:2507.15024*.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Dikang Du, Hao Hu, Huarong Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Y. Charles, Zhilin Yang, and Tao Yu. 2025. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*.

Yuyang Wanyan, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Jiabo Ye, Yutong Kou, Ming-Hsuan Yang, Fei Huang, Xiaoshan Yang, Weiming Dong, and Changsheng Xu. 2025. Look before you leap: A gui-critic-r1 model for pre-operative error diagnosis in GUI automation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Penghao Wu, Shengnan Ma, Bo Wang, Jiaheng Yu, Lewei Lu, and Ziwei Liu. 2025a. Gui-reflection: Empowering multimodal GUI models with self-reflection behavior. *arXiv preprint arXiv:2506.08012*.

Zhenyu Wu, Jingjing Xie, Zehao Li, Bowen Yang, Qiushi Sun, Zhaoyang Liu, Zhoumianze Liu, Yu Qiao, Xiangyu Yue, Zun Wang, and Zichen Ding. 2025b. Os-oracle: A comprehensive framework for cross-platform GUI critic models. *arXiv preprint arXiv:2512.16295*.

Tao Xiong, Xavier Hu, Yurun Chen, Yuhang Liu, Changqiao Wu, Pengzhi Gao, Wei Liu, Jian Luan, and Shengyu Zhang. 2025. GUI-PRA: process reward agent for GUI tasks. *arXiv preprint arXiv:2509.23263*.

Tianyi Xiong, Shihao Wang, Guilin Liu, Yi Dong, Ming Li, Heng Huang, Jan Kautz, and Zhiding Yu. 2026. Phycritic: Multimodal critic models for physical AI. *arXiv preprint arXiv:2602.11124*.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, Zhiyuan Chen, Jitong Liao, Qi Zheng, Jiahui Zeng, Ze Xu, Shuai Bai, Junyang Lin, Jingren Zhou, and Ming Yan. 2026. Mobile-agent-v3.5: Multi-platform fundamental GUI agents. *arXiv preprint arXiv:2602.16855*.

Ran Xu, Kaixin Ma, Wenhao Yu, Hongming Zhang, Joyce C. Ho, Carl Yang, and Dong Yu. 2025a. Retrieval-augmented GUI agents with generative guidelines. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2025b. Androidlab: Training and systematic benchmarking of android autonomous agents. In *Proceedings of the Association for Computational Linguistics (ACL)*.

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. *arXiv preprint arXiv:2601.07779*.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Ruihan Yang, Fanghua Ye, Jian Li, Siyu Yuan, Yikai Zhang, Zhaopeng Tu, Xiaolong Li, and Deqing Yang. 2025. The lighthouse of language: Enhancing LLM agents via critique-guided improvement. *arXiv preprint arXiv:2511.21631*.

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

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the Association for Computational Linguistics (ACL)*.

Yuhao Zheng, Li'an Zhong, Yi Wang, Rui Dai, Kaikui Liu, Xiangxiang Chu, Linyuan Lv, Philip Torr, and Kevin Qinghong Lin. 2026. Code2world: A GUI world model via renderable code generation. *arXiv preprint arXiv:2602.09856*.

Victor Zhong, Dipendra Misra, Xingdi Yuan, and Marc-Alexandre Côté. 2024. Policy improvement using language feedback models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

A Details of Experimental Setups

Training Dataset. To construct the SFT datasets for our critic, we use the ScaleCUA (Liu et al., 2026) training corpus as source data that contains multi-domain GUI trajectories, including web, mobile, and desktop. We employ Qwen3-VL-32B-Thinking as the MLLM annotator to extract the multimodal rationale for error analysis and generate the macro-action history. The resulting training data contains 52k mixed SFT samples: 20k samples for the history state tracking task, and 32k samples for the visually grounded error analysis task, containing 16k samples for expert actions and 16k samples for perturbed actions.

Evaluation Benchmarks. We provide in-depth explanations of the cross-platform GUI navigation benchmarks used in our experiments, following the evaluation setups in Liu et al. (2026). For the web environment, WebArenaLitev2 (Liu et al., 2026) serves as a vision-centric adaptation of the WebArena (Zhou et al., 2024) framework, consisting of 154 tasks that span diverse interactive websites, including online shopping, content management platforms, map services, GitLab, and Reddit. To assess mobile GUI control, AndroidLab (Xu et al., 2025b) utilizes a pixel-grounded visual interaction mode to test true spatial understanding, evaluating agents across 138 complex tasks distributed over 9 native applications such as the calendar, clock, contacts, map, and settings. Finally, for the desktop environment, WindowsAgentArena (Bonatti et al., 2025) tests agents within a fully functional Windows 11 virtual machine across 145 diverse tasks. We chose a 30-step execution horizon as an intermediate setting between the shorter 15-step horizons and the 50-step limits in the previous paper (Liu et al., 2026; Yang et al., 2026), which allows us to evaluate the policy's ability to solve long-horizon tasks without high costs. To thoroughly evaluate performance across varying operational complexities, we follow the functional categorization in Yang et al. (2026), grouping these tasks into six specific Windows OS application scenarios: Office, Web Browsing, Windows System, Code, Media, and Windows Utilities. ScaleCUA and WebArenaLiteV2 are released under the Apache 2.0 License, and AndroidLab and WindowsAgentArena are released under the MIT License.

Baselines. We provide a more detailed explanation of the baseline models and their specific prompt configurations used in our experiments.

- **OpenCUA** (Wang et al., 2025) originally employs a strong proprietary model (e.g., Claude) as a step reflector in their chain-of-thought data annotation pipeline. It generates reflections based on previous step reasoning and current screenshots. To avoid coupling complexities and ensure a fair comparative evaluation, we simplify their reflector prompt into a standard Process Reward Model (PRM) template. Given the prompt, Qwen3-VL-8B-Thinking base model outputs a scalar quality score.
- **SE-WSM** (Sun et al., 2025) conducts a comprehensive, step-by-step analysis of input trajectories, providing multidimensional evaluations that cover trajectory correctness, the identification of redundant steps, the first error step, and correct action suggestions. We adapt SE-WSM's dense prompt template for our scalar feedback baseline to evaluate candidate actions in a Best-of-N search, also utilizing Qwen3-VL-8B-Thinking as the base evaluator.
- **CGI** (Yang et al., 2025) is a zero-shot critique generation framework. We implement this by prompting our base models (Qwen3-VL-8B-Thinking and Qwen3-VL-32B-Thinking) to systematically analyze the proposed action across three explicit dimensions: *Contribution* (task progress), *Feasibility* (validity within the action space), and *Efficiency* (optimality without redundancy). This structured analysis is then provided as verbal feedback to contextually guide the agent's error recovery.
- **GUI-Critic-R1** (Wanyan et al., 2025) is a specialized, open-source critic model explicitly trained to generate fine-grained verbal feedback for mobile GUI environments. We utilize it as a supervised baseline to compare our zero-shot and test-time scaling approaches against a model that has been directly optimized for trajectory critique and error identification.

Implementation Details. We implement our training framework using the `LlamaFactory` (Zheng et al., 2024) recipes on 8 NVIDIA H100 GPUs (80GB). The model is optimized for 1 epoch using a cosine learning rate schedule with a peak learning rate of 5e-6 and a global batch size of 256, taking approximately 4 hours to train. During test-time deployment, we

evaluate the benchmarks using 8 NVIDIA H100 GPUs (80GB). We set the history window sizes to $W = 4$ and $W = 2$ for Qwen3-VL-32B-Thinking and Gemini-3-Flash, respectively, following their official computer-use frameworks (Qwen Team, 2025; Gemini Team, 2025). All experimental results are from a single run, due to heavy compute costs for long-horizon GUI tasks.

B Complete Taxonomy of Synthesized Error Dimensions

To ensure our critic learns to detect and recover from a comprehensive set of failure modes, we synthesize plausible errors that mirror the actual mistakes made by standard Computer Use Agents (CUAs). Rather than relying on arbitrary perturbations, our taxonomy is empirically based in the primary failure modes observed across recent agent evaluations and benchmarks. We categorize our 12 error dimensions into two core vulnerabilities of CUAs, directly addressing the limitations outlined in our methodology: Visual and Spatial Grounding and Cognitive Execution. The distribution of these error dimensions within our constructed training data is illustrated in Figure 4. Additionally, comprehensive examples for each error dimension are provided across Figures 12 to 17.

Visual and Spatial Grounding Errors These errors occur when the policy's logical reasoning is disconnected from the actual visual reality of the current GUI state, a persistent bottleneck in MLLMs interacting with raw screenshots.

- **Grounding/Spatial Error:** The semantic intent is correct, but coordinate precision fails. The agent near-misses the target, landing in adjacent dead space (e.g., a minor pixel offset just outside the bounding box). This is a well-documented failure mode in unconstrained coordinate environments (Li et al., 2025; Wang et al., 2025).
- **Visual Hallucination:** The agent interacts with a non-existent UI element, such as targeting a ghost element from a previous state or guessing a layout coordinate based on parametric priors rather than visual evidence (Jin et al., 2026).
- **Observation Neglect:** The agent attempts to search, scroll, or open menus for target elements that are already clearly visible on the current screen (Jin et al., 2026).
- **Parameter Vector Miscalibration:** The agent fails physical vector execution (Rawles et al., 2025). It either reasons the exact opposite direction of the goal (Polarity Reversal, e.g., the wrong mathematical sign for scrolling) or uses an insufficient scale, resulting in negligible UI movement (Magnitude Insufficiency).
- **Action-Operation Misalignment:** The verbalized intent contradicts the executable JSON string. The reasoned action description is logical, but the actual generated action execution is invalid (Liu et al., 2026).

Cognitive and Step-Level Execution Errors These errors occur when the agent successfully perceives the screen but fails to reason about the correct immediate action, format, or sequence.

- **Termination Misjudgment:** The agent misjudges task completion. It either prematurely outputs a `terminate` command, fails to explicitly report required data using the `answer` tool, or hallucinates redundant steps long after the goal has been met (Wang et al., 2025).
- **Procedural Prerequisite Neglect:** The agent skips a mandatory preceding state change, such as failing to focus a field before typing or failing to dismiss a foreground overlay blocking the target (Zhou et al., 2024; Wang et al., 2025).
- **Semantic Error:** The agent targets perfectly but misinterprets vocabulary, icons, or UI paradigms (e.g., clicking "Sign Up" instead of "Log In", or clicking a deceptive ad) (Deng et al., 2023)
- **Constraint Neglect:** The agent ignores a specific attribute or positional constraint explicitly stated in the goal (e.g., selecting the wrong author or wrong position) (Liu et al., 2026).
- **Action Formulation Error:** The intent is correct, but the generated JSON crashes the parser due to syntax errors (e.g., missing quotes, trailing commas) or missing required arguments/invalid enums (Yang et al., 2024).
- **Suboptimal Path:** The agent selects highly inefficient micro-actions (e.g., repetitive arrow or backspace clicks) instead of standard, faster paradigms (e.g., direct text entry, bulk delete) (Zhou et al., 2024).
- **Timing and Latency Neglect:** The agent executes an action prematurely, ignoring system busy indicators like loading spinners, unfolding

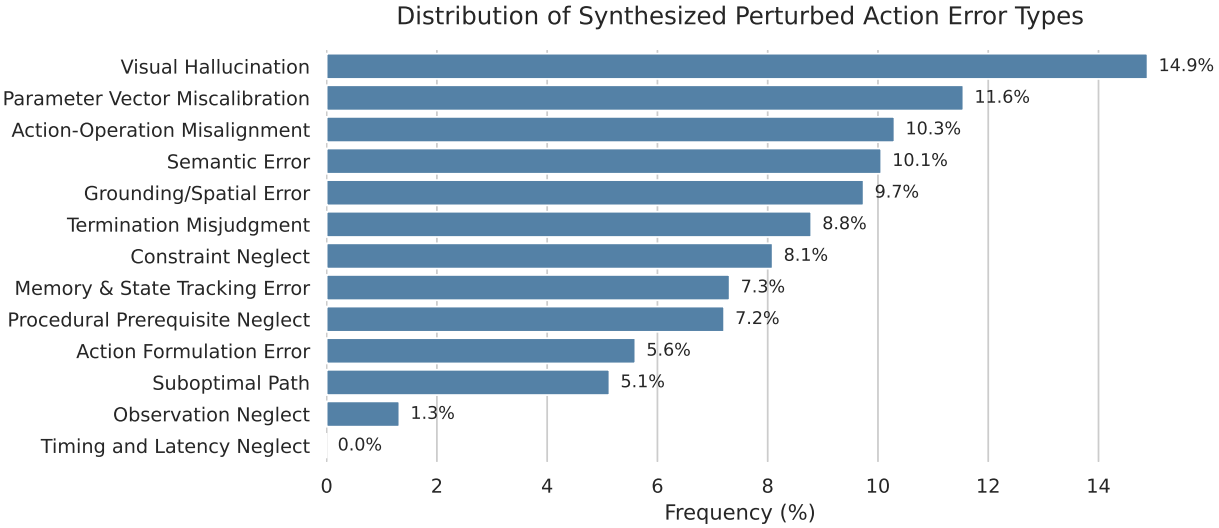


Figure 4: **Distribution of synthesized error types.** We show the distribution of error types within the training dataset for the visually grounded error analysis task.

menus, or disabled buttons.

C Computer Use Agent Action Space

To ensure reproducibility, we provide the exact JSON schemas that detail the action spaces, which are adapted from Qwen3-VL (Qwen Team, 2025). for both desktop and mobile environments, as shown in Figure 8 and Figure 9. Across all platforms, the visual observation is normalized to a standardized 1000×1000 coordinate grid. By establishing this unified continuous pixel space and action grammar, we ensure that both the underlying policies and our test-time critic can generalize their interactions and visually grounded evaluations across diverse GUI environments, independent of the host operating system.

D Human Validation of Automated Data Filtering

In Section 2.2, we used the automated data filtering pipeline with the multimodal judge (Qwen3-VL-32B) to ensure the quality of the synthesized perturbed actions. To validate the reliability of our automated double-blind filtering pipeline, we conducted a human evaluation study to ensure the multimodal judge (Qwen3-VL-32B) correctly identifies defects without arbitrary rejections. We randomly sampled 50 training instances from our data source. For each instance, three authors were provided with the visual observation and user instruction, and tasked with identifying the correct expert action between the ground truth and the synthesized perturbed action (with labels blinded). Using majority vote aggregation, the human consensus achieved a 72.3% agreement rate with the raw pipeline labels and a 76.6% agreement rate with the decision made by our automated judge. This strong correlation demonstrates that our automated filtering aligns closely with human judgment, confirming it as a reliable, scalable mechanism for removing invalid synthesized errors.

E Additional Experiments

E.1 Lack of Visual Grounding in Previous Critics

In Figure 10, we provide an example where ungrounded GUI-Critic-R1 (Wanyan et al., 2025) failed to capture the spatial error while HiVIG-critic captures the error. In this scenario, the policy proposes an action with a logically correct textual intent ("Click on the Sales menu") to fulfill the user's instruction, but outputs incorrect spatial coordinates that actually target the adjacent "Catalog" icon. As GUI-Critic-R1 lacks explicit spatial verification, it over-relies on the agent's verbalized intent. It assumes the text aligns with the spatial execution, leading it to hallucinate a successful outcome and erroneously approve the flawed action. In contrast, HiVIG-critic uses a visual marker on the exact execution coordinates, which helps the critic to verify the actual UI element being targeted. This allows it to immediately detect the contradiction between the stated intent and the visual reality, correctly detecting an "Action-Operation Misalignment" error and providing verbal feedback to the policy to prevent a costly misnavigation.

VisAnalysis	HisTrack	Overall
Separate HiViG	Separate HiViG	23.4
Separate HiViG	Qwen3-VL-8B	24.7
Separate HiViG	Qwen3-VL-32B	**25.3**
Mixed training HiViG (Ours)		**25.3**

Table 5: **Ablation on critic training.** We compare the overall success rate of a unified 8B critic trained on a mixed dataset (Mixed training HiViG) against separately trained 8B critics (Separate HiViG) and zero-shot baselines that use Qwen3-VL models for historical progress grounding across WebArenaLitev2. VisAnalysis: Visually grounded error analysis, HisTrack: Historical state tracking. Both Qwen3-VL-8B and Qwen3-VL-32B are thinking models.

E.2 Lack of History State Tracking in Previous Critics

In Figure 11, we illustrate a failure case where the baseline critic, GUI-Critic-R1 (Wanyan et al., 2025), fails to account for task history, whereas HiViG-critic successfully leverages its macro-action history to rectify the agent's behavior. In this scenario, the agent intends to open a direction planner but performs an incorrect click that fails to trigger the navigation interface. Because the baseline critic lacks a mechanism to track history states, it erroneously labels the action as "Correct" and fails to provide any corrective guidance. This lack of state awareness leaves the policy forget its previous mistake, leading it to repeat the same ineffective click. In contrast, our critic maintains a macro-action history that logs history states across multiple interactions by the policy. Providing this marco-action history, our critic allows the policy to recogize the past failure, allowing the policy to make more history-aware decisions to avoid making the same mistake. This demonstrates that history tracking is essential to guide the policy in long-horizon GUI tasks.

E.3 Impact of Mixed Dataset Training and Training Efficiency

We ablate the training data composition by comparing a single critic trained on the mixed dataset against maintaining separate models for each capability, isolating the contribution of each component. As shown in Table 5, training two separate 8B critics (one exclusively for visually grounded error analysis dataset and another for history state tracking dataset) yields a suboptimal overall success rate of 23.4% on WebArenaLitev2. In contrast, jointly training a single model on the mixed dataset increases performance to 25.3%. This improvement suggests that visually grounded error analysis and history state tracking are synergic tasks, as they both require pixel-level understanding of GUI screenshots and state-transitions. Thus, co-training might provide complementary supervisory signals that enhance the model's overall visual reasoning. Furthermore, when substituting our critic with the zero-shot history state tracking capabilities of the base Qwen3-VL-8B-Thinking and Qwen3-VL-32B-Thinking models, where the latter is the annotator model, we observe that the 32B model achieves 25.3% overall success rate. Our mixed-trained 8B model matches this performance, confirming that our mixed SFT approach successfully distills the history state tracking of the 32B annotator into an efficient, unified 8B critic, minimizing inference overhead without sacrificing accuracy.

Finally, we analyze the impact of training duration. We compare HiViG-critic trained for a single epoch, which is our default configuration, against a model trained for two epochs with our mixed dataset. Performance remains highly competitive between the two. The default setting achieves 25.3% and 51.5% on WebArenaLitev2 and AndroidLab, respectively, while the two-epoch model reaches 24.7% and 52.2%. This consistency across training durations demonstrates that our framework achieves robust performance with minimal training overhead.

E.4 Synergistic Effects of HiViG's Two Core Components

To further understand how visually grounded error analysis and history state tracking components combine in HiViG framework, we analyze the task-level success of the base agent, visually grounded error analysis, history state tracking, and HiViG on WebArenaLitev2 when using Qwen3-VL-32B-Thinking as an agent. As illustrated in Figure 5, each component independently solves a unique subset of tasks that the base agent cannot, demonstrating that they address orthogonal failure modes. These independent capabilities are largely inherited by the combined HiViG framework, which successfully retains the unique fixes from both individual components. This synergistic effect is also evident in the Gemini-3-Flash experiments on WebArenaLitev2. While visually grounded error analysis alone yields a comparably smaller performance increase (from 30.5% to 35.1%) com-

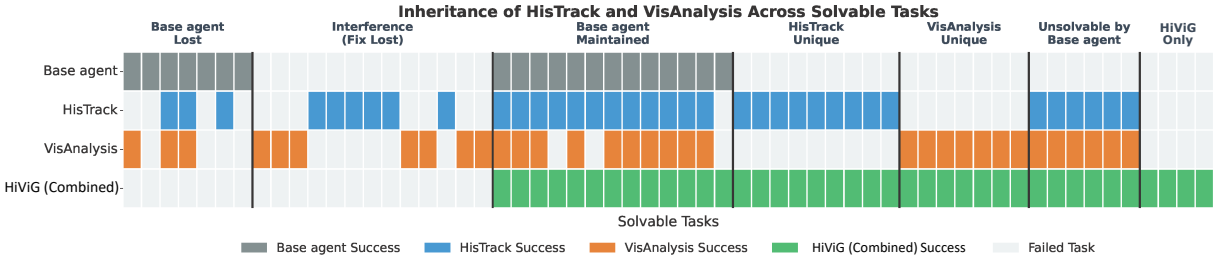


Figure 5: **Task-level success overlap of HiViG components.** We visualize the success patterns of the base agent, HisTrack (history state tracking) alone, VisAnalysis (visually grounded error analysis) alone, and the combined HiViG framework across all tasks in WebArenaLitev2 solvable by at least one configuration. We use Qwen3-VL-32B-Thinking as the CUA. The presence of distinct tasks uniquely solvable by HisTrack (9 tasks) or VisAnalysis (7 tasks) demonstrates that the two components resolve orthogonal failure modes. The combined framework successfully inherits these independent capabilities—alongside 6 tasks fixed by both—retaining the vast majority of individual fixes. Furthermore, the combined approach exhibits emergent synergy by uniquely solving 4 tasks that all other configurations failed, achieving the highest overall success rate.

pared to the gains from history state tracking alone (from 30.5% to 42.9%), integrating both components achieves the highest overall success rate of 45.5%. This shows that even when history state tracking acts as the primary driver of performance gains in long-horizon tasks, visually grounded error analysis reliably resolves distinct, complementary errors, making their combination essential for maximizing the agent's overall capabilities.

E.5 Impact of State-transitions

In our data construction pipeline, multimodal rationale grounds the action's consequence in GUI environments by providing the annotator with the ground-truth verbalized state-transition (v_t) derived from the actual execution tuple (o_t, a_t, o_{t+1}). To validate the impact of grounding multimodal rationale on the state-transition, we ablate this component by denying the annotator access to v_t, instead letting the annotator predict the state-transition based on its internal parametric knowledge like GUI-Critic-R1 (Wanyan et al., 2025). Training our critic on this ablation dataset causes a performance drop: the overall success rate falls from 25.3% to 20.8% on WebArenaLitev2. This degradation highlights the difficulty of acting as a reliable world model in complex GUI environments. Without grounding in actual visual outcomes, the annotator frequently hallucinates incorrect state-transitions, distilling these inaccuracies into the trained critic. This demonstrates that extracting and learning from verified, ground-truth state-transitions is important to bypass the hallucination issues that can limit prior baseline critics.

Intent Masking Ratio	WALv2	ALab	Avg.
0%	25.3	47.1	36.2
30%	25.3	**51.5**	**38.4**
50%	**26.0**	49.3	37.7

Table 6: **Ablation on intent masking.** We evaluate the impact of the intent masking ratio for visual grounding. While a 50% mask yields slight improvements on WebArenaLitev2, the 30% ratio provides the most robust and balanced performance across platforms. WALv2: WebArenaLitev2, ALab: AndroidLab. Avg.: average overall success rate.

E.6 Impact of Intent Masking

To encourage visual grounding, we incorporate intent masking during data construction. We evaluate the sensitivity of our framework to this hyperparameter by comparing our default 30% masking ratio against 0% (no masking) and 50% masking. As shown in Table 6, while increasing the masking ratio to 50% yields marginal gains on WebArenaLitev2 (26.0%), it leads to a performance drop on AndroidLab (49.3%). Conversely, the absence of intent masking (0%) results in lower performance across both benchmarks compared to our default setting. These results demonstrate that while intent masking is essential for enhancing the critic's visual grounding capabilities, our 30% masking ratio provides the most balanced and competitive performance across diverse GUI environments. The performance decline at higher masking ratios (50%) potentially be due to a distributional shift between training and test-time inputs. While we employ intent masking during training to force visual grounding, the critic receives unmasked intent actions at test time. Excessive masking during training increases the discrepancy between the training distri-

bution and the test-time objective, causing the critic to struggle to fully understand the given actions.

E.7 Evaluation on OS-Critic-Bench

We evaluate HiViG-critic on OS-Critic-Bench (Wu et al., 2025b), which is a benchmark designed to measure how well GUI critics catch erroneous or suboptimal actions proposed by policies. As shown in Table 11, while OS-Oracle achieves higher overall performance, it is important to note that OS-Critic-Bench is constructed from the held-out test splits of the data sources used to train OS-Oracle, constituting an in-domain evaluation. In contrast, HiViG-critic is evaluated on OS-Critic-Bench in a strictly zero-shot, out-of-domain setting. Despite this distribution shift, HiViG-critic demonstrates highly competitive performance, notably outperforming OS-Oracle on the web split (65.6% vs. 62.2% accuracy).

Furthermore, we analyze the intervention behavior of HiViG-critic using OS-Critic-Bench. Because the long-horizon GUI benchmarks evaluated in our main experiments lack step-by-step ground-truth actions, OS-Critic-Bench, which features human-annotated validity labels for proposed actions, serves as a good proxy for human-aligned validation. On this benchmark, our critic achieves a 71.4% overall F1 score, a 1.9% false rejection rate, and an 89.7% negative predictive value. This demonstrates that HiViG-critic acts as a conservative, high-precision filter. The extremely low false rejection rate ensures that the policy is rarely derailed from a correct trajectory, while the high negative predictive value guarantees that when our critic does intervene, its corrections are highly reliable. This conservative behavior reflects our strict intervention threshold in zero-shot settings, prioritizing non-disruptive support to prevent the critic from injecting erroneous feedback into the policy's execution.

E.8 Robustness Analysis

We also conducted multi-run experiments ($n = 3$) to verify the stability of our approach. As shown in Table 7, HiViG consistently maintains state-of-the-art performance with low standard deviations, confirming that our performance gains are robust to execution randomness and environmental noise.

Method	WALv2	ALab	WAA
Base agent	12.6 (0.8)	44.4 (1.2)	35.1 (0.6)
OpenCUA	15.8 (1.5)	46.1 (0.8)	35.1 (0.1)
SE-WSM	16.5 (0.8)	44.9 (1.3)	32.0 (0.5)
CGI (32B)	16.5 (0.8)	44.0 (0.5)	34.1 (0.9)
CGI (8B)	14.5 (2.0)	46.5 (0.7)	31.5 (0.8)
GUI-Critic-R1	11.9 (1.6)	49.0 (0.4)	33.7 (0.9)
HiViG (Ours)	**25.5 (0.4)**	**51.9 (0.4)**	**36.7 (0.6)**

Table 7: **Robustness analysis.** Average success rate (%) and standard deviation (std) over $n = 3$ runs on Qwen3-VL-32B-Thinking. WALv2: WebArenaLitev2, ALab: AndroidLab, WAA: WindowsAgentArena.

E.9 Failure Case Analysis

While HiViG framework significantly improves history awareness and error recovery for policies across diverse domains, our qualitative analysis reveals failure modes stemming from the model's capacity limit in fine-grained visual discrimination. First, during visually grounded error analysis, the critic occasionally misses nuanced visual discrepancies. For instance, when an agent is instructed to draw a "red" circle in Windows Paint but erroneously targets the "dark red" color option, HiViG fails to detect this error. This indicates that while the critic excels at broad visual grounding, it can struggle with highly precise visual feature recognition, such as subtle color variations. Second, during history state tracking, the critic can sometimes generate inaccurate multi-step achieved goals due to textual bias. For instance, when the agent attempted to delete an alarm in a mobile environment, the executed action merely hid the alarm's detailed configuration view, leaving the alarm itself present on the screen. However, mistaking this hidden view for a deleted state, the critic incorrectly recorded a successful deletion in its compressed macro-action history. Ultimately, both failure modes underscore the need for a more fine-grained understanding of visual features and state-to-state GUI dynamics, suggesting room for further improvements to our test-time intervention framework to continue pushing the performance limits of policies.

E.10 Detailed performance

Table 8, Table 9, and Table 10 show detailed performance of WebArenaliteV2, AndroidLab, and WindowsAgentArena benchmarks, respectively.

F Large Language Model (LLM) Use

In our research, we employed LLMs for training data construction and writing assistance. During writing, LLMs were utilized for sentence-level refinement and grammatical polishing. All AI-

generated suggestions were carefully reviewed and edited by the authors to maintain the coherency and accuracy.

Method	Admin	GitLab	Shopping	Map	Reddit	Overall
Computer Use Agent: Qwen3-VL-32B-Thinking						
Base agent	11.4	16.7	15.9	3.9	15.8	13.0
OpenCUA	14.3	16.7	20.5	3.9	15.8	14.9
SE-WSM	11.4	16.7	15.9	3.9	5.3	11.7
CGI (32B)	17.1	16.7	25.0	3.9	15.8	16.9
CGI (8B)	14.3	13.3	13.6	3.9	15.8	12.3
GUI-Critic-R1	14.3	13.3	15.9	3.9	21.1	13.6
HiVig (Ours)	28.6	13.3	31.8	23.1	26.3	**25.3**
Computer Use Agent: Gemini-3-Flash						
Base agent	25.7	56.7	25.0	19.2	26.3	30.5
OpenCUA	22.9	53.3	22.7	23.1	31.6	29.9
SE-WSM	25.7	53.3	31.8	23.1	10.5	30.5
CGI (32B)	37.1	33.3	31.8	19.2	21.1	29.9
CGI (8B)	17.1	50.0	25.0	11.5	31.6	26.6
GUI-Critic-R1	8.6	36.7	27.3	15.4	21.1	22.1
HiVig (Ours)	42.9	70.0	38.6	34.6	42.1	**45.5**

Table 8: Comparison of test-time intervention methods on Qwen3-VL-32B-Thinking and Gemini-3-Flash in WebArenalitev2 benchmark.

Method	Bluecoins	Calendar	Cantook	Clock	Contacts	Map	Pimusic	Setting	Zoom	Overall
Computer Use Agent: Qwen3-VL-32B-Thinking										
Base agent	20.0	35.7	33.3	74.1	53.3	6.7	16.7	69.6	40.0	44.2
OpenCUA	33.3	28.6	41.7	70.4	53.3	6.7	25.0	73.9	60.0	47.1
SE-WSM	20.0	42.9	41.7	70.4	53.3	13.3	8.3	69.6	60.0	45.7
CGI (32B)	26.7	42.9	33.3	74.1	53.3	6.7	33.3	76.5	0.0	46.9
CGI (8B)	26.7	42.9	25.0	77.8	53.3	0.0	16.7	69.6	20.0	44.2
GUI-Critic-R1	40.0	42.9	16.7	77.9	53.3	13.3	25.0	73.9	60.0	49.3
HiVig (Ours)	40.0	42.9	50.0	70.4	60.0	13.3	16.7	78.3	60.0	**51.5**
Computer Use Agent: Gemini-3-Flash										
Base agent	66.7	42.9	75.0	74.1	53.3	13.3	25.0	78.3	80.0	58.0
OpenCUA	60.0	42.9	50.0	81.5	26.7	40.0	33.3	78.3	80.0	57.2
SE-WSM	53.3	42.9	58.3	77.9	40.0	40.0	33.3	73.9	80.0	57.2
CGI (32B)	60.0	42.9	50.0	74.1	33.3	40.0	33.3	78.3	60.0	55.8
CGI (8B)	60.0	35.7	58.3	88.9	26.7	26.7	33.3	73.9	40.0	55.1
GUI-Critic-R1	60.0	42.9	50.0	66.7	46.7	26.7	25.0	73.9	80.0	53.6
HiVig (Ours)	73.3	42.9	50.0	88.9	40.0	40.0	33.3	78.3	80.0	**61.6**

Table 9: Comparison of test-time intervention methods on Qwen3-VL-32B-Thinking and Gemini-3-Flash in AndroidLab benchmark.

Method	Office	Web Browsing	Windows System	Code	Media	Windows Utilities	Overall
Computer Use Agent: Qwen3-VL-32B-Thinking							
Base agent	4.7	57.1	50.0	58.3	27.7	50.0	35.7
OpenCUA	2.3	61.9	70.8	50.0	24.5	25.0	35.2
SE-WSM	2.3	52.4	58.3	50.0	22.9	25.0	31.6
CGI (32B)	4.7	47.6	58.3	50.0	37.2	25.0	33.7
CGI (8B)	4.7	47.6	58.3	41.7	28.1	25.0	31.0
GUI-Critic-R1	4.7	61.9	45.8	54.2	28.1	41.7	34.4
HiVIG (Ours)	7.0	57.1	58.3	58.3	28.9	50.0	**38.0**
Computer Use Agent: Gemini-3-Flash							
Base agent	4.7	52.4	58.3	45.8	32.9	58.3	35.8
OpenCUA	4.7	52.4	58.3	45.8	32.9	58.3	35.8
SE-WSM	4.7	42.9	62.5	45.8	28.1	66.7	35.1
CGI (32B)	4.7	47.6	62.5	58.3	32.9	50.0	37.9
CGI (8B)	4.7	57.1	62.5	54.2	37.7	41.7	37.9
GUI-Critic-R1	2.3	57.1	58.3	45.8	19.3	41.7	32.5
HiVIG (Ours)	23.3	57.1	62.5	54.2	28.9	66.7	**44.2**

Table 10: Comparison of test-time intervention methods on Qwen3-VL-32B-Thinking and Gemini-3-Flash in WindowsAgentArena benchmark.

Model	Mobile		Desktop		Web		Overall	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1
Qwen2.5-VL-7B	58.5	65.8	61.2	69.4	54.7	64.2	58.3	66.2
Qwen3-VL-8B	61.6	62.3	59.9	63.9	60.1	60.4	61.0	62.3
UI-TARS-1.5-7B	56.2	66.4	58.6	69.6	60.1	70.1	57.5	67.8
ScaleCUA-7B	50.2	59.9	44.7	58.8	34.5	49.2	45.9	57.5
ScaleCUA-32B	51.6	63.3	49.3	64.8	50.7	65.4	51.0	64.1
SE-WSM-7B	50.9	17.0	50.7	13.8	41.2	18.7	48.9	16.8
GUI-Critic-R1	59.1	66.5	59.2	70.2	60.8	72.9	59.5	68.8
OS-Oracle-7B-SFT	63.5	71.2	63.2	72.0	62.2	71.7	63.1	71.5
OS-Oracle-7B	**70.8**	**74.3**	**65.8**	**71.1**	62.2	70.5	**68.0**	**72.8**
HiVIG-critic (Ours)	57.0	69.7	57.9	70.1	**65.5**	**76.1**	59.3	71.4

Table 11: Comparison of critic performance on OS-Critic-Bench. Acc and F1 denote Accuracy and F1-score, respectively. Baseline performances are from Wu et al. (2025b).

System Prompt of History State Tracking

You are the Macro-Strategy and Reflection module for a GUI navigation agent. Your task is to analyze the agent's recent action, perform a visual diff, and synthesize a strategic summary that explicitly connects the current visual state to the User Instruction.

Task Description
You will be provided with:
- User Instruction (Goal): The overall, high-level task the user wants the agent to accomplish.
- Previous Summary: The memory bank from the last turn.
- Previous Observation: The screenshot visual representation of the screen BEFORE the proposed step.
- Previous Proposed Step: The combined verbalized intent (Action: ...) and the executable command (<tool_call>...</tool_call>) previously suggested by the agent for the previous observation.
- Current Observation: The screenshot visual representation of the screen AFTER the proposed step.

Your task is to act as a summary model. You must determine if the agent's current trajectory is actively converging on the User Instruction.

Action Space
{platform_action_space}

Output Format
You must strictly follow this exact structure. Before outputting the final summary, you must reason through the visual state and logical deduction step-by-step inside a <think> block.

<think>
In your monologue, you must naturally flow through these logical steps:

- Perform a fast visual diff. State the 'Previous Proposed Step', compare the 'Previous Observation' to the 'Current Observation', and explicitly describe exactly what changed physically on the screen. Definitively conclude whether the 'Previous Proposed Step' resulted in a UI update or a silent failure.

- Map the verified 'Current Observation' to the 'User Instruction (Goal)'. Identify the visual gap: what goal-related UI elements are currently present, and what are missing?
</think>
<summary>
Progress Summary: [LONG TERM MEMORY: Synthesize the 'Previous Summary' with the current result. Compress completed micro-steps into dense macro-achievements (e.g., "Successfully logged in"). Only append the new action if it succeeded.]
Last Action Result: [SHORT TERM MEMORY: An objective, static description of exactly what the last action accomplished visually.]
</summary>

Figure 6: Input prompt for H1V1G for history state tracking at test-time.

System Prompt of Visually Grounded Error Analysis

You are an expert GUI Environment Simulator and Action Critic for a GUI navigation agent.

Task Description
You will be provided with:
- User Instruction (Goal): The overall, high-level task the user wants the agent to accomplish.
- Trajectory: A sequential record of the agent's past proposed steps.
- Current Observation: The screenshot visual representation of the current screen. **Note: IF the agent's proposed step involves spatial coordinates, a visual marker (red 'X' or blue arrow) has been injected into this image. If the action is non-spatial (e.g., type, wait), the image is clean.**
- Current Proposed Step: The combined verbalized intent (Action: ...) and the executable command (<tool_call>...</tool_call>) suggested by the agent for the current observation.

Your task is to predict the physical and logical consequences of the agent's action and provide goal-oriented environmental feedback so the agent can deduce the correct action itself.

Action Space
{platform_action_space}

Error Dimension Taxonomy
{error_dimension_str}

Output Format
You must strictly follow this exact structure. Before predicting the final evaluation, you must reason through the visual state and logical deduction step-by-step inside a <think> block.

<think>
In your monologue, you must naturally flow through these logical steps:

In your first paragraph, start exactly with this format: "Action String Check: [Insert the EXACT raw <tool_call> string here] - Syntax is [Valid/Invalid]." (Note any JSON parsing errors if invalid).
CRITICAL VISUAL GROUNDING: Next, you MUST evaluate ALL `<tool_call>` blocks in the 'Current Proposed Step' based on the arguments provided in the JSON.
- **If ANY tool call contains `coordinate` arguments (e.g., click, swipe, mouse_move):** The coordinates perfectly match the injected visual marker. Locate the **red 'X' marker** or the **blue arrow**. To prevent spatial hallucinations, you must perform a strict, literal visual scan:
  1. **Literal Center:** First, read the exact text or identify the specific icon located directly beneath the exact center intersection of the marker.
  2. **Adjacency Check:** Identify the elements immediately adjacent (left/right/above/below) to the marker.
  3. **Final Identification:** Based strictly on the literal center, explicitly describe exactly what UI element (or empty space) is targeted. (e.g., "The X is directly on the 'Subscribe' text, adjacent to the '0' counter. Therefore, it targets the Subscribe button, not the counter.")
- **If NO tool calls in the 'Current Proposed Step' contain `coordinate` arguments (e.g., ONLY scroll, type, key, wait, terminate):** You are STRICTLY FORBIDDEN from looking for or mentioning a red 'X' or blue arrow. **CRITICAL WARNING: If you see a red 'X' in the image during a non-coordinate action, it is a residual artifact from a previous step. You MUST completely ignore it.** Evaluate the global screen state and the logical parameters provided in the 'Current Proposed Step' JSON.
Next, analyze the 'Trajectory' against the 'User Instruction (Goal)' to establish the current progress state.
Finally, if a verbal intent (Action: ...) is provided, explicitly compare what the agent claims it is doing against the visual reality you just observed in the context of this current state, calling out any contradictions. If no verbal intent is provided, explicitly state what UI element the action targets in the context of the current state.

In your second paragraph, using your knowledge of GUI interfaces, predict exactly what will physically happen to the UI layout if this specific action is executed. Use forward-predictive future tense.

In your third paragraph, compare your predicted consequence directly to the 'User Instruction (Goal)'. You must organically deduce why this predicted structural UI change succeeds or fails. If it succeeds, conclude 'Good'. If it fails, conclude 'Bad', state the matching 'Target Error Dimension', and explain the UI physics rule that the agent violated.
</think>
<criticism>
[If the action is correct, output exactly this format:]
Overall Grading: Good
Explanation: [Describe the immediate physical/structural consequence of this action in the future tense. Then, explicitly state the universal UI rule or environmental mechanic that makes this interaction successful.]
[If the action is flawed, output exactly this format:]
Overall Grading: Bad
Error Dimension: [You MUST select the exact matching category from the Error Dimension Taxonomy above].
Explanation: [Describe the immediate physical/structural consequence of the flawed action in the future tense. Then, state the universal UI rule the agent violated, and explain the correct environmental mechanic required for this type of interaction.]
</criticism>

Figure 7: Input prompt for HiVIG for visually-grounded error analysis at test-time.

Desktop GUI Action Space

<tools>
{"type": "function", "function": {"name": "computer_use", "description": "Use a mouse and keyboard to interact with a computer, and take screenshots.
* This is an interface to a desktop GUI. You do not have access to a terminal or applications menu. 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.
* The screen's resolution is 1000x1000.
* 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.", "parameters": {"properties": {"action": {"description": "The action to perform. The available actions are:
* `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.
* `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.
* `wait`: Wait specified seconds for the change to happen.
* `terminate`: Terminate the current task and report its completion status.
* `answer`: Answer a question.
* `interact`: Resolve the blocking window by interacting with the user.", "enum": ["key", "type", "mouse_move", "left_click", "left_click_drag", "right_click", "middle_click", "double_click", "triple_click", "scroll", "hscroll", "wait", "terminate", "answer", "interact"], "type": "string"}, "keys": {"description": "Required only by `action=key`.", "type": "array"}, "text": {"description": "Required only by `action=type`, `action=answer` and `action=interact`.", "type": "string"}, "coordinate": {"description": "(x, y): The x (pixels from the left edge) and y (pixels from the top edge) coordinates to move the mouse to. Required only by `action=mouse_move` and `action=left_click_drag`.", "type": "array"}, "pixels": {"description": "The amount of scrolling to perform. Positive values scroll up, negative values scroll down. Required only by `action=scroll` and `action=hscroll`.", "type": "number"}, "time": {"description": "The seconds to wait. Required only by `action=wait`.", "type": "number"}, "status": {"description": "The status of the task. Required only by `action=terminate`.", "type": "string", "enum": ["success", "failure"]}}, "required": ["action"], "type": "object"}}}
</tools>

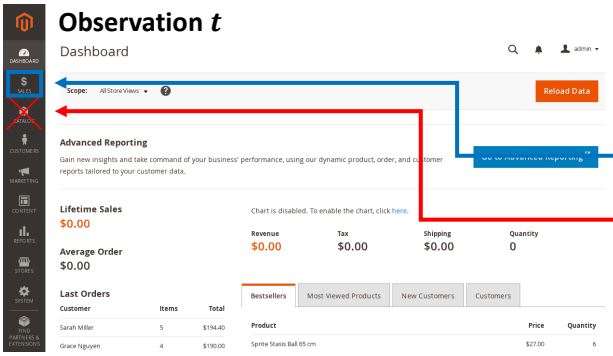
Figure 8: Action space definition in desktop (e.g., web, WindowsOS, MacOS) GUI environments.

Mobile GUI Action Space

<tools>
{"type": "function", "function": {"name_for_human": "mobile_use", "name": "mobile_use", "description": "Use a touchscreen to interact with a mobile device, and take screenshots.
* This is an interface to a mobile device with touchscreen. You can perform actions like clicking, typing, swiping, etc.
* 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.
* The screen's resolution is 1000x1000.
* 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.", "parameters": {"properties": {"action": {"description": "The action to perform. The available actions are:
* `key`: Perform a key event on the mobile device.
 - This supports adb's `keyevent` syntax.
 - Examples: \"volume_up\", \"volume_down\", \"power\", \"camera\", \"clear\".
* `click`: Click the point on the screen with coordinate (x, y).
* `long_press`: Press the point on the screen with coordinate (x, y) for specified seconds.
* `swipe`: Swipe from the starting point with coordinate (x, y) to the end point with coordinates2 (x2, y2).
* `type`: Input the specified text into the activated input box.
* `system_button`: Press the system button.
* `open`: Open an app on the device.
* `wait`: Wait specified seconds for the change to happen.
* `answer`: Terminate the current task and output the answer.
* `interact`: Resolve the blocking window by interacting with the user.
* `terminate`: Terminate the current task and report its completion status.", "enum": ["key", "click", "long_press", "swipe", "type", "system_button", "open", "wait", "answer", "interact", "terminate"], "type": "string"}, "coordinate": {"description": "(x, y): The x (pixels from the left edge) and y (pixels from the top edge) coordinates to move the mouse to. Required only by `action=click`, `action=long_press`, and `action=swipe`.", "type": "array"}, "coordinate2": {"description": "(x, y): The x (pixels from the left edge) and y (pixels from the top edge) coordinates to move the mouse to. Required only by `action=swipe`.", "type": "array"}, "text": {"description": "Required only by `action=key`, `action=type`, `action=open`, `action=answer`,and `action=interact`.", "type": "string"}, "time": {"description": "The seconds to wait. Required only by `action=long_press` and `action=wait`.", "type": "number"}, "button": {"description": "Back means returning to the previous interface, Home means returning to the desktop, Menu means opening the application background menu, and Enter means pressing the enter. Required only by `action=system_button`", "enum": ["Back", "Home", "Menu", "Enter"], "type": "string"}, "status": {"description": "The status of the task. Required only by `action=terminate`.", "type": "string", "enum": ["success", "failure"]}}, "required": ["action"], "type": "object"}, "args_format": "Format the arguments as a JSON object."}}
</tools>

Figure 9: Action space definition in mobile GUI environments.

User Instruction: What's the total number of items sold in the most recent 7 orders?



Observation t

Computer Use Agent

Proposed Action t

Click on the Sales menu in the sidebar.
<tool_call>
{"name": "computer_use", "arguments": {"action": "left_click", "coordinate": [34, 310]}}
</tool_call>

Verbal Feedback

Error Analysis by GUI-Critic-R1

<thinking>
Observation: The screenshot shows the Magento admin dashboard with various sections such as "Advanced Reporting," "Lifetime Sales," and "Last Orders." The "Sales" menu option is visible in the sidebar, indicating it can be interacted with.
Possible Result: **Clicking on the "Sales" menu will likely navigate to a page displaying sales-related data,** including order details, which could help in identifying the most recent 7 orders and their item counts.
Critique: The decision to click on the "Sales" menu is correct because accessing sales-related data is necessary to fulfill the user's request for the total number of items sold in the most recent 7 orders. This action aligns with the goal of gathering the required information.
</thinking>
<score>**Correct**
</score>
<suggestion>
The agent correctly clicked on the "Sales" menu to access relevant sales data for further analysis.
</suggestion>

Visually Grounded Error Analysis by Our critic

<think>
The marker targets the "CATALOG" sidebar item (red cube icon), which directly **contradicts the agent's stated intent to click the "Sales" icon. Executing this click will navigate to the product catalog page**, which exclusively contains product management tools and zero order-related metrics. The goal requires calculating items from the 7 most recent orders. **By violating the UI rule that order data is housed under "Sales," the agent traps itself in an irrelevant section** and cannot complete the task.
</think>
<criticism>
Overall Grading: **Bad**
Error Dimension: **Action-Operation Misalignment**
Explanation: **The action will navigate to the Catalog section, which displays product management tools rather than order data**. The UI rule is that order-related metrics and lists are exclusively accessible through the Sales section, not the Catalog. Since the goal requires analyzing order quantities, **this misnavigation prevents access to the necessary order details, making the step ineffective**.
</criticism>

Figure 10: Illustration of the visual grounding limitation in existing critics. When the policy generates an action with a logically correct textual intent ("Click on the Sales menu") but outputs misaligned spatial coordinates that actually target the "Catalog" icon, the baseline critic erroneously approves the action by over-relying on the text. In contrast, our proposed critic evaluates the precise physical execution using a visual marker, successfully detecting the spatial mismatch and intercepting the Action-Operation Misalignment before it executes.

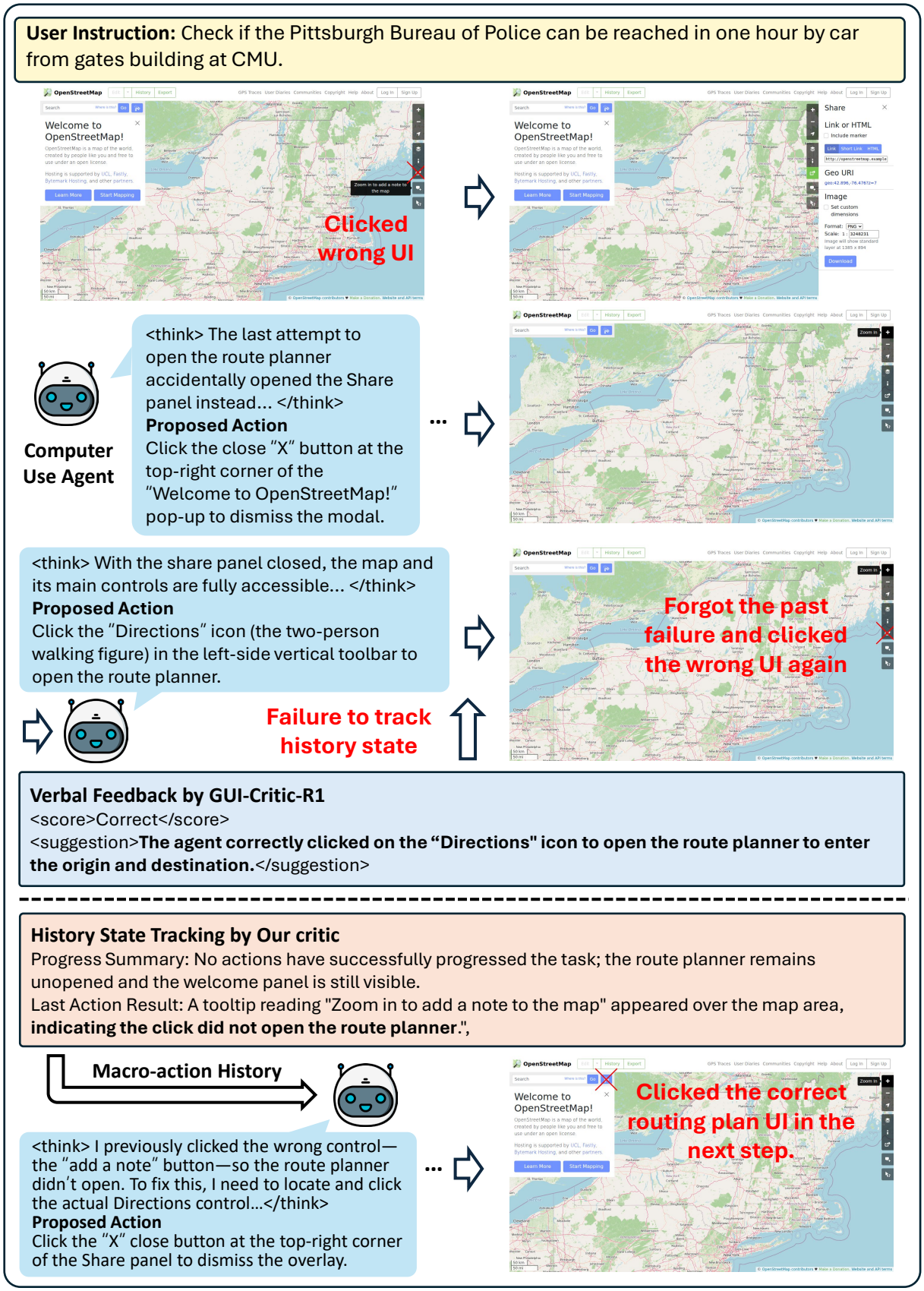
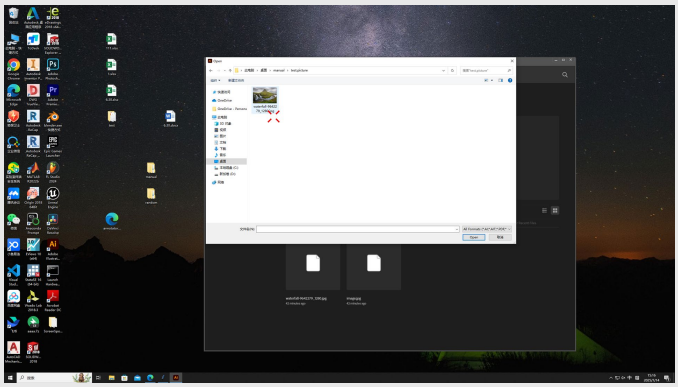
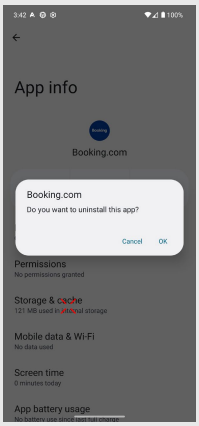


Figure 11: Illustration of the history state tracking limitation in existing critics. When the policy generates an action that clicked the wrong UI, the baseline critic fails to log past failures, leaving the policy without the feedback needed to avoid making the same mistake. In contrast, our proposed critic maintains a macro-action history, tracking the history state and enabling the policy to recognize prior failures to proceed in long-horizon GUI task.

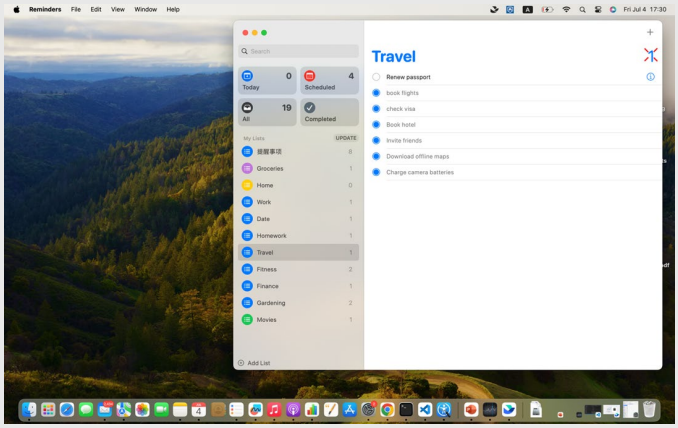


Task: Use adobe illustrator to open the picture ...waterfall-9642279_1280.jpg
Proposed Action: In the file explorer window, click on the image file named "waterfall-9642279_1280.jpg" to select it. <tool_call>{"name": "computer_use", "arguments": {"action": "left_click", "coordinate": [402, 295]}}</tool_call>
Overall Grading: Bad
Error Dimension: Grounding/Spatial Error
Explanation: The click will land in the empty space beneath the image file entry, resulting in no selection and leaving the filename field blank.

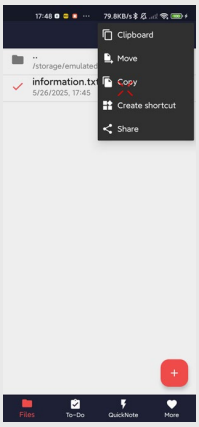


Task: Remove the booking.com application.
Proposed Action: Click on the 'Storage & cache' section to review storage details <tool_call>{"name": "mobile_use", "arguments": {"action": "click", "coordinate": [338, 722]}}</tool_call>
Overall Grading: Bad
Error Dimension: Procedural Prerequisite Neglect
Explanation: The click on the background "Storage & cache" section will be ignored because the active confirmation dialog blocks all interactions with elements behind it.

Figure 12: Representative examples of common CUA failure modes. (Top) *Grounding/Spatial Error*: The agent forms the correct semantic intent but outputs flawed coordinates, clicking the empty space beneath the file instead of selecting it. (Bottom) *Procedural Prerequisite Neglect*: The agent attempts to interact with a background UI element while a confirmation dialog is active, blocking the click and resulting in a silent failure.

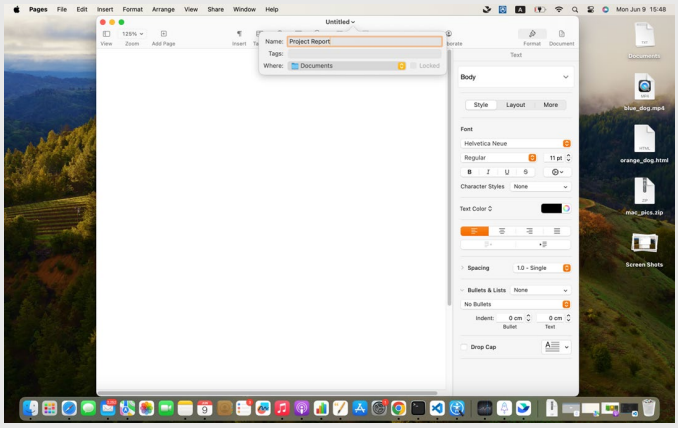


Task: In the Reminders app, mark every reminder in the "Travel" list as completed
Proposed Action: Click the blue '1' badge in the top-right corner of the Travel list to complete the remaining task
Overall Grading: Bad
Error Dimension: Semantic Error
<tool_call>{"name": "computer_use", "arguments": {"action": "left_click", "coordinate": [964, 121]}}</tool_call>
Explanation: Clicking the numeric badge will not change any reminder's completion state because it is a read-only counter that reflects the number of uncompleted items, not a control to mark tasks done.

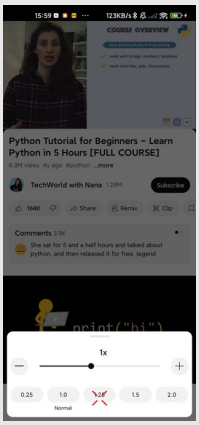


Task: Relocate the note information.txt from ailab to pjlab in the Markor app.
Proposed Action: Click on 'Rename' in the file options menu to prepare the file for relocation
<tool_call>{"name": "mobile_use", "arguments": {"action": "click", "coordinate": [634, 200]}}</tool_call>
Overall Grading: Bad
Error Dimension: Visual Hallucination
Explanation: The action will result in a silent failure because the menu does not contain a "Rename" option, so no operation will be initiated.

Figure 13: Representative examples of common CUA failure modes (continued). (Top) *Semantic Error*: The agent misunderstands UI affordances, attempting to click a read-only numeric counter. (Bottom) *Visual Hallucination*: The agent hallucinates a UI element, attempting to click a "Rename" option that does not exist in the current screenshot.

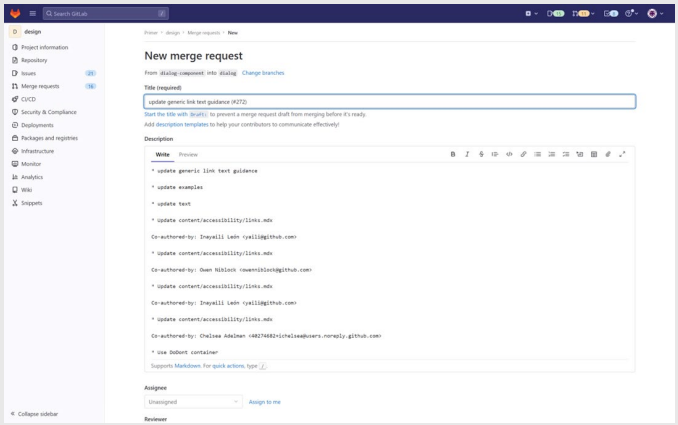


Task: In Pages, rename the document to 'Project Report
Proposed Action: Press Enter to confirm the document name change
<tool_call>{"name": "computer_use", "arguments": {"action": "key"}}</tool_call>
Overall Grading: Bad
Error Dimension: Action Formulation Error
Explanation: The key action will trigger a parser crash because it lacks the required "keys" array to specify which key to press, leaving the dialog unchanged.



Task: While playing any video, tap the settings icon, select “Playback Speed”, and change it to 1.5x.
Proposed Action: Tap the 1.25 speed option to increase video playback speed
<tool_call>{"name": "mobile_use", "arguments": {"action": "click", "coordinate": [500, 945]}}</tool_call>
Overall Grading: Bad
Error Dimension: Constraint Neglect
Explanation: The playback speed will change to 1.25x instead of 1.5x, violating the explicit requirement to select the 1.5x option.

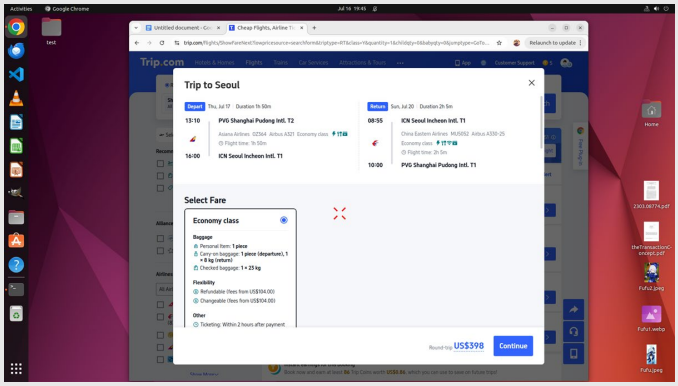
Figure 14: Representative examples of common CUA failure modes (continued). (Top) *Action Formulation Error*: The agent intends to press ’Enter’ but omits the required key parameter in the JSON payload, crashing the parser. (Bottom) *Constraint Neglect*: The agent successfully navigates the UI but selects a 1.25x playback speed instead of the explicitly requested 1.5x.



Task: Submit a request to merge dialog-component branch into dialog branch, assign Carol as the reviewer.
Proposed Action: Scroll down slightly to reveal the Reviewer section where Carol can be assigned as reviewer.
<tool_call>{"name": "computer_use", "arguments": {"action": "scroll", "pixels": -20}}</tool_call>
Overall Grading: Bad
Error Dimension: Parameter Vector Miscalibration
Explanation: The page will shift downward by only 20 pixels, an imperceptible amount that won't meaningfully reveal the Reviewer section.

Task: Send the message I just sent again to 464857546 via Simple SMS Messenger.
Proposed Action: Terminate the process as the task is complete
<tool_call>{"name": "mobile_use", "arguments": {"action": "terminate", "status": "success"}}</tool_call>
Overall Grading: Bad
Error Dimension: Termination Misjudgment
Explanation: The process halts abruptly without opening the conversation for 464857546 or sending the message, leaving the UI unchanged and the task incomplete.

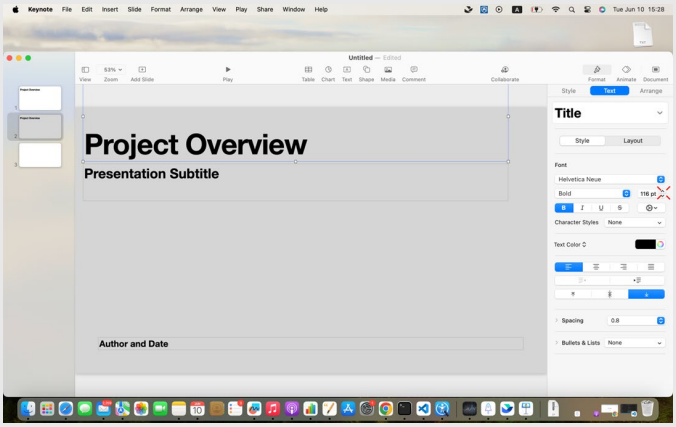
Figure 15: Representative examples of common CUA failure modes (continued). (Top) *Parameter Vector Miscalibration*: The agent attempts to scroll to off-screen content but issues a mere 20-pixel magnitude, failing to reveal the target 'Reviewer' section. (Bottom) *Termination Misjudgment*: The agent prematurely halts execution before opening the target conversation and sending the message.



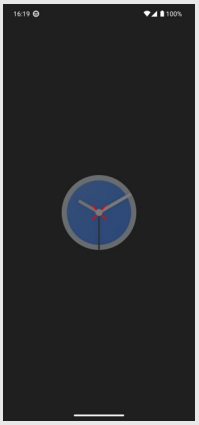
Task: Visit Trip.com and book a round-trip flight from Shanghai to Seoul, departing on July 17, 2025 and returning on July 20, 2025.
Proposed Action: Scroll down slightly to look for additional navigation controls that may be below the visible area.
<tool_call>{"name": "computer_use", "arguments": {"action": "mouse_move", "coordinate": [500, 554]}}</tool_call><tool_call>{"name": "computer_use", "arguments": {"action": "scroll", "pixels": -500}}</tool_call>
Overall Grading: Bad
Error Dimension: Observation Neglect
Explanation: The action will cause the page to scroll downward, moving the visible "Continue" button out of view and forcing the user to scroll back up to access the control that is already present.

Task: List the titles of my high priority tasks in the Tasks app, using commas to separate multiple entries.
Proposed Action: Open the Tasks app from your home screen to view your to-do list with upcoming assignments and appointments
<tool_call>{"name": "mobile_use", "arguments": {"action": "click", "coordinate": [500, 710]}}</tool_call>
Overall Grading: Bad
Error Dimension: Action-Operation Misalignment
Explanation: The screen will unexpectedly transition to the camera viewfinder, showing a live preview of the surroundings, shutter controls, and mode selection options instead of the task list interface.

Figure 16: Representative examples of common CUA failure modes (continued). (Top) *Observation Neglect*: The agent scrolls downward to search for controls, unnecessarily moving the already visible "Continue" button out of view. (Bottom) *Action-Operation Misalignment*: The agent intends to open the "Tasks" app but outputs coordinates for the camera app, launching the viewfinder instead.



Task: In Keynote, set the font size of the title to 36pt.
Perturbed Action: Click the small down arrow button next to the '116 pt' font size field to decrease the font size.
<tool_call>{"name": "computer_use", "arguments": {"action": "left_click", "coordinate": [986, 451]}}</tool_call>
Overall Grading: Bad
Error Dimension: Suboptimal Path
Explanation: The font size will increase by one step (e.g., to 117 pt) because the up arrow control raises the value, moving further from the 36 pt target.



Task: While playing any video, tap the settings icon, select “Playback Speed”, and change it to 1.5x.
Proposed Action: Click on the center of the clock face to navigate to the bedtime settings.
<tool_call>{"name": "mobile_use", "arguments": {"action": "click", "coordinate": [500, 500]}}</tool_call>
Overall Grading: Bad
Error Dimension: Timing and Latency Neglect
Explanation: The system will ignore the click as the app is still in a loading state with no interactive elements available.

Figure 17: Representative examples of common CUA failure modes (continued). (Top) *Suboptimal Path*: The agent attempts to decrease a font size but mistakenly clicks the "up" arrow, moving further from the target goal. (Bottom) *Timing and Latency Neglect*: The agent interacts with the UI during a non-interactive loading state, causing the system to ignore the premature click.