Reflection with Action-Induced Visual Differences for Desktop GUI Agents

Yijie Ma Chaoyue Niu Fan Wu Guihai Chen
Shanghai Jiao Tong University

Abstract

The Planner-Operator-Reflector (POR) framework is widely used in GUI agents to maintain objective alignment in complex tasks through modular collaboration. However, desktop GUIs introduce a key challenge: large, dense interfaces often exhibit subtle or scattered state changes, placing most of the burden on the reflector, which must compare pre- and post-action screens, while the planner and operator reason over a single state. Existing reflectors collapse change detection and outcome verification into one step, leaving evidence implicit and yielding weakly grounded decisions. To address this limitation, we propose Evidence-First Reflection (EFR), a two-stage reflector that explicitly decouples action-induced visual differences extraction from outcome verification. EFR identifies the action location and candidate changed regions with Set-of-Marks annotations, describes and filters action-relevant changes, and makes the final judgment from the cleaned evidence. This evidence-reasoning decoupled design makes reflection better grounded in screen transitions, while reducing both visual search complexity and reasoning burden. Experiments on OSWorld-Verified and WindowsAgentArena demonstrate that EFR improves reflector accuracy by 7.11%, yielding average end-to-end task success gains of 5.94% and 4.92% on the two benchmarks, respectively. [1]

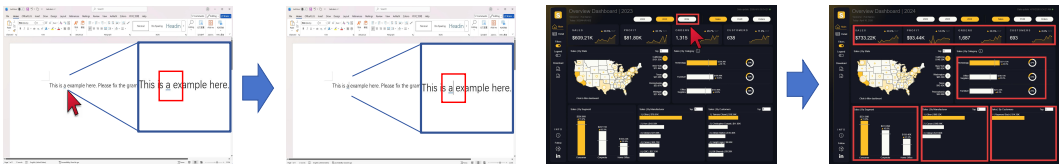
1 Introduction

Graphical User Interface (GUI) agents [1, 8, 17, 19, 22, 23, 29, 33, 34] are designed to execute user instructions by interacting with digital interfaces in a human-like manner, typically leveraging vision-language models (VLMs). By unifying the observation space into screenshots and the action space into human-like operations, GUI agents can handle diverse tasks across both mobile and desktop environments [6, 29]. This broader task scope introduces higher complexity, motivating existing studies to adopt the mainstream Planner-Operator-Reflector (POR) framework [1, 2, 11, 25, 32], which decomposes agent control into collaborative modules to maintain task alignment and reduce the burden on any single module. The planner formulates a high-level plan, the operator executes the corresponding low-level action, and the reflector assesses whether the action has successfully achieved the intended goal. Serving as a critical safeguard in this workflow, the reflector helps ensure action accuracy and prevents error accumulation over steps. It typically performs this assessment in a single inference pass by observing the pre- and post-action screenshots and comparing the observation with the intended action to generate a judgment [2, 12, 21, 25, 32].

However, reflection in desktop environments is generally more challenging than in mobile environments, as desktop screens are often several to even dozens of times larger. Such expansive interfaces tend to exhibit more complex layouts and a richer set of elements, making it difficult to comprehensively detect and interpret GUI changes [1, 12, 18, 33]. This challenge disproportionately affects the reflector compared to the planner and operator: while the latter mainly need to reason about the

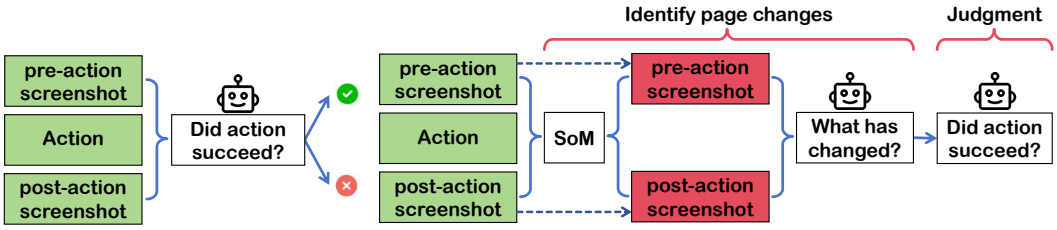
[1] The code is available at `https://github.com/JXMYJ/EFR`.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



(a) Only a cursor change. (b) Many page regions change simultaneously.

Figure 1: Two representative examples of interface changes in desktop environments. Fig. 1a: the action outcome is indicated by a subtle cursor movement after clicking the position of the letter "a", making the change easy to overlook. Fig. 1b: switching the year triggers extensive updates across the interface, making it challenging to organize all relevant evidence to assess the action's effect.



(a) Conventional Reflector. (b) EFR.

Figure 2: The conventional reflector performs one-step reasoning to judge action success directly, whereas EFR first describes page changes from SoM annotations as explicit evidence and then verifies the action based on the extracted evidence.

current interface state, the reflector must jointly analyze both pre- and post-action screenshots to determine whether an action has succeeded. Figure 1 shows two representative examples: one where the action outcome is extremely subtle, and the other where the action induces multiple, spatially distributed changes. In both cases, the reflector struggles to capture and organize the complete set of interface changes, leading it to judge the action outcome based on only a limited and potentially incorrect subset of changes. In our experiment with GUI-Owl-32B trajectories using the conventional POR framework on OSWorld-Verified [27, 28], 61 out of 223 failed trajectories (27.35%) are caused by incorrect reflector judgments. Among these reflection failures, 43 cases (70.49%) stem from page difference identification errors, while 18 cases (29.51%) result from success judgment errors given correctly identified page differences, as detailed in Appendix Table 8. This indicates that the conventional reflector is not sufficiently reliable in complex desktop environments.

By deconstructing the logic of conventional reflectors, we find that they always couple two operations with different roles: first establishing what changed as observable evidence, and then interpreting whether the evidence supports a successful action. When these operations are coupled into one inference step, the evidence used for the final judgment remains implicit and cannot be checked before the VLM reaches a conclusion. To address this limitation, we revisit the internal design of the reflector itself and propose **Evidence-First Reflection (EFR)**, a two-stage reflector that explicitly decouples visual evidence extraction from action outcome verification. It mitigates potential bias by decoupling the objective observation of page changes from subjective assumptions regarding intended action outcomes, while reducing the visual burden by repurposing Set-of-Marks (SoM), originally used to label candidate page elements, to annotate page changes before and after the action. Figure 2 shows the core difference between conventional reflector and EFR. Specifically, EFR first leverages SoM to annotate action-induced change regions and the grounded action location, prompting the VLM to objectively describe and filter action-relevant page changes as evidence. Then, it judges whether the action succeeded based on the cleaned evidence and the intended action. By separating "what changed" from "whether the action succeeded", EFR reduces both the visual burden and the reasoning burden of reflection. In summary, the key contributions include:

- We observe that the inherent complexity of desktop environments substantially complicates change identification, thereby compromising the reliability of conventional reflectors that infer action success directly in a single reasoning step.

- We reconstruct the reflector as a two-stage sequential process that explicitly separates page difference identification from action success judgment. The first stage describes and organizes action-relevant visual changes from SoM annotations, and the second stage verifies action success based on the extracted evidence, making the dependency between evidence construction and outcome judgment explicit. This design grounds the final judgment in a checkable evidence chain and reduces interference between visual search and outcome reasoning.
- We implement the POR framework following Mobile-Agent-v3 [32] and evaluate EFR with both GUI-specialized and general backbones on OSWorld-Verified [27, 28] and WindowsAgentArena [4] benchmarks. Averaged over three runs, EFR improves end-to-end task success rates across all model-benchmark settings, with average gains of 5.94% and 4.92%, respectively. Further analysis demonstrates that the performance gains are strongly correlated with improvements in reflector accuracy. Ablation studies validate the necessity of each module. Notably, the performance improvement comes with moderate per-task latency overheads, averaging 15.8% and 13.3%, respectively.

2 Related Work

POR Framework. The key of POR framework is to explicitly decouple planning, execution, and feedback verification [15, 20, 21]. This paradigm shares a lineage with the reasoning-acting loop exemplified by ReAct [31]. Many studies adopt this paradigm and further refined it to better handle complex tasks. For instance, Mobile-Agent-v3 [32] introduces Planner-Operator-Reflector-Notetaker framework to handle multi-step navigation and error correction. Mobile-Agent-E [25] and MobileUse [11] enhance hierarchical planning and introduce an on-demand, reflection-driven workflow. Agent-S2 [2] uses dynamic collaboration to plan various modules. While these studies largely retain the POR framework as-is and enhance it with additional components or strategies, our work revisits its internal modular design by reconstructing the reflector module.

Reflector. Reflector is typically used to convert failures into reusable memory or to support hierarchical closed-loop correction [7, 15, 20]. In GUI agents, the use of reflectors has been further extended within the overall agent pipeline: MobileUse [11] introduces hierarchical reflection to identify and correct errors across different granularities, InfiGUIAgent [13] and D-Artemis [16] perform reflection at different stages of the action execution process, and BacktrackAgent [26] leverages reflection for trajectory rollback. Their improvements are largely about deploying the reflector in different roles or stages of the agent pipeline. As a result, most approaches still implicitly assume that the reflector can operate as a single-step module. In contrast, our work focuses on its internal reasoning logic and improves it by explicitly decoupling evidence extraction from success judgment.

SoM in GUI Agent. SoM annotates interface content by overlaying explicit markers, such as letters, numbers, or bounding boxes, on screenshots [30]. In prior GUI agents, it was used to identify candidate elements before taking actions [14, 33, 34]. For instance, AppAgent [34] overlays semi-transparent numeric markers onto candidate elements parsed from XML on mobile screenshots, enabling the model to execute operations using numeric indices rather than raw coordinates. UFO [33] simultaneously provides the agent with a raw screenshot and an annotated screenshot, encoding UI controls with numbers and colors. In addition to these specific methods, environments like VisualWebArena [10] and OSWorld [27] also employ standardized element-level SoM observation representations to facilitate element recognition. In contrast, we extend SoM beyond target-element localization and apply it to post-action page-change identification within the reflection module.

3 Problem Formulation

We first describe the workflow of the conventional POR framework, which employs a closed-loop iterative process to complete tasks. Within a single execution round, the Planner first formulates a comprehensive plan and decomposes it into discrete sub-goals based on the current state and overall task goal. Subsequently, the Operator receives the active sub-goal and generates a grounded action for execution. Finally, the Reflector verifies whether the executed action achieves the intended goal by comparing pre- and post-action screenshots. Formally, at each step t, the operations of the modules are defined as follows:

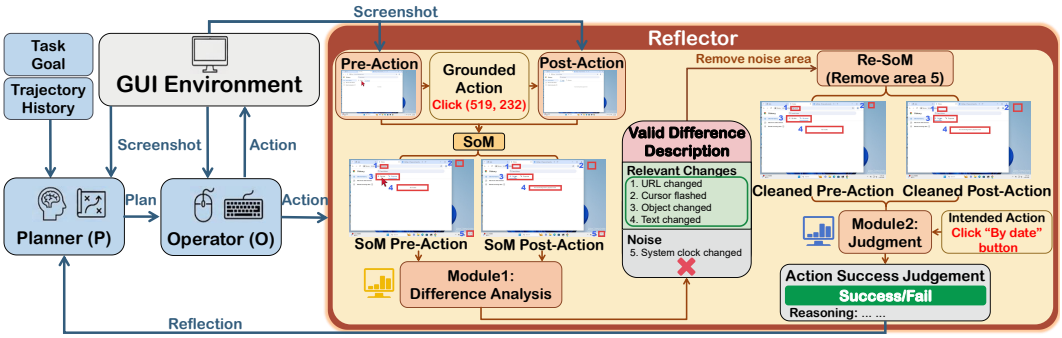


Figure 3: Overview of EFR in POR framework. Given the pre-action screenshot S_{pre}, post-action screenshot S_{post}, grounded action description A_{ground}, and intended action description A_{intend}, EFR first annotates candidate visual differences and the action location with SoM, then filters action-relevant evidence, and finally judges action success based on the cleaned evidence.

Planner. Given the task goal G, the Planner leverages the current screenshot S_t together with the previous state, comprising the action history $H_{<t}$ and the reflection from the preceding step R_{t-1}, to derive the next sub-goal P_t:

$$P_t = f_{\text{planner}}(G, S_t, H_{<t}, R_{t-1}). \tag{1}$$

Operator. The Operator then grounds the sub-goal P_t in the current visual observation S_t and outputs the executable action A_t:

$$A_t = f_{\text{operator}}(P_t, S_t). \tag{2}$$

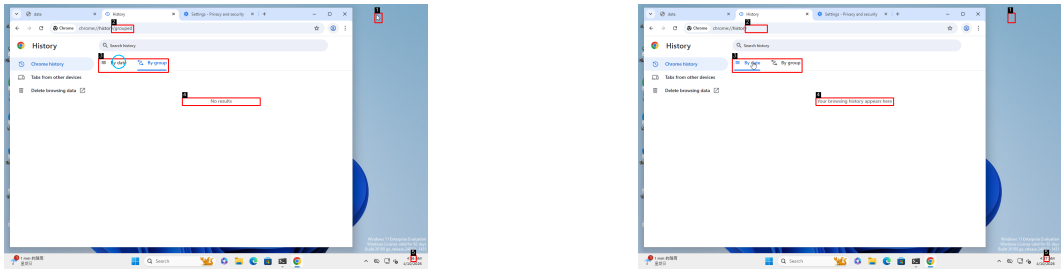
Reflector. Finally, after execution, the Reflector compares the pre-action screenshot S_t^{pre} with the post-action screenshot S_t^{post} to assess the outcome of the action A_t and produce the reflection R_t:

$$R_t = f_{\text{reflector}}(P_t, A_t, S_t^{pre}, S_t^{post}). \tag{3}$$

Despite the modular decomposition of the POR framework, the reflection phase remains a monolithic challenge. To produce a reliable assessment, the Reflector must simultaneously address two distinct objectives: (1) identifying visual transformations between pre- and post-action screenshots, (2) determining whether these changes satisfy the intended goal. Conventional methodologies typically process these steps in a single pass by directly feeding the action description and both screenshots into the VLM. However, this design is brittle in desktop environments where screenshots are high-resolution and information-dense. Specifically, subtle changes are often overlooked, while intricate changes can lead to fragmented analysis and hasty conclusions. To address these limitations, it is essential to equip the reflector with the ability to comprehensively perceive the page changes, and to assess action correctness strictly on the basis of the observed state transition.

4 Method

In this section, we introduce **EFR**, a two-stage evidence-based reflection framework designed to extract salient features under complex visual transformations and formulate judgments grounded in explicit evidence. Figure 3 illustrates the overall framework, with detailed descriptions of each component provided in the subsequent subsections. Within the POR loop, EFR replaces the conventional reflector: after the planner generates a plan and the operator executes the corresponding action, EFR evaluates whether the action produced correct result and feeds the reflection result back into the next planning step. Rather than directly inferring action succeeded from raw pre- and post-action screenshots, EFR explicitly decouples visual change recognition from outcome verification. This approach is built upon a twofold strategy: first, we preprocess the screenshots to reduce the perceptual burden associated with detecting subtle or distributed interface changes; second, we implement a two-stage reasoning pipeline to mitigate the cognitive complexity of success evaluation. Specifically, EFR utilizes SoM to annotate both the grounded action location and visual differences between the screenshots, thereby constructing a foundation of explicit visual evidence. Then it systematically describes key changes and filters out environmental noise. By synthesizing this refined evidence with the intended goal, EFR determines the correctness of the executed action. Appendix C presents an example, illustrating the comparison between the conventional reflector and EFR.



(a) Pre-action marked screenshot

(b) Post-action marked screenshot

Figure 4: A concrete example of SoM annotation. Fig. 4a shows the pre-action marked screenshot, where the executed action location and difference regions are annotated. Fig. 4b shows the post-action marked screenshot, where difference regions are annotated.

4.1 Page Changes Annotated With SoM

Identifying page changes is a specialized perceptual task for which general-purpose VLMs often underperform, whereas task-specific computer vision methods are more effective and efficient. Instead of relying on the VLM to detect changes unaided, we first apply the SoM technique to explicitly annotate visual differences. Existing SoM methods were typically designed to assist target-element localization during action execution. We repurpose this idea for the reflection stage to support page-change analysis. This step explicitly converts dispersed differences between two screenshots into consolidated, structured evidence that the VLM can directly reference: changed regions are marked with red bounding boxes and indexed for region-level comparison. In addition, we annotate the action coordinates with blue circles to provide spatial context regarding where the action was executed. These explicit visual markers allow the VLM to directly locate salient changes on the page and reason over them as structured visual evidence.

Given the pre-action screenshot S_{pre}, the post-action screenshot S_{post}, and the grounded action A_{ground} which contains action type and coordinates, we first identify candidate changes through pixel-wise comparison. Nearby changed regions are then merged into coherent visual-change areas, which are finally mapped to rectangular bounding boxes. These boxes and their corresponding indices are then overlaid onto both screenshots. In addition, the action coordinates are annotated on the pre-action screenshot. For actions without explicit screen coordinates, such as keyboard shortcuts, we omit the annotation but provide the executed action description to the subsequent difference analysis stage. This procedure yields two intermediate representations: S^{som}_{pre}, which depicts the pre-action screenshot marked with change areas and action coordinates, and S^{som}_{post}, which only displays change areas on the post-action screenshot:

$$(S^{som}_{pre}, S^{som}_{post}) = f_{\text{som}}(S_{pre}, S_{post}, A_{ground}). \tag{4}$$

By incorporating SoM, we explicitly indicate the locations of visual changes. An illustrative example is presented in Figure 4. Further details regarding the identification and annotation process are provided in the Appendix A. This approach reduces the perceptual load, thereby enabling the VLM to perform localized comparisons within the annotated regions. The annotations describe changes in screen regions, without establishing correspondences between individual GUI elements. From subtle local updates to broader transitions, this region-based representation exposes the spatial extent and distribution of detected changes, helping the VLM interpret local evidence in the context of the overall screen transition.

4.2 Action-Induced Page Difference Analysis and Filtering

After obtaining the annotated screenshots, the next phase involves an intermediate step aimed at extracting explicit visual evidence. An illustrative example is presented in Figure 5. During this stage, the VLM is provided with the marked pre- and post-action screenshots (S^{som}_{pre}, S^{som}_{post}) and the grounded action A_{ground}. Importantly, A_{ground} contains only the action type and its coordinates, rather than the original intended action A_{intend}. For example, the intended action may be "click the button in the upper-left corner," whereas the grounded action is represented as "click at coordinate (x, y)." This design encourages the reflector to extract evidence from the actual execution result

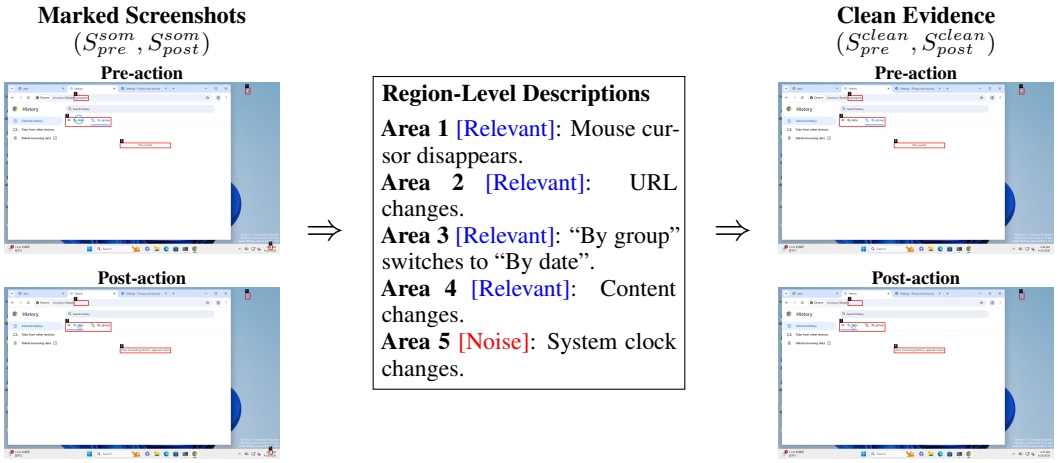


Figure 5: From total changes to clean evidence. The reflector first compares the annotated pre-action and post-action screenshots and outputs change descriptions together with relevant/noise tags. In this example, the bottom right Area 5 is filtered out as environment noise. The remaining relevant regions are then redrawn to form the clean evidence pair.

rather than from the expected intention, reducing intention bias and improving the objectivity of the intermediate visual evidence.

Given these inputs, the VLM generates explicit visual evidence in the form of region-level change descriptions. For each annotated region, it compares the corresponding visual contents in the marked pre- and post-action screenshots and independently describes the observed change. Since all detected changes are annotated, these regions may also contain incidental system noise, such as clock refreshes. Therefore, during this region-wise traversal, the VLM further distinguishes whether each described change reflects system noise or a meaningful state transition. Finally, it outputs the set of change descriptions C, categorized into system noise $\mathcal{N}$ and action relevant changes $\mathcal{E}$:

$$(C, \mathcal{N}, \mathcal{E}) = f_{\text{desc}}(S_{pre}^{som}, S_{post}^{som}, A_{ground}). \tag{5}$$

We retain only the descriptions and markers associated with meaningful regional changes while filtering out system noise. This procedure requires re-labeling the original screenshots, S_{pre} and S_{post}, to derive S_{pre}^{clean} and S_{post}^{clean}, which isolate non-noise changes for use in the subsequent success judgment phase. We also filter out noise-related descriptions from C and obtain C^{clean}:

$$(S_{pre}^{clean}, S_{post}^{clean}, C^{clean}) = f_{\text{clean}}(S_{pre}, S_{post}, \mathcal{E}, C). \tag{6}$$

4.3 Action Outcome Verification Based on Evidence

Upon obtaining the filtered evidence, the final stage involves determining action success. At this step, the screenshots S_{pre}^{clean} and S_{post}^{clean}, the filtered change description C^{clean}, and the original intended action A_{intend} are jointly provided to the reflector. Based on this filtered visual evidence, the VLM evaluates whether the action has achieved the intended objective and outputs the reflection R, which consists of a success label, key evidence, and a status summary for downstream modules:

$$R = f_{\text{judge}}(S_{pre}^{clean}, S_{post}^{clean}, C^{clean}, A_{intend}). \tag{7}$$

5 Evaluation

5.1 Evaluation Setup

Benchmarks. We use two benchmarks across diverse environments for computer-based agents: **OSWorld-Verified** [27, 28] and **WindowsAgentArena** [4]. OSWorld-Verified comprises 369 real-world Linux system tasks. In accordance with official guidelines and recent evaluations [18, 24, 27, 32], we exclude 8 Google Drive-related tasks that require manual configuration, resulting

Table 1: Mean success rates (%) over three runs on the OSWorld-Verified benchmark. Rows correspond to task types. For each model backbone, the table reports the mean POR result, the mean EFR result, and their absolute difference Δ in percentage points. The best mean accuracy in each row is bolded; ties are bolded together.

Task Type	GUI-Owl-32B [32]			Qwen3-VL-32B [3]			Kimi-K2.5 [9]			Seed-1.8 [5]		
	POR	EFR	Δ	POR	EFR	Δ	POR	EFR	Δ	POR	EFR	Δ
OS	61.11	62.50	+1.39	56.94	55.56	-1.38	59.72	68.06	+8.34	59.72	**70.83**	+11.11
Office	30.16	33.62	+3.46	23.02	26.72	+3.70	50.97	**61.20**	+10.23	47.23	51.25	+4.02
Daily	50.24	58.02	+7.78	42.03	48.67	+6.64	57.88	**64.50**	+6.62	49.00	58.36	+9.36
Professional	64.63	**80.95**	+16.32	66.67	73.47	+6.80	**80.95**	78.91	-2.04	62.59	63.27	+0.68
Workflow	16.92	24.60	+7.68	15.48	21.51	+6.03	32.50	**41.66**	+9.16	25.32	25.78	+0.46
Overall	37.82	44.91	+7.09	33.36	38.38	+5.02	52.36	**59.74**	+7.38	44.88	49.16	+4.28

Table 2: Mean success rates (%) over three runs on the WindowsAgentArena benchmark. For each model backbone, Δ denotes the mean EFR result minus the mean POR result in percentage points. The best mean accuracy in each row is bolded; ties are bolded together.

Task Type	GUI-Owl-32B [32]			Qwen3-VL-32B [3]			Kimi-K2.5 [9]			Seed-1.8 [5]		
	POR	EFR	Δ	POR	EFR	Δ	POR	EFR	Δ	POR	EFR	Δ
Office	5.43	8.66	+3.23	6.98	6.98	+0.00	16.28	18.60	+2.32	36.43	**37.02**	+0.59
Web Browsing	21.33	32.76	+11.43	30.00	43.22	+13.22	44.11	**56.44**	+12.33	47.44	49.78	+2.34
Windows System	48.61	44.44	-4.17	47.22	51.39	+4.17	73.61	79.17	+5.56	**83.14**	80.56	-2.58
Coding	58.33	75.00	+16.67	66.67	65.28	-1.39	73.47	**81.81**	+8.34	58.33	67.92	+9.59
Media & Video	39.63	34.92	-4.71	34.92	48.52	+13.60	59.35	**61.30**	+1.95	24.79	26.40	+1.61
Windows Utilities	16.67	25.00	+8.33	16.67	30.56	+13.89	41.67	41.67	+0.00	42.53	**58.33**	+15.80
Overall	29.04	34.12	+5.08	31.60	37.55	+5.95	47.40	**52.88**	+5.48	48.16	51.32	+3.16

in a 361-task subset. This evaluation spans 10 application domains, including Chrome, GIMP, LibreOffice, Thunderbird, VLC, and VSCode. WindowsAgentArena consists of 154 Windows system tasks encompassing various applications; we include the complete task set for our analysis. Appendix D shows the task type. The screen resolution of these two benchmarks are different, which are 1920×1080 and 1280×800, respectively.

Baselines and Models. We adopt the conventional POR framework as our baseline, adhering to the implementation established in Mobile-Agent-v3 [32]. Our method replaces the standard reflector module with the proposed EFR variant. To ensure a comprehensive evaluation across diverse architectures, we select both GUI-specialized and general-purpose backbones from open-source and proprietary families. Specifically, we benchmark both POR and POR-EFR using four models: GUI-Owl-32B [32], Qwen3-vl-32B-instruct [3], Kimi-K2.5 [9], and Seed-1.8 [5].

Configurations. All experiments are conducted within the Docker environments provided by the respective benchmarks. The agent operates on raw, screenshot-only observations without further preprocessing or structured UI metadata; all actions are executed via the `pyautogui` library, for details see Appendix B. We set the maximum number of steps to 50 for GUI-specialized and proprietary models, while open-source general-purpose models are restricted to a maximum of 15 steps. We repeat each main POR and POR-EFR experiment three times and report the mean success rate across 3 runs in Tables 1 and 2.

5.2 Main Results and Analysis

Comparison with Baseline. On OSWorld-Verified, Table 1 shows that replacing the standard reflector in POR with EFR consistently improves the mean overall success rate across all four backbones, with an average gain of 5.94%. The strongest EFR setting, POR-EFR with Kimi-K2.5, achieves the best mean overall score of 59.74%, improving over its matched POR counterpart by 7.38%. On Win-

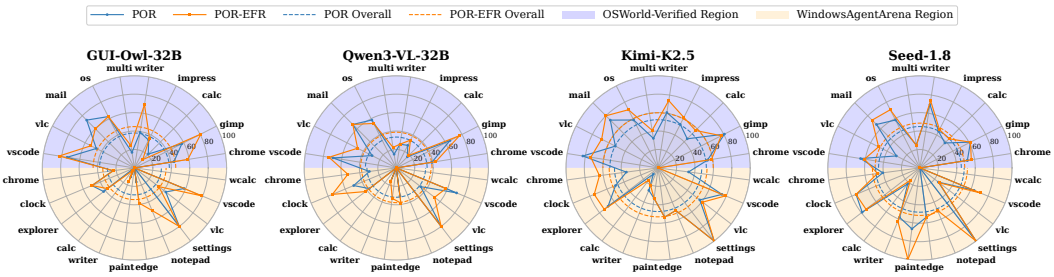


Figure 6: Per-app success-rate comparison between POR and POR-EFR on OSWorld-Verified and WindowsAgentArena. Each subplot corresponds to one backbone model.

dowsAgentArena, Table 2 similarly shows consistent mean improvements across all four backbones, yielding an average gain of 4.92%. POR-EFR with Kimi-K2.5 achieves the best mean overall score of 52.88%, improving over its matched POR counterpart by 5.48%. Across the three runs, the standard deviations of the overall success rates range from 0.17% to 1.13%. Specifically, the POR/EFR standard deviations for GUI-Owl-32B, Qwen3-VL-32B, Kimi-K2.5, and Seed-1.8 are 0.26/0.17, 0.76/0.90, 0.25/0.17, and 0.42/0.56 on OSWorld-Verified, and 0.41/0.63, 0.68/1.13, 0.18/0.25, and 0.54/0.18 on WindowsAgentArena, respectively. These deviations are small relative to the observed overall gains, indicating that the improvements are stable across runs. The corresponding per-app results for the originally reported run are shown in Figure 6.

Analysis over Different Task Categories. At the category level, the three-run means show that EFR provides the clearest gains on several task categories rather than uniformly across all of them. On OSWorld-Verified, Kimi-K2.5 improves by 10.23% and 9.16% on "Office" and "Workflow" tasks, reaching mean success rates of 61.20% and 41.66%, respectively. On WindowsAgentArena, Kimi-K2.5 improves by 12.33% on "Web Browsing", reaching 56.44%, while Seed-1.8 improves by 15.80% on "Windows Utilities", reaching 58.33%. These tasks often involve dense interfaces and frequent state transitions. EFR is particularly beneficial in such settings because it makes page changes more explicit and helps the agent deal with complex situations. We also observe a few category-level decreases compared with the conventional reflector. These cases are scattered across task categories and model backbones, suggesting that they do not indicate a systematic degradation on a specific task type. We provide representative failure examples in the Appendix H and summarize their causes in Appendix H.4.

Analysis over Different Model Backbones. EFR brings consistent mean overall gains across all backbones. On OSWorld-Verified, the improvements are 7.09%, 5.02%, 7.38%, and 4.28% for GUI-Owl-32B, Qwen3-VL-32B, Kimi-K2.5, and Seed-1.8, respectively. On WindowsAgentArena, the corresponding gains are 5.08%, 5.95%, 5.48%, and 3.16%. The gains are not determined solely by the capability of the backbone, nor do stronger models make EFR unnecessary. For example, Kimi-K2.5 obtains a larger improvement than GUI-Owl-32B on OSWorld-Verified and a comparable improvement on WindowsAgentArena, indicating that even strong models still benefit from explicit page-difference evidence and serial reflection.

5.3 Impact of Reflector Judgment on Task Success

To analyze the relationship between reflector accuracy and overall task success, we manually annotated the correctness of the reflectors judgments for each step within the GUI-Owl-32B trajectories on the OSWorld-Verified benchmark. The result is provided in Appendix Figure 11. Our EFR framework improves the reflectors judgment accuracy by 7.11%. A Pearson correlation coefficient of 0.86 indicates a strong positive correlation between reflector accuracy and end-to-end task success across various task categories. These findings suggest that a more reliable reflector enables the agent to evaluate execution outcomes more precisely, thereby enhancing overall performance.

5.4 Ablation Study

We conduct an ablation study on the OSWorld-Verified benchmark by using GUI-Owl-32B model.

Table 3: Ablation results of GUI-Owl-32B on OSWorld-Verified.

Setting	Overall (%)	Drop (%)
POR baseline (w/o both)	37.86	-6.97
Only Two-stage Judging (w/o SoM)	42.11	-2.72
Only SoM (w/o Two-stage Judging)	38.13	-6.70
Diff. Identification w/ Intended Action	43.45	-1.38
Outcome Verification w/o Screenshots	34.25	-10.58
Full method (SoM + Two-stage Judging)	**44.83**	–

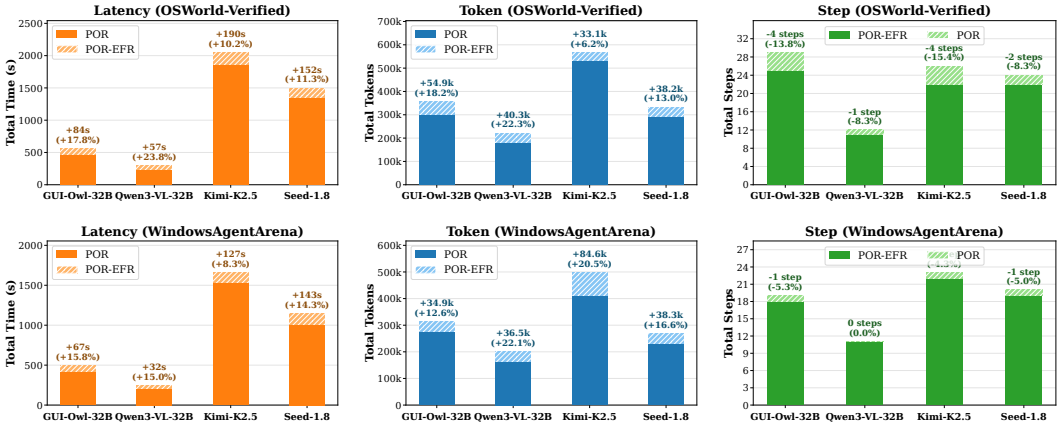


Figure 7: Average per-task cost comparison between POR and POR-EFR on OSWorld-Verified and WindowsAgentArena. The first row reports OSWorld-Verified and the second row reports WindowsAgentArena. From left to right, each row shows latency, token usage, and executed steps.

Effect of SoM. Removing the SoM module requires the reflector to directly analyze the page differences and subsequently evaluate action success based on global difference descriptions. This configuration results in a 2.72% decrease in the success rate, demonstrating that SoM facilitates the difference analysis phase by providing localized focus.

Effect of Page Difference Identification. Ablating the page difference identification module, whereby the reflector evaluates success directly after receiving the marked screenshots results in a significant 6.70% decrease in the success rate. This drop underscores the critical role of the page difference identification process within our framework. The explicit separation of difference analysis and success judgment is essential for realizing the full benefits of the marked visual evidence.

Replace Grounded with Intended Action in Page Difference Identification Stage. We further replace the grounded action with the original intended action during the page difference identification stage. This variant leads to a 1.38% drop in accuracy. Since page difference identification aims to describe what actually changes after execution, conditioning this process on the intended action may cause the VLM to focus on expected outcomes rather than the observed screen transition.

Action Outcome Verification without Screenshots. We further evaluate a variant in which the action outcome verification stage relies only on the change descriptions, without access to screenshots. This setting results in a 10.58% drop in accuracy. This degradation suggests that textual change descriptions alone are insufficient for reliable outcome verification in complex GUI environments. When the interface is dense and contains multiple changes, the verifier may perceive only scattered changes, rather than forming a holistic understanding of the page structure.

5.5 Cost Analysis

Figure 7 compares the average per-task cost of POR and POR-EFR on two benchmarks. Averaged over the four backbone models, POR-EFR increases per-task latency by 15.8% and token usage by 14.9% on OSWorld-Verified, and by 13.3% and 17.9% on WindowsAgentArena, respectively.

Table 4: Accuracy and cost of Claude Opus 4.7 on OSWorld-Verified with different reflector thinking-effort settings.

Metric	POR (`xhigh`)	EFR (`medium`)	Difference
Tokens	1,072,140	**820,316**	-23.49%
Time (s)	2,781	**2,534**	-8.88%
Accuracy (%)	77.80	**79.84**	$+2.04\%$

This overhead mainly stems from the additional VLM call introduced for page-difference analysis, which increases the cost of each reflection step. Although EFR reduces redundant execution by lowering the average number of executed steps by 11.5% on OSWorld-Verified and 3.7% on WindowsAgentArena, this reduction only partially offsets the increased per-step reflection cost. Detailed per-step latency and token costs are provided in Appendix G.

5.6 EFR on Frontier Models: Beyond Increased Thinking Effort

To assess whether EFR's gains persist with a stronger backbone and are not merely attributable to the additional computation introduced by its extra reflection steps, we evaluate Claude Opus 4.7 on OSWorld-Verified. We assign `xhigh` thinking effort to the conventional POR reflector and only `medium` effort to the EFR reflector. As shown in Table 4, EFR achieves higher accuracy despite its lower thinking-effort setting, improving accuracy by 2.04% while reducing token usage by 23.49% and execution time by 8.88%. Even with a frontier backbone, EFR achieves a better accuracy-cost trade-off than the conventional reflector operating at a higher thinking effort setting. These results suggest that EFRs gains stem from explicitly separating visual evidence extraction from subsequent outcome verification, rather than merely increasing test-time computation. Details are provided in Appendix E.

6 Conclusion

We revisit the role of the reflector in GUI agents and contend that its complexity and functionality are frequently underestimated in desktop environments. We thus introduce EFR, a two-stage reflection framework that decouples explicit visual evidence extraction from subsequent success judgment. Empirical evaluations demonstrate its superior performance. More broadly, our findings highlight that building GUI agents requires more than directly applying existing methods. While POR and SoM provide methodological foundations, their effectiveness depends on how well they are adapted to the perceptual, spatial, and interactive properties of GUI tasks.

Acknowledgments and Disclosure of Funding

This work was supported in part by China NSF grant (No. 62432007, No. 62572299, No. 62441236, No. 62372296, No. U24A20326, No. U25A6024, No. U25A20437), the Key Research and Development Program of Zhejiang Province (No. 2024C03270), CCF-Tencent Rhino-Bird Open Research Fund (No. RAGR20260126), Alibaba Innovation Research (AIR) Program (No. 56657574-1), and SJTU-Huawei Explore X Gift Fund. Chaoyue Niu and Fan Wu are the corresponding authors.

References

[1] Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S: An open agentic framework that uses computers like a human. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=lIVRgt4nLv.

[2] Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S2: A compositional generalist-specialist framework for computer use agents. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=zg5is4GJ3R.

[3] Shuai Bai, Yuxuan Cai, Ruizhe Chen, et al. Qwen3-VL Technical Report. *arXiv preprint arXiv:2511.21631*, 2025. doi: 10.48550/arXiv.2511.21631. URL https://arxiv.org/abs/2511.21631.

[4] 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. Windows agent arena: Evaluating multi-modal OS agents at scale. In *Proceedings of the 42nd International Conference on Machine Learning*, pages 4874–4910. PMLR, 2025. URL https://proceedings.mlr.press/v267/bonatti25a.html.

[5] ByteDance Seed. Seed1.8 model card: Towards generalized real-world agency, 2025. URL https://lf3-static.bytednsdoc.com/obj/eden-cn/lapzild-tss/ljhwZthlaukjlkulzlp/research/Seed-1.8-Modelcard.pdf.

[6] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.505. URL https://aclanthology.org/2024.acl-long.505/.

[7] Yu Du, Fangyun Wei, and Hongyang Zhang. Anytool: Self-reflective, hierarchical agents for large-scale API calls. In *Proceedings of the 41st International Conference on Machine Learning*, pages 11812–11829. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24h.html.

[8] Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, Hengxu Wang, Luowei Zhou, and Mike Zheng Shou. AssistGUI: Task-oriented desktop graphical user interface automation. *arXiv preprint arXiv:2312.13108*, 2023. doi: 10.48550/arXiv.2312.13108. URL https://arxiv.org/abs/2312.13108.

[9] Kimi. Kimi k2.5: Visual agentic intelligence, 2026. URL https://www.kimi.com/blog/kimi-k2-5.

[10] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.50. URL https://aclanthology.org/2024.acl-long.50/.

[11] Ning Li, Xiangmou Qu, Jiamu Zhou, Jun Wang, Muning Wen, Kounianhua Du, Xingyu Lou, Qiuying Peng, Jun Wang, and Weinan Zhang. Mobileuse: A GUI agent with hierarchical reflection for autonomous mobile operation. *arXiv preprint arXiv:2507.16853*, 2025. doi: 10.48550/arXiv.2507.16853. URL https://arxiv.org/abs/2507.16853.

[12] Haowei Liu, Xi Zhang, Haiyang Xu, Yuyang Wanyan, Junyang Wang, Ming Yan, Ji Zhang, Chunfeng Yuan, Changsheng Xu, Weiming Hu, and Fei Huang. PC-Agent: A hierarchical multi-agent collaboration framework for complex task automation on PC. *arXiv preprint arXiv:2502.14282*, 2025. doi: 10.48550/arXiv.2502.14282. URL https://arxiv.org/abs/2502.14282.

[13] Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. InfiGUIAgent: A multimodal generalist GUI agent with native reasoning and reflection. *arXiv preprint arXiv:2501.04575*, 2025. doi: 10.48550/arXiv.2501.04575. URL https://arxiv.org/abs/2501.04575.

[14] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. OmniParser for pure vision based GUI agent. *arXiv preprint arXiv:2408.00203*, 2024. doi: 10.48550/arXiv.2408.00203. URL https://arxiv.org/abs/2408.00203.

[15] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

[16] Hongze Mi, Yibo Feng, Wenjie Lu, Yuqi Wang, Jinyuan Li, Song Cao, He Cui, Tengfei Tian, Xuelin Zhang, Haotian Luo, Di Sun, Jun Fang, Hua Chai, Naiqiang Tan, and Gang Pan. D-Artemis: A deliberative cognitive framework for mobile GUI multi-agents. *arXiv preprint arXiv:2509.21799*, 2025. doi: 10.48550/arXiv.2509.21799. URL https://arxiv.org/abs/2509.21799.

[17] Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. ScreenAgent: A vision language model-driven computer control agent. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 6433–6441. International Joint Conferences on Artificial Intelligence Organization, 2024. doi: 10.24963/ijcai.2024/711. URL https://doi.org/10.24963/ijcai.2024/711.

[18] 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. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025. doi: 10.48550/arXiv.2501.12326. URL https://arxiv.org/abs/2501.12326.

[19] 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. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=il5yUQsrjC.

[20] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

[21] Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-Agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. In *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-0088. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/0520537ba799d375b8ff5523295c337a-Abstract-Conference.html.

[22] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-Agent: Autonomous multi-modal mobile device agent with visual perception. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. URL https://openreview.net/forum?id=jE6pDYCnVF.

[23] Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, Bin Wang, Chuhan Wu, Yasheng Wang, Ruiming Tang, and Jianye Hao. GUI agents with foundation models: A comprehensive survey. *arXiv preprint arXiv:2411.04890*, 2024. doi: 10.48550/arXiv.2411.04890. URL https://arxiv.org/abs/2411.04890.

[24] Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, et al. OpenCUA: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025. doi: 10.48550/arXiv.2508.09123. URL https://arxiv.org/abs/2508.09123.

[25] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-Agent-E: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025. doi: 10.48550/arXiv.2501.11733. URL https://arxiv.org/abs/2501.11733.

[26] Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. BacktrackAgent: Enhancing GUI agent with error detection and backtracking mechanism. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4250–4272, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.212. URL https://aclanthology.org/2025.emnlp-main.212/.

[27] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html.

[28] Tianbao Xie, Mengqi Yuan, Danyang Zhang, Xinzhuang Xiong, Zhennan Shen, Zilong Zhou, Xinyuan Wang, Yanxu Chen, Jiaqi Deng, Junda Chen, Bowen Wang, Haoyuan Wu, Jixuan Chen, Junli Wang, Dunjie Lu, Hao Hu, and Tao Yu. Introducing OSWorld-Verified. *xlang.ai*, July 2025. URL https://xlang.ai/blog/osworld-verified.

[29] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous GUI interaction. In *Proceedings of the 42nd International Conference on Machine Learning*, pages 69772–69805. PMLR, 2025. URL https://proceedings.mlr.press/v267/xu25ae.html.

[30] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. *arXiv preprint arXiv:2310.11441*, 2023. doi: 10.48550/arXiv.2310.11441. URL https://arxiv.org/abs/2310.11441.

[31] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

[32] Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, Jitong Liao, Qi Zheng, Fei Huang, Jingren Zhou, and Ming Yan. Mobile-Agent-v3: Fundamental agents for GUI automation. *arXiv preprint arXiv:2508.15144*, 2025. doi: 10.48550/arXiv.2508.15144. URL https://arxiv.org/abs/2508.15144.

[33] Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. UFO: A UI-focused agent for windows OS interaction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 597–622, Albuquerque, New Mexico, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.26. URL https://aclanthology.org/2025.naacl-long.26/.

[34] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. AppAgent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–20, New York, NY, USA, 2025. Association for Computing Machinery. doi: 10.1145/3706598.3713600. URL https://doi.org/10.1145/3706598.3713600.

Appendix Contents

A Set-of-Marks Annotation Details **16**

A.1 Visual Change Area Annotation 16

A.2 Action Location Annotation 16

A.3 Behavior under extreme visual changes. 17

B Action Space **19**

C Step-Level Reflection Example **19**

C.1 Conventional Reflector 20

C.2 EFR 20

D Benchmark Task Grouping Details **21**

E Evaluation with Claude Opus 4.7 on OSWorld-Verified **21**

F Reflector Reliability Analysis **22**

G Per-Step Cost Analysis **23**

H Case Study **25**

H.1 Failure to Success 25

H.2 Failure to Failure 28

H.3 Success to Failure 30

H.4 EFR Failure Case Analysis 32

I Complete EFR Prompt **33**

A Set-of-Marks Annotation Details

A.1 Visual Change Area Annotation

Algorithm 1 summarizes the visual difference annotation procedure used in EFR. The algorithm takes a pair of pre- and post-action screenshots and outputs two consistently annotated images together with the merged difference boxes. At this stage, the goal is only to identify *where* visual changes occur; the action-location annotation step is described in the next subsection. All visual-change markers use red bounding boxes, with each box assigned a unique index so that the reflector can refer to the same changed region across the pre- and post-action screenshots.

Algorithm 1: Visual Difference Region Annotation

Input: Pre-action screenshot S_{pre}, post-action screenshot S_{post}
Output: Annotated screenshots $\hat{S}_{pre}, \hat{S}_{post}$ and merged box set $\mathcal{B}$

function SHOULDMERGE(b_i, b_j, M)
    Compute the union box u of b_i and b_j
    Compute the x/y gaps between b_i and b_j
    Compute the difference density of u on binary mask M
    if b_i and b_j are tightly adjacent **then**
        return density$(u) \geq \tau_{\text{dens}}$
    else
        return False
    end if
end function
Convert S_{pre} and S_{post} to grayscale images
$D \leftarrow$ pixel-wise absolute difference between the two grayscale images
$M \leftarrow$ binary mask obtained by thresholding D
Extract external contours from M and convert each contour to a bounding box
Remove boxes with a size of at most 2 pixels
Remove boxes fully contained in the top status-bar area
Expand each remaining box by 5 pixels on all sides
$\mathcal{B} \leftarrow$ the resulting box set
repeat
    changed $\leftarrow$ False
    for all pairs (b_i, b_j) in $\mathcal{B}$ **do**
        if SHOULDMERGE(b_i, b_j, M) **then**
            Replace (b_i, b_j) with their union in $\mathcal{B}$
            changed $\leftarrow$ True
            break
        end if
    end for
until changed = False
Sort $\mathcal{B}$ from top to bottom and from left to right
Draw the same red bounding boxes and indices in $\mathcal{B}$ on S_{pre} and S_{post}
return annotated images $\hat{S}_{pre}, \hat{S}_{post}$ and $\mathcal{B}$

The output of Algorithm 1 constitutes the candidate visual evidence used by the reflector. However, these boxes alone do not indicate where the current action was actually executed. To connect the detected visual changes with the concrete operation site, we further annotate the action location on the pre-action screenshot in the next subsection.

A.2 Action Location Annotation

Compared with the difference-region identification in Algorithm 1, action-location annotation is a lighter-weight post-processing step. Based on the geometric parameters associated with the actual action A_{ground} executed by the operator, it directly visualizes where the action takes place on the annotated pre-action screenshot $\hat{S}_{pre}$, thereby providing the reflector with a spatial anchor aligned

with the observed visual changes. We denote this process as

$$(\tilde{S}_{pre}, L) = f_{\text{act}}(\hat{S}_{pre}, A_{ground}). \tag{8}$$

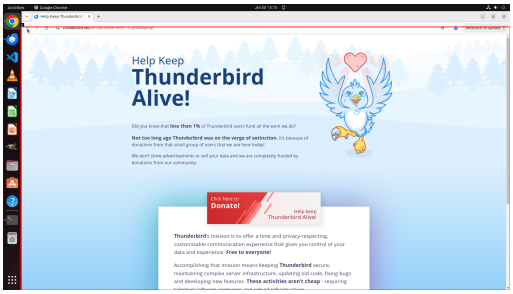
where $\tilde{S}_{pre}$ denotes the pre-action screenshot annotated with the action marker, and L denotes the geometric location information of the action.

In our implementation, actions are divided into three categories in this stage, and the detailed action types are provided in Appendix B. For actions that can be reduced to a single screen point, we directly read the execution coordinates (x, y) and map them onto the pixel grid. We then draw a blue hollow circle centered at that location on S_{pre} to highlight the actual operation site. For the `drag` action, we further parse the drag start point (x_s, y_s) and end point (x_e, y_e) from the executable code. The drag trajectory is visualized with yellow markers: the start point is shown as a yellow hollow circle, the end point is shown as a yellow filled circle, and the two points are connected by a yellow arrow indicating the movement direction. For actions that do not require explicit screen coordinates, such as keyboard shortcuts or waiting actions, no action-location annotation is added. Accordingly, the geometric representations of point-based actions, drag actions, and coordinate-free actions are written as

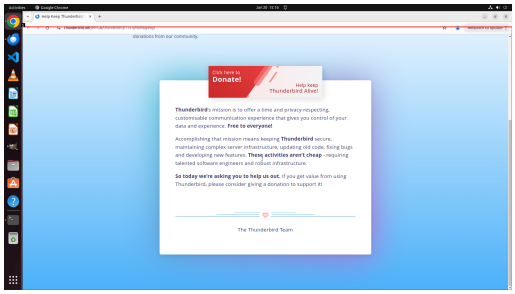
$$L_{\text{point}} = \big((x, y)\big), \qquad L_{\text{drag}} = \big((x_s, y_s), (x_e, y_e)\big), \qquad L_{\text{none}} = \varnothing. \tag{9}$$

This action annotation step does not involve any additional visual inference; it merely projects the parameters of the action already executed by the operator back into the screenshot space. Its purpose is to explicitly indicate where the actual operation took place, thereby assisting the reflector in its observation. Finally, the action-location marker and the candidate difference boxes together form S^{som}_{pre} in the main text, while the post-action screenshot retains only the difference boxes to form S^{som}_{post}.

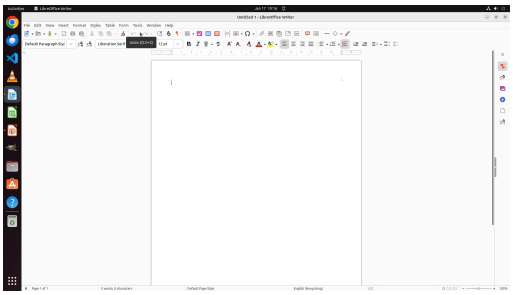
A.3 Behavior under extreme visual changes.



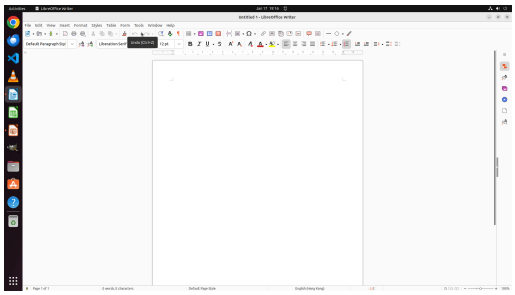
(a) Thunderbird webpage before scrolling.



(b) Thunderbird webpage after scrolling.



(c) Writer document before `Ctrl+V`.



(d) Writer document after `Ctrl+V`.

Figure 8: Visual evidence under extreme changes. Top: a single red box covers nearly the entire Thunderbird webpage content area; upward content displacement and newly visible lower-page content support the expected successful-scroll judgment. Bottom: no difference boxes are detected after `Ctrl+V`; the unchanged blank Writer document supports the expected failed-paste judgment.

Figure 8 presents two concrete cases in which the difference annotation covers almost the entire webpage or contains no boxes. We use these screenshot pairs to work through the expected reasoning

of EFR, specifying what the model identifies in the evidence-description stage and how this evidence determines the subsequent outcome judgment. The analysis traces each judgment to specific visual evidence and the expected effect of the action.

Large-scale changes: scrolling the Thunderbird help webpage. The executed action is `scroll(-5)`, intended to reveal content further down the webpage. As shown in Figures 8a and 8b, a single red box covers 88.9% of the screenshot, encompassing almost the entire webpage content area. In the trajectory, the model observes entire screenshot changes and identifies that the donation panel has moved upward, while additional text and the panel's lower portion have entered the viewport. It retains the region as relevant to the executed scroll and summarizes the evidence as an upward displacement of page content with previously hidden lower content becoming visible. The resulting judgment is A (successful or partially successful). Here, the broad red box directs the evidence analysis to the overall webpage transition, and the first-stage description establishes the concrete content movement that justifies the success judgment without requiring separate boxes for individual page elements.

No detected change: pasting into an empty Writer document. The executed action is `Ctrl+V`, intended to paste copied title text into the active LibreOffice Writer document. In Figures 8c and 8d, there is no change between the screenshots. In the trajectory, the model records that no action-relevant visual changes are detected and that the document body remains blank, with no new text appearing. Since the intended changes are copied text appears in the document, it concludes that the expected paste result has not been achieved and assigns C (failed). Thus, the absence of red boxes contributes concrete evidence to the judgment: the missing content change contradicts the visible effect required by this action.

B Action Space

To remain consistent with the underlying execution interface and to avoid introducing additional confounding factors into the reflector analysis due to differences in low-level action API design, we adopt the action space definition of Mobile-Agent-v3 [32]. Specifically, the operator first outputs a structured action. If the action contains an element description, a grounding model is first invoked to resolve the corresponding element into screen coordinates, after which the grounded result is compiled into executable desktop-operation code. Table 5 summarizes the action types used in this work, their semantic meanings, and the corresponding executable code templates. For readability, we omit the shared `import pyautogui` prefix in the table and retain only the core execution statements.

Table 5: Action space directly adopted from Mobile-Agent-v3 [32]. The code column shows the executable template after grounding. Shared import prefixes are omitted for brevity.

Action Type	**Action Details**	
	Description	**Executable Code Template**
`click`	Click at a grounded screen location.	`pyautogui.click(x=x, y=y)`
`double_click`	Double-click at a grounded screen location.	`pyautogui.doubleClick(x=x, y=y)`
`right_click`	Right-click at a grounded screen location.	`pyautogui.rightClick(x=x, y=y)`

`type`	Focus an input region and type text, optionally clearing the field or pressing Enter.	`pyautogui.doubleClick(x=x, y=y);`
		`hotkey("ctrl", "a"); press("delete");`
		`pyautogui.typewrite(text, interval=1.0);`
		`pyautogui.press("enter")`

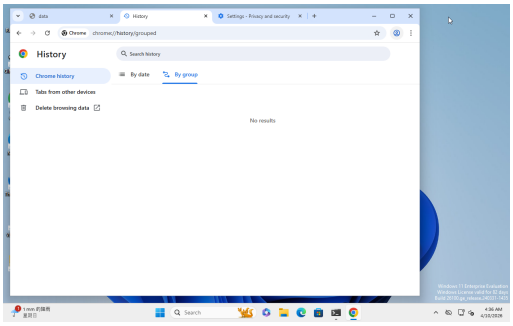
`hotkey`	Execute a keyboard shortcut combination.	`pyautogui.hotkey(key1, ..., keyn)`
`scroll`	Move to a grounded location and scroll by a signed amount.	`pyautogui.moveTo(x, y);` `pyautogui.scroll(value)`
`wait`	Pause execution for a specified duration.	`time.sleep(t)`
`done`	Mark the current task as completed.	`DONE`

`drag`	Drag from one grounded location to another grounded location.	`pyautogui.moveTo(x1, y1);`
		`pyautogui.dragTo(x2, y2, duration=1.0);`
		`pyautogui.mouseUp()`

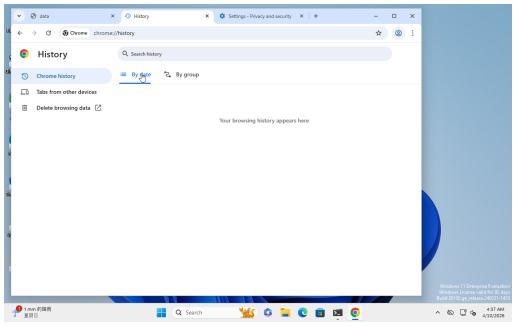
In terms of grounding, `click`, `double_click`, `right_click`, `type`, and `scroll` each require resolving only a single screen point; `drag` requires separately resolving both the start point and the end point; whereas `hotkey`, `wait`, and `done` do not depend on visual grounding. This also explains why the action-location annotation in the previous subsection only needs to distinguish between three cases.

C Step-Level Reflection Example

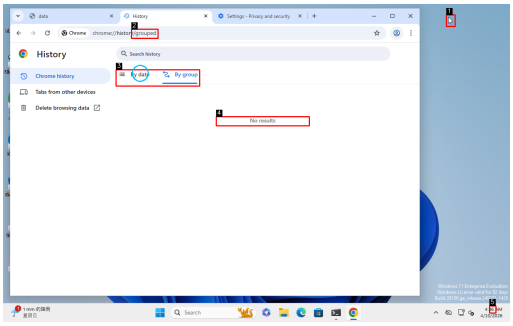
We provide a concrete one-step example to illustrate the difference between the conventional reflector and EFR. The intended action is `Click the "By date" button in Chrome History to sort the history view by date`, while the actually executed grounded action is `Click at coordinate (519,232)`. This is a simple example in which both reflectors make the correct judgment. The raw screenshots and EFR intermediate screenshots are shown in Figure 9. The conventional reflector directly reasons over the raw pre-action and post-action screenshots, whereas EFR generates explicit intermediate evidence before making the final reflection judgment.



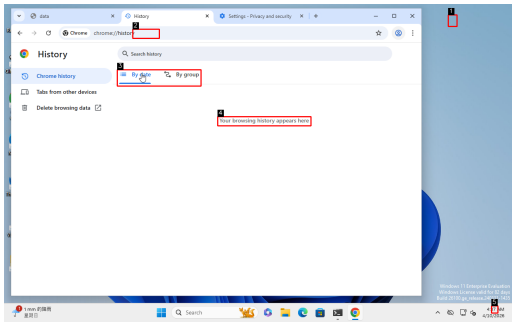
(a) Raw pre-action screenshot



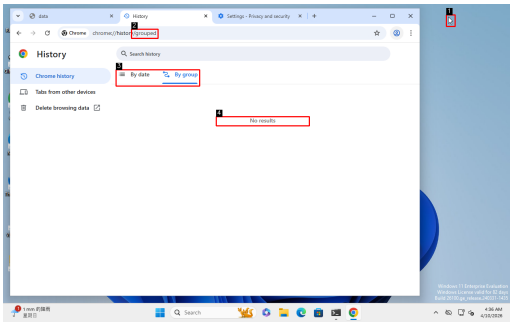
(b) Raw post-action screenshot



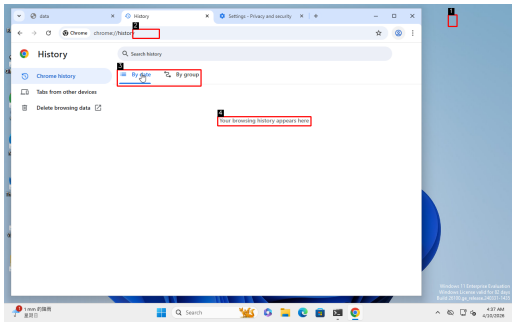
(c) SoM-marked pre-action screenshot



(d) SoM-marked post-action screenshot



(e) Clean evidence before action



(f) Clean evidence after action

Figure 9: A one-step reflection example for clicking the "By date" button in Chrome History. The clean evidence keeps Areas 1–4 and removes Area 5, which only reflects the taskbar clock update.

C.1 Conventional Reflector

- **Step 1: One-step action verification.**
  VLM input: the intended action: `Click the "By date" button in Chrome History to sort the history view by date`. The raw pre-action and post-action screenshots: Figure 9a and Figure 9b.
  VLM output: The observed screen changes from the grouped history view to the date-based history view, indicate that the action succeeded.

C.2 EFR

- **Step 1: SoM annotation.**
  SoM module input: The raw pre-action and post-action screenshots: Figure 9a and Figure 9b. The grounded action: `Click at coordinate (519,232)`.
  SoM module output: The marked pre-action screenshot: Figure 9c, page change regions and action position are marked. The marked post-action screenshot: Figure 9d, only page change regions are marked.

- **Step 2: Page-difference identification.**

  VLM input: the SoM-marked pair in Figures 9c and 9d, plus the grounded action `Click at coordinate (519,232)`.

  VLM output: a structured evidence description: Area 1 shows that the mouse cursor moves away; Area 2 shows that the URL changes from `chrome://history/grouped` to `chrome://history`; Area 3 shows that the selected history mode changes from "By group" to "By date"; Area 4 shows that the center message changes from "No results" to "Your browsing history appears here"; Area 5 is identified as environment noise because it only reflects the taskbar clock update. The clean evidence output is Figure 9e and Figure 9f, which retain Areas 1–4 and remove Area 5.

- **Step 3: Evidence-based action verification.**

  VLM input: the intended action `Click the "By date" button in Chrome History to sort the history view by date`, the structured evidence from Step 2, and the clean evidence screenshots in Figures 9e and 9f.

  Actual output: The action succeeds because Areas 2–4 show the page changing from the grouped history view to the date-based history view, which matches the intended action.

D Benchmark Task Grouping Details

We summarize the correspondence between the app-level categories and the domain-level categories used in Tables 1 and 2. The left table reports OSWorld-Verified and the right table reports WindowsAgentArena. The category mappings follow the official benchmark definitions [4, 27].

Table 6: Correspondence between the original app-level partition and the merged task categories used in the main results.

(a) OSWorld-Verified category mapping.

Domain	Included Apps	# Tasks	%
OS	OS tasks	24	6.6%
Office	LibreOffice Calc, Impress, Writer	117	32.4%
Daily	Chrome, Thunderbird, VLC	78	21.6%
Professional	VS Code, GIMP	49	13.6%
Workflow	Tasks involving multiple apps	93	25.8%
Total	–	361	100.0%

(b) WindowsAgentArena category mapping.

Domain	Included Apps	# Tasks	%
Office	LibreOffice Calc, LibreOffice Writer	43	27.9%
Web Browsing	Chrome, Microsoft Edge	30	19.5%
Windows System	File Explorer, Settings	24	15.6%
Coding	VS Code	24	15.6%
Media & Video	VLC	21	13.6%
Windows Utilities	Clock, Paint, Notepad, Windows Calculator	12	7.8%
Total	–	154	100.0%

E Evaluation with Claude Opus 4.7 on OSWorld-Verified

We extend the evaluation to Claude Opus 4.7 on OSWorld-Verified. We compare the conventional POR reflector with `xhigh` thinking effort against EFR with `medium` thinking effort. This comparison examines whether EFR can achieve a favorable accuracy-cost trade-off with a stronger backbone. Table 7 and Figure 10 show the results.

Table 7: Success rates (%) of Claude Opus 4.7 on OSWorld-Verified. Rows correspond to task types. The table reports the POR result with `xhigh` thinking effort, the EFR result with `medium` thinking effort, and their difference Δ (EFR minus POR) in percentage points. The best success rate in each row is bolded; ties are bolded together.

Task Type	**Claude Opus 4.7**		
	POR (`xhigh`)	**EFR** (`medium`)	Δ
OS	**95.83**	**95.83**	0.00
Office	83.73	**86.29**	+2.56
Daily	72.72	**74.18**	+1.46
Professional	**83.67**	**83.67**	0.00
Workflow	66.84	**70.33**	+3.49
Overall	77.80	**79.84**	+2.04

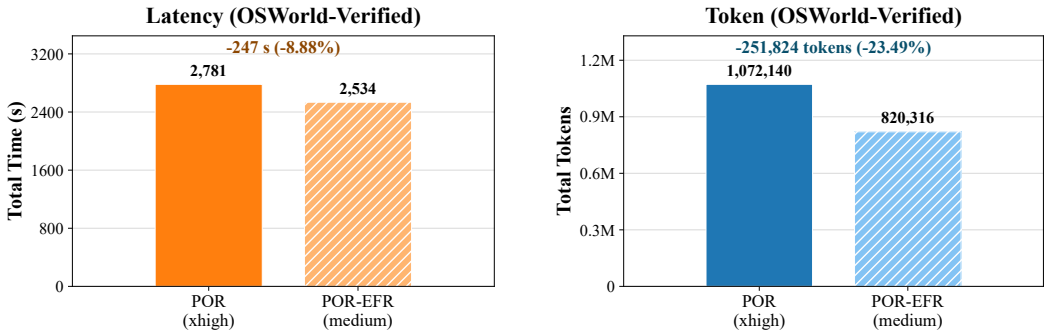


Figure 10: Average per-task cost comparison between POR (`xhigh`) and POR-EFR (`medium`) with Claude Opus 4.7 on OSWorld-Verified. The left panel shows latency and the right panel shows token usage. Annotations report absolute and relative changes from POR to POR-EFR.

POR-EFR increases the overall success rate from 77.80% to 79.84%, an absolute gain of 2.04%, despite using a lower thinking effort setting. The largest improvement occurs on Workflow tasks (+3.49%), followed by Office (+2.56%) and Daily (+1.46%), while success rates on OS and Professional tasks remain unchanged, respectively. These gains are also accompanied by lower execution costs: average per-task latency decreases from 2,781 s to 2,534 s, a reduction of 247 s (8.88%), while token usage decreases from 1,072,140 to 820,316, saving 251,824 tokens (23.49%). These results suggest that EFR remains effective with a strong backbone and that its gains are not merely attributable to increased test-time computation.

F Reflector Reliability Analysis

We provide two manually annotated analyses based on GUI-Owl-32B trajectories on the OSWorld-Verified benchmark. First, Figure 11 compares reflector judgment accuracy with end-to-end task success across five categories. Overall, replacing the conventional reflector with EFR increases average reflector accuracy from 80.39% to 87.50%, while average task success rises from 38.23% to 44.88%. The trend is particularly clear on Professional tasks, where reflector accuracy improves by 9.88% and task success increases by 16.32%. Across all method-category points in the figure, reflector accuracy and task success have a Pearson correlation of 0.86, suggesting that more reliable intermediate verification is strongly associated with higher final task completion.

Second, Table 8 reports how often failed trajectories can be attributed to reflector errors under the conventional POR and EFR framework. EFR reduces reflector-attributed failures from 61 of 223 failed trajectories (27.35%) to 27 of 199 (13.57%), a decrease of 13.78%. More specifically, page-difference identification errors decrease from 43 of 223 failures (19.28%) to 14 of 199 (7.04%), a

12.25% reduction, while success-judgment errors decrease from 18 of 223 (8.07%) to 13 of 199 (6.53%), a 1.54% reduction. Thus, EFR reduces both error types, with the larger improvement arising from more reliable page-difference identification.

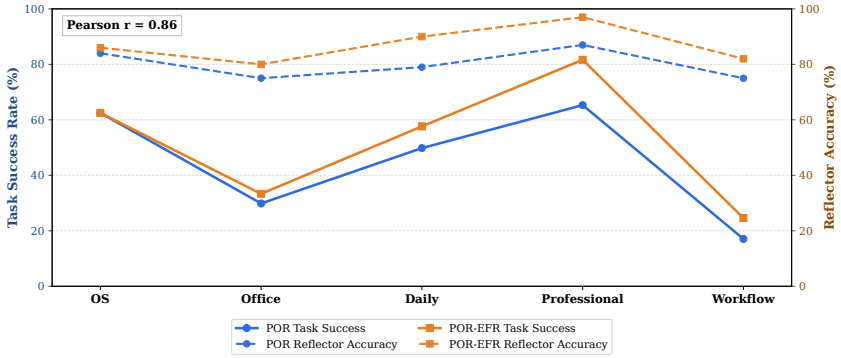


Figure 11: Task success rate vs. reflector success rate in GUI-Owl-32B trajectories on the OSWorld-Verified benchmark.

Table 8: Manual failure attribution for GUI-Owl-32B under the conventional POR framework and EFR on OSWorld-Verified, using the same model and mutually exclusive error categories. "Diff. Error" denotes page-difference identification errors, "Judge. Error" denotes success-judgment errors given correctly identified page differences, and "Total" denotes all failed trajectories. For the reflector-error rate, Δ denotes EFR minus POR in percentage points. The lower rate in each row is bolded; ties are bolded together.

Application	**Reflector**		**Diff. Error**		**Judge. Error**		**Total**		**Rate (%)**		
	POR	**EFR**	**POR**	**EFR**	**POR**	**EFR**	**POR**	**EFR**	**POR**	**EFR**	Δ
chrome	3	0	2	0	1	0	26	19	11.54	**0.00**	-11.54
gimp	0	0	0	0	0	0	11	5	**0.00**	**0.00**	+0.00
libreoffice_calc	23	11	16	7	7	4	37	41	62.16	**26.83**	-35.33
libreoffice_impress	12	2	8	0	4	2	31	30	38.71	**6.67**	-32.04
libreoffice_writer	4	1	4	1	0	0	14	7	28.57	**14.29**	-14.28
multi_apps	17	11	12	6	5	5	77	70	22.08	**15.71**	-6.37
os	0	0	0	0	0	0	9	9	**0.00**	**0.00**	+0.00
thunderbird	0	0	0	0	0	0	4	6	**0.00**	**0.00**	+0.00
vlc	1	1	1	0	0	1	9	8	**11.11**	12.50	+1.39
vs_code	1	1	0	0	1	1	5	4	**20.00**	25.00	+5.00
All	61	27	43	14	18	13	223	199	27.35	**13.57**	-13.78

G Per-Step Cost Analysis

Figure 12 reports the average latency and token usage per executed step. POR-EFR increases latency and token usage per step by 30.8% and 29.8% on OSWorld-Verified, and by 17.7% and 22.4% on WindowsAgentArena, respectively. These results show that EFR introduces additional cost at each reflection step, while the step reduction shown in Figure 7 helps keep the final per-task overhead moderate.

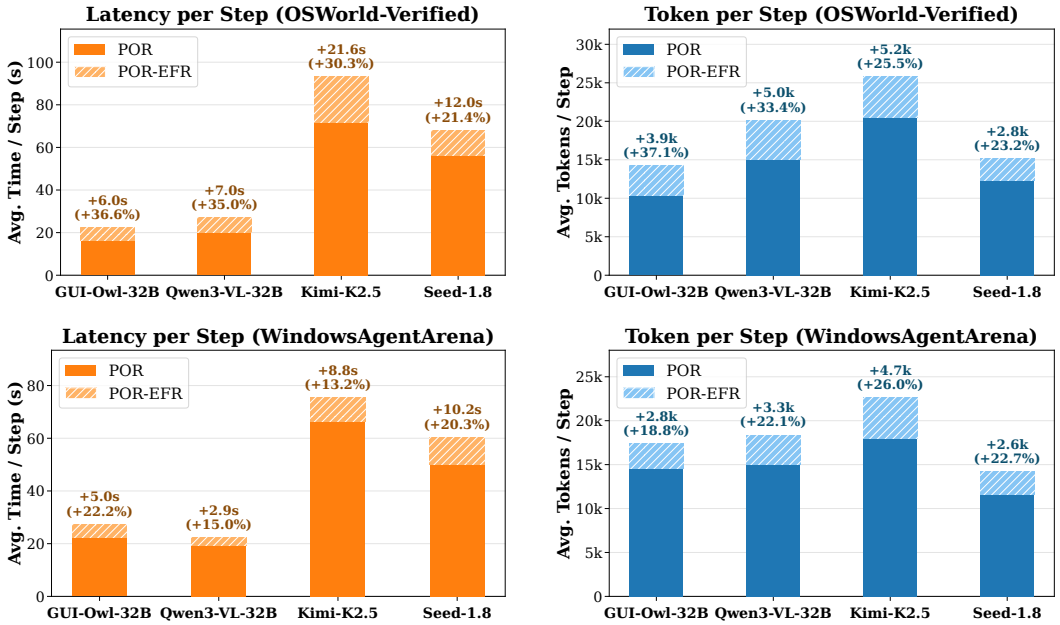


Figure 12: Average per-step cost comparison between POR and POR-EFR on OSWorld-Verified and WindowsAgentArena. The first row reports OSWorld-Verified and the second row reports WindowsAgentArena. From left to right, each row shows latency and token usage averaged over executed steps.

H Case Study

Before examining individual trajectories, we first summarize how EFR changes task-level outcomes relative to the conventional reflector, as illustrated in Figure 13. On OSWorld-Verified, 84.05% of samples remain failure→failure, 12.68% are converted from failure→success, and 3.27% shift from success→failure. On WindowsAgentArena, the corresponding proportions are 85.51%, 10.19%, and 4.30%, respectively. These transition statistics show that EFR improves task success mainly by recovering previously failed samples while introducing only a small fraction of negative flips. At the same time, the large failure→failure proportion indicates that many GUI-agent tasks remain unsolved even after applying our method, highlighting that robust GUI agents still require substantial further progress. We illustrate each of these three types with representative examples.

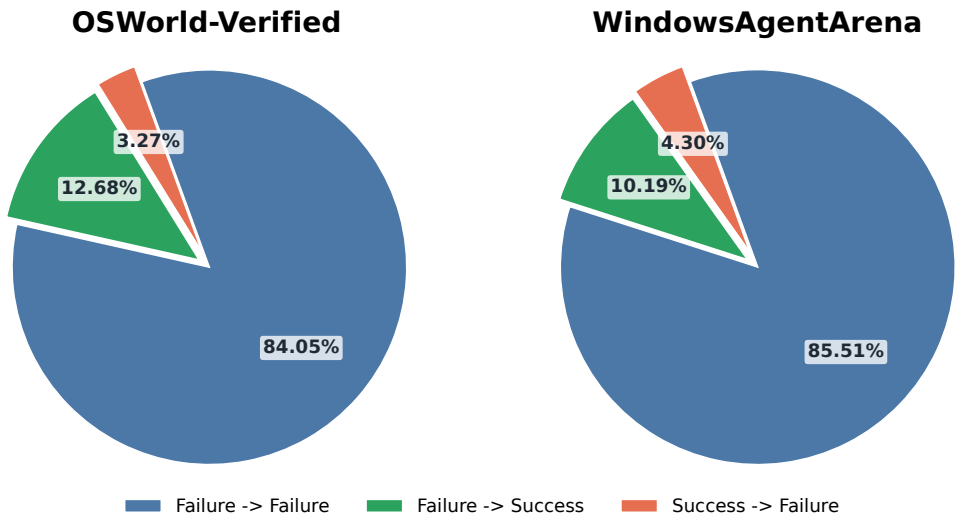


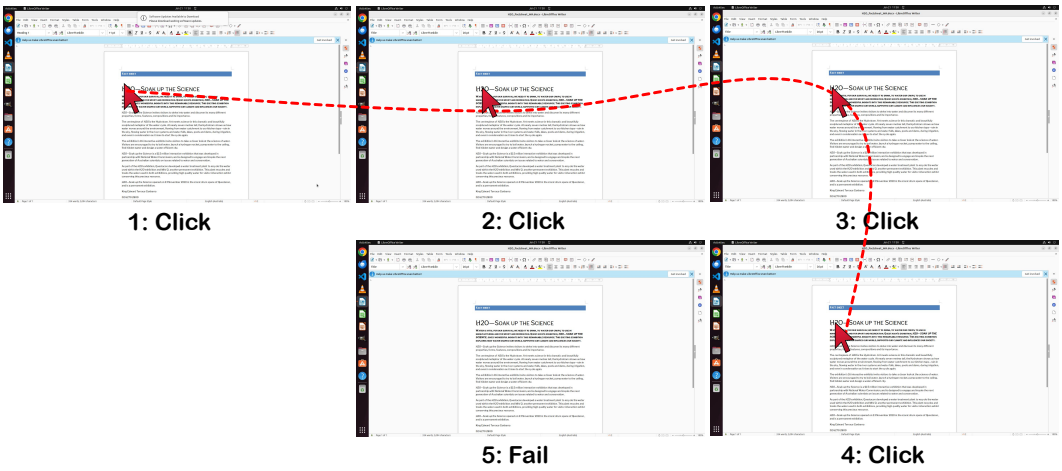
Figure 13: Outcome transition distributions on OSWorld-Verified and WindowsAgentArena, except success to success. Each pie chart shows the proportion of trajectories that remain failed, are corrected from failure to success, or regress from success to failure after applying our method.

H.1 Failure to Success

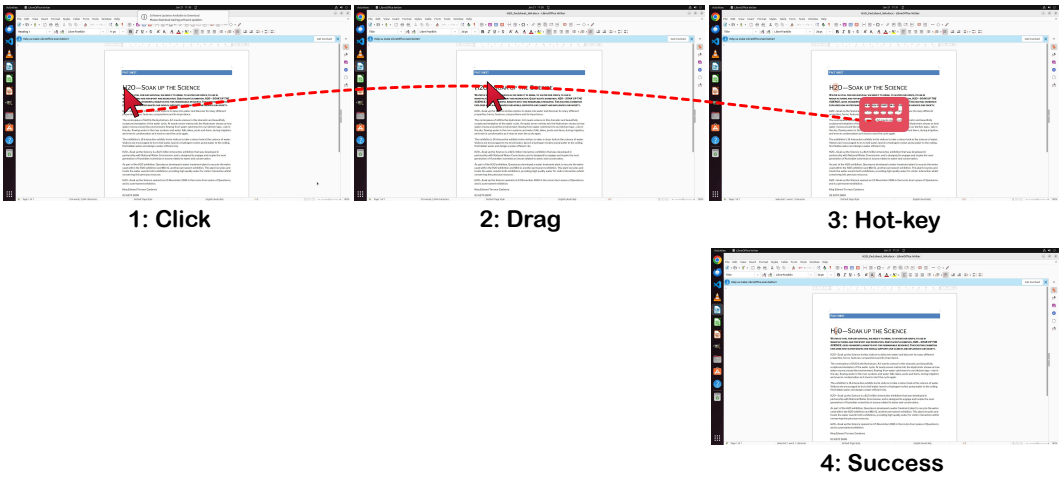
In this section, we compare two paired trajectories from OSWorld-Verified. Each pair contains a successful trajectory with EFR and a failed trajectory with the original one-step reflector. Case 1 illustrates the "small-change" regime, where the decisive evidence is only a tiny cursor or formatting change. Case 2 illustrates the "distributed-change" regime, where the target cell looks correct but the action also damages unrelated regions. In both cases, the operator can produce similar low-level actions; the key difference is whether the reflector can first organize the relevant visual evidence before deciding success.

H.1.1 Case 1: Change the 2 in "H2O" to a subscript in LibreOffice Writer

This case corresponds to `libreoffice_writer/0b17a146-2934-46c7-8727-73ff6b6483e8`. The task is to convert the "2" in "H2O" into a subscript. The success and failure trajectories differ mainly in how the reflector treats very subtle page changes. This example shows how EFR improves the reflector in the small-change regime: by explicitly grounding judgment in observed evidence, it converts a nearly invisible cursor change into a usable signal, preventing repeated false failure judgments.



Top: Failed trajectory with the conventional reflector.



Bottom: Successful trajectory with EFR.

Figure 14: Case 1: EFR succeeds by recognizing subtle cursor and formatting evidence, while the conventional reflector repeatedly misses the small change and keeps retrying the same click.

Failed trajectory with the conventional reflector.

1. **Action:** Click the character "2".
   Reflector: No visible page change is reported, so the click is judged as failed. However, a small cursor occurs next to character "2".
2. **Action:** Click the character "2" again.
   Reflector: The reflector still reports no visible page change and again judges the action as failed. However, a small cursor occurs next to character "2".
3. **Action:** Click the character "2" for a third time.
   Reflector: The reflector still reports no visible page change and again judges the action as failed. However, a small cursor occurs next to character "2".
4. **Action:** Click the character "2" for a fourth time.
   Reflector: The reflector again reports no visible change, the trajectory remains stuck in repeated retries, and the task ultimately fails.

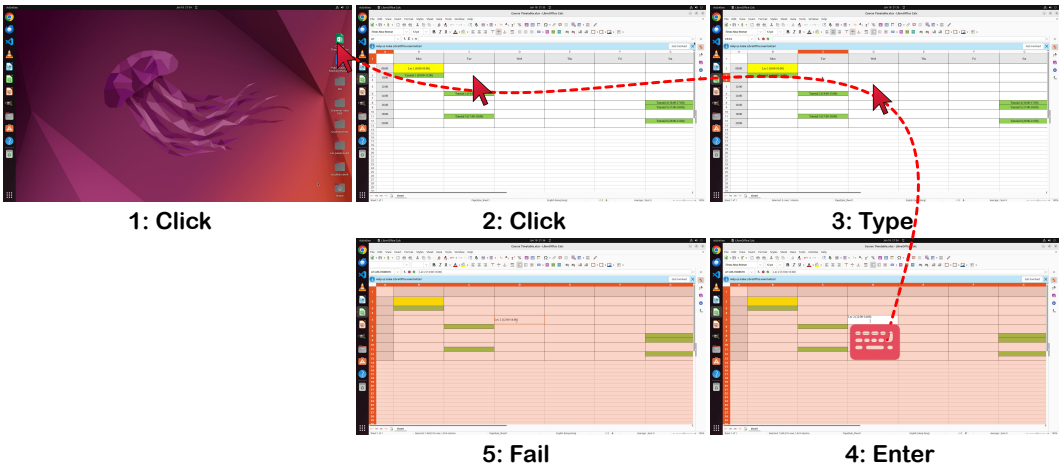
Successful trajectory with EFR.

1. **Action:** Click the character "2".
   Reflector: A text cursor appears next to "2", indicating that the click has landed on the correct position.

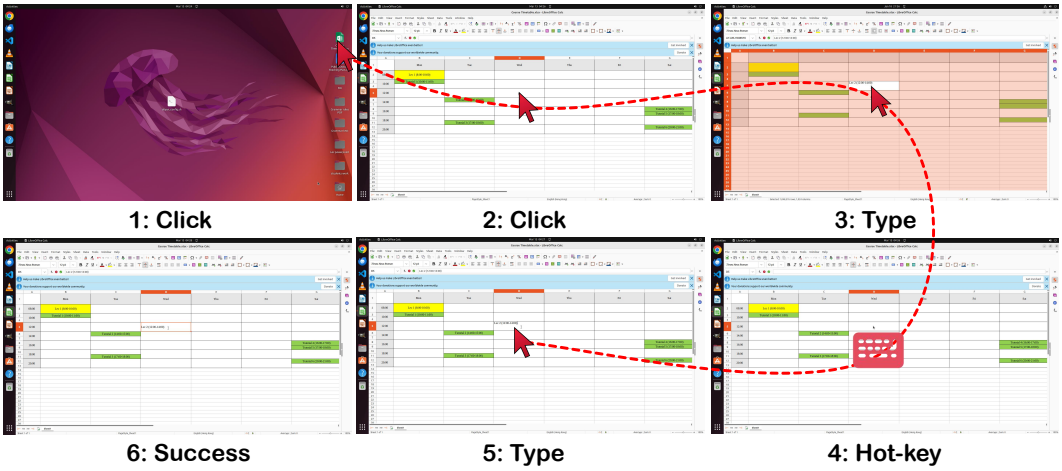
2. **Action:** Drag over the character “2”.
   Reflector: The background color of “2” changes, showing that the character has been successfully selected.
3. **Action:** Press `Ctrl+Shift+B`.
   Reflector: The “2” is rendered as a subscript, so the reflector marks the task as successful.

H.1.2 Case 2: Add Wednesday 12:00 course slot in Calc schedule

This case corresponds to `multi_apps/3a93cae4-ad3e-403e-8c12-65303b271818`. The task is to enter the course content into the Wednesday 12:00 slot in a Calc spreadsheet. Here the critical difficulty is not a missing change, but a misleading partial success: the target cell is updated, yet unrelated cells are also modified. This example shows how EFR improves the reflector in the large-change regime: instead of checking only whether the intended region contains the expected changes, it first organizes the full set of differences and can therefore reject actions with harmful side effects.



Top: Failed trajectory with the conventional reflector.



Bottom: Successful trajectory with EFR.

Figure 15: Case 2: EFR rejects a target-cell update that also damages other cells, triggers recovery, and succeeds on retry; the conventional reflector accepts the same corrupted intermediate state as success.

Failed trajectory with the conventional reflector.

1. **Action:** Click the target `xlsx` file.
   Reflector: The page switches to the spreadsheet, confirming that the file is open.

2. **Action:** Click the Wednesday 12:00 cell.
   Reflector: The changed border indicates that the target cell has been selected.

3. **Action:** Select the cell content, delete the original text, and type the new course content.
   Reflector: The reflector focuses only on the target cell, observes that the expected content appears there, and therefore judges the action as successful, while overlooking unintended changes in other cells. However, the contents of other cells are deleted as well.

4. **Action:** Press `Enter` to confirm the edit.
   Reflector: The target cell exits edit mode, so the trajectory appears successful locally, even though the spreadsheet has already been corrupted globally.

Successful trajectory with EFR.

1. **Action:** Click the target `xlsx` file.
   Reflector: The page switches to the spreadsheet, confirming that the file has been opened successfully.

2. **Action:** Click the Wednesday 12:00 cell.
   Reflector: The cell border changes, indicating that the target cell has been selected.

3. **Action:** Select the cell content, delete the original text, and type the new course content.
   Reflector: Although the target cell is filled with the desired content, the reflector also observes large-scale page changes and missing text in unrelated regions, so this step is judged as failed because the action has harmful side effects.

4. **Action:** Undo the previous edit.
   Reflector: The spreadsheet returns to the earlier state, confirming that the incorrect edit has been rolled back successfully.

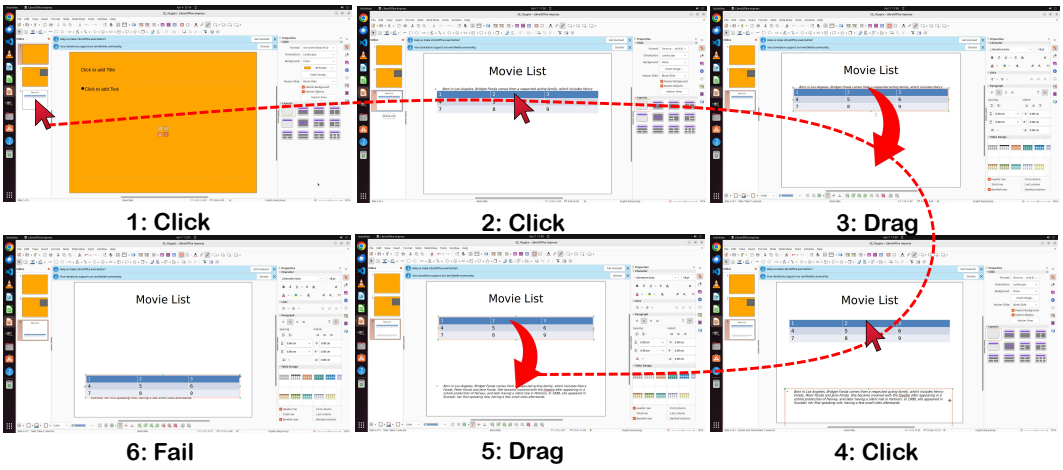
5. **Action:** Re-enter the course content in the target cell.
   Reflector: This time only the intended cell changes, so the reflector judges the action as successful and the task is completed.

H.2 Failure to Failure

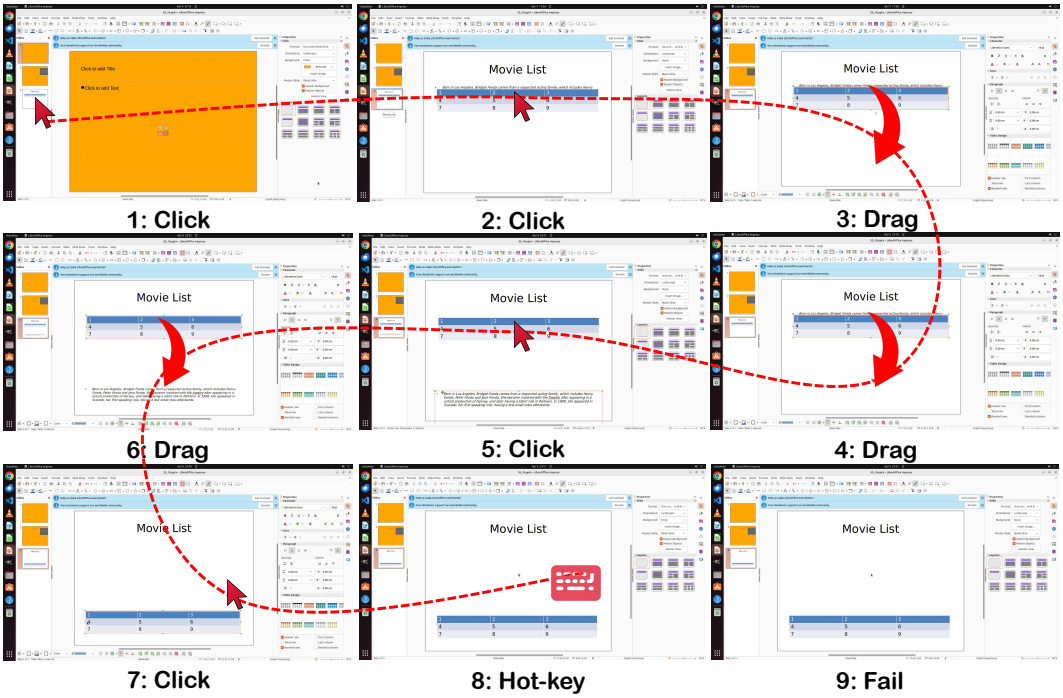
In this section, we examine a trajectory pair where both the conventional reflector and EFR fail from OSWorld-Verified. This type of case is important because it clarifies the scope of reflector improvement: EFR can make reflection more evidence-aware, but the reflector is still only one module in a larger GUI-agent system. The essential reason for failure is the errors in the planner and operator. Even if the reflector produces a correct judgment, the subsequent execution trajectory does not necessarily change as a result.

H.2.1 Case 3: Move the table on Page 3 to the bottom of the slide

This case corresponds to `libreoffice_impress/ac1b39ff-ee4d-4483-abce-c117e98942f0`. The task is to move the table on Page 3 to the bottom of the slide. Both trajectories eventually fail, but they reveal different limitations. The conventional reflector mostly accepts each apparent movement as successful, while EFR detects one failed drag action. However, after detecting the failure, the overall agent does not roll back effectively, and later actions still suffer from grounding errors.



Top: Failed trajectory with the conventional reflector.



Bottom: Failed trajectory with EFR.

Figure 16: Case 3: Both reflectors fail to complete the slide-editing task. EFR identifies an unsuccessful drag action, but the agent still does not roll back or re-plan correctly, and grounding errors continue to affect later steps.

Failed trajectory with the conventional reflector.

1. **Action:** Click Page 3 in the left slide thumbnail panel.
   Reflector: The main canvas switches to Page 3, indicating that the target slide has been opened successfully.

2. **Action:** Click the table to select it.
   Reflector: Green control points appear around the table, showing that the object has been selected.

3. **Action:** Drag the selected object downward.
   Reflector: The object appears to move down, so the reflector judges the drag as successful.

However, the table is still above the text, and the actual manipulation is likely applied to the wrong object or an incorrect selection region.

4. **Action:** Click the table again to adjust its position.
   Reflector: Green control points appear again, showing that the object has been selected.

5. **Action:** Drag the object below the text.
   Reflector: The object moves further toward the bottom area, action successful.

Failed trajectory with EFR.

1. **Action:** Click Page 3 in the left slide thumbnail panel.
   Reflector: The main canvas switches to Page 3, indicating that the target slide has been opened successfully.

2. **Action:** Click the table to select it.
   Reflector: Green control points appear around the table, showing that the object has been selected.

3. **Action:** Drag the selected object downward.
   Reflector: The page does not visibly change, action failed.

4. **Action:** Try again to drag the selected object downward.
   Reflector: The text appears to move down instead of the table, so the reflector judges the drag as failed. However, the agent does not roll back. Instead, it continues with the next steps, which still suffer from grounding errors.

5. **Action:** Click the table again to adjust the position.
   Reflector: The table is selected and green control points appear, action successful.

6. **Action:** Continue dragging the table below the text.
   Reflector: The selected object moves further toward the bottom region, action successful.

7. **Action:** Click an empty area of the slide.
   Reflector: The table selection is canceled successfully.

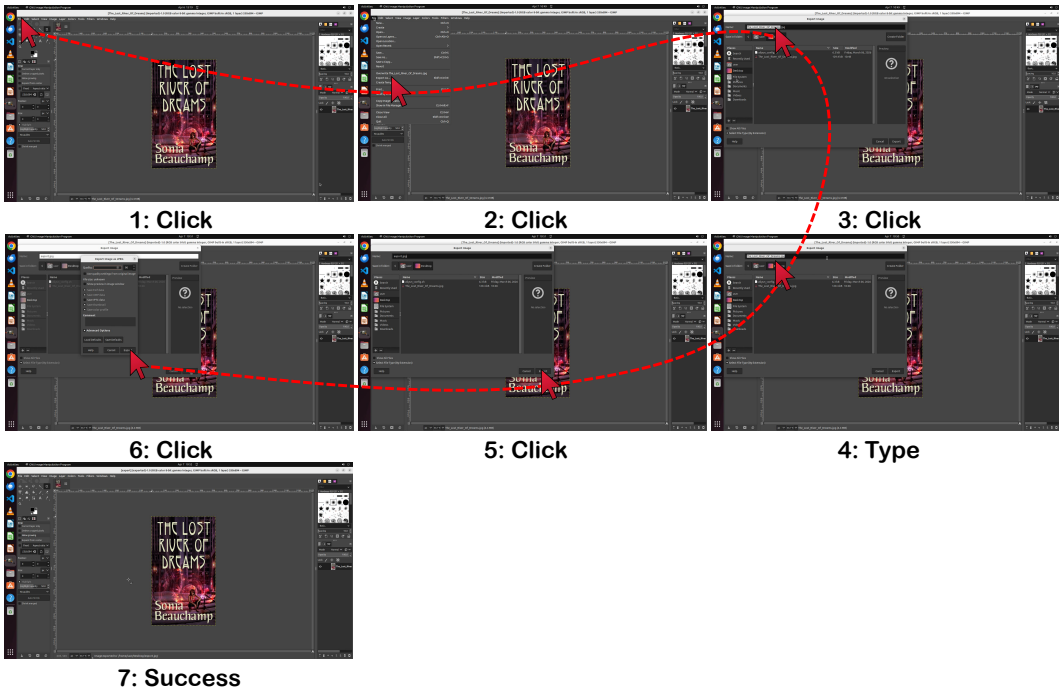
8. **Action:** Press `Ctrl+S` to save the file.
   Reflector: The file is saved successfully, but the saved state still contains the unresolved layout error.

H.3 Success to Failure

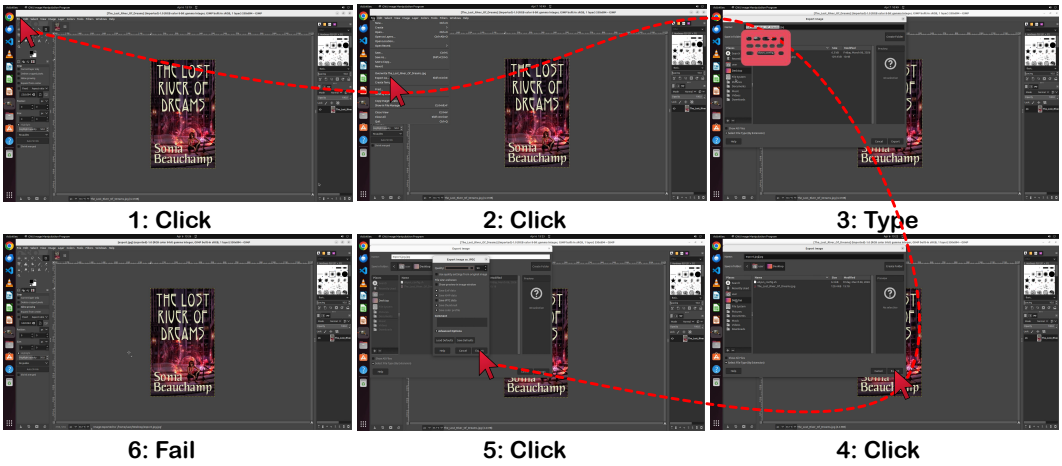
In this section, we examine a negative transition trajectory introduced by EFR from OSWorld-Verified. This example shows that the current reflector can still make an incorrect final judgment even when it observes the key visual evidence, revealing that success judgment remains difficult even when visual evidence is sufficient.

H.3.1 Case 4: Place the photo on the desktop and rename it to `export.jpg`

This case corresponds to `gimp/77b8ab4d-994f-43ac-8930-8ca087d7c4b4`. The task is to export the opened image to the desktop and rename it as `export.jpg`. The conventional reflector completes the export correctly, while EFR fails because it accepts `export.jpg.jpg` as a successful filename. Notably, the original trajectory succeeds because its operator takes the correct action, rather than because the conventional reflector is intrinsically better than EFR. This example highlights a remaining limitation of the reflector: EFR can improve evidence extraction, but the final success judgment still depends on correctly interpreting the semantics of the observed evidence.



Top: Successful trajectory with the conventional reflector.



Bottom: Failed trajectory with EFR.

Figure 17: Case 4: The conventional reflector exports the image as `export.jpg`, while EFR observes the duplicated suffix `export.jpg.jpg` but incorrectly judges it as harmless.

Successful trajectory with the conventional reflector.

1. **Action:** Click `File`.
   Reflector: A drop-down menu appears, indicating that the click succeeds.
2. **Action:** Click `Export As`.
   Reflector: The export configuration dialog appears, confirming that the export operation has been opened successfully.
3. **Action:** Triple-click the filename input box.
   Reflector: The current filename is selected, so the input field is ready to be replaced.
4. **Action:** Type `export.jpg`.
   Reflector: The filename changes to `export.jpg`, indicating that the rename step succeeds.

5. **Action:** Click `Export`.
   Reflector: A new export configuration dialog appears, confirming that the first export click succeeds.
6. **Action:** Click `Export` again.
   Reflector: The export configuration dialog disappears, indicating that the export has been confirmed and the task is completed successfully.

Failed trajectory with EFR.

1. **Action:** Click `File`.
   Reflector: A drop-down menu appears, indicating that the click succeeds.
2. **Action:** Click `Export As`.
   Reflector: The export configuration dialog appears, confirming that the export operation has been opened successfully.
3. **Action:** Type `export.jpg` in the filename input box.
   Reflector: The filename text is selected, so the reflector judges that the filename field has been activated successfully.
4. **Action:** Type `export.jpg`.
   Reflector: The filename becomes `export.jpg.jpg`. EFR correctly observes the duplicated suffix, but incorrectly judges that the extra suffix is caused by system behavior and does not affect the task.
5. **Action:** Click `Export`.
   Reflector: A new export configuration dialog appears, confirming that the first export click succeeds.
6. **Action:** Click `Export` again.
   Reflector: The export configuration dialog disappears, so the reflector judges the export as complete, but the final filename is still incorrect.

H.4 EFR Failure Case Analysis

The two EFR failure cases above reflect two different sources of remaining errors. In Case 3, the main issue is fine-grained grounding. Moving an object in a slide requires accurately selecting the intended table and dragging it to a precise location. EFR detects that one drag action does not produce the intended effect, and later observes that the text rather than the table appears to move. However, this reflection signal is not effectively converted into a recovery strategy by the subsequent planner and operator. The agent continues to issue similar manipulation actions, so the trajectory still accumulates unresolved layout errors. This case shows that even when the reflector correctly identifies a local execution failure, the full agent may still fail if later modules cannot revise the plan or repair the state.

In Case 4, the failure comes from both an input error and a semantic misunderstanding of the observed change. The agent first fails to properly replace the existing filename, causing the typed string to produce `export.jpg.jpg`. EFR observes this duplicated suffix, but incorrectly interprets it as a harmless system behavior rather than a violation of the target filename requirement. As a result, the agent accepts the intermediate state and proceeds to export the file with an incorrect name. This case shows that explicit visual evidence alone is not always sufficient: the final judgment still requires correctly mapping observed changes to task semantics, especially when small textual differences determine task success.

Overall, these cases indicate that the scattered negative results are not caused by a systematic weakness of EFR on a particular task category. Instead, they arise from limitations of the broader agent pipeline: precise grounding and manipulation remain difficult, recovery from harmful actions is still weak, and the verifier can still misinterpret the semantics of correctly observed evidence.

I Complete EFR Prompt

We report the prompt of the two-stage EFR reflector. To make the runtime assembly explicit, we replace concrete values with bracketed placeholders.

Stage 1: Page Difference Identification:

Prompt in stage 1 (System prompt):

You are a GUI Visual Change Analyzer. Your task is to identify and describe page changes that are directly related to a specific action, based on the provided action description and screenshots.
INPUT DATA:
1. Action Description: Text explaining the user's operation.
2. Before-Action Screenshot (first attached image): Contains Action Markers (e.g., blue circles, yellow arrows) indicating the operation location. It also contains the same Red Bounding Boxes with unique IDs as the After Image, highlighting the regions where visual differences were detected between the two screenshots.
3. After-Action Screenshot (second attached image): Contains Red Bounding Boxes with unique IDs highlighting all detected visual differences. The area outside the Red Bounding Boxes has not undergone any changes.
4. Bounding Boxes List: Contains the list of the coordinates of the Red Bounding Boxes, mapped to Red Box IDs in both images.
OUTPUT STRUCTURE:
Step 1: Filtering Analysis
[List every Box ID and judge it. Format: ID - Reason - Verdict]
* Area 1: [It is just the system clock changing time. Not relevant to the action.] - [Noise]
* Area 2: [It is a dropdown menu appearing near the click location, which is caused by the click action.] - [Relevant]
* Area 3: [It is the system clock changing time and a menu appearing near the click location, caused by the click action.] - [Relevant]
Step 2: Final Description
[Based ONLY on the "Relevant" areas from Step 1, generate the final description.]
1. Detailed Changes: [Describe specific changes inside relevant red boxes by logical groups.]
- Areas [List of Relevant IDs]: [E.g., "- Areas [2, 5]: A dropdown menu appeared showing options 'Save' and 'Exit'."]
2. Description: [Summarize the visual changes caused by the action in a concise manner, referencing only the Relevant areas. E.g., "After clicking 'File', a dropdown menu appeared at the top-left corner showing options 'Save' and 'Exit'."]

Prompt in stage 1 (User prompt):

Input Data:
Action Description: []
Bounding Box List (mapped to Red Box IDs in images): []

Stage 2: Action Outcome Verification:

Prompt in stage 2 (System prompt):

You are a GUI Task Verifier. Your role is to determine if the previous action was successful.
INPUT DATA:
1. Before-Action Screenshot (first attached image): This image shows the GUI state BEFORE the action was executed. Red Bounding Boxes with IDs have been artificially overlaid on this image to mark WHERE changes will happen.
2. After-Action Screenshot (second attached image): This image shows the GUI state AFTER the action was executed. The same Red Bounding Boxes with IDs are overlaid to highlight the regions where changes occurred.

3. Visual Change Description (provided in text): This is the ABSOLUTE GROUND TRUTH text report describing what actually changed AFTER the action. You MUST prioritize and trust the "Visual Change Description".
Your Task: Compare the before and after screenshots along with the "Visual Change Description" against the "Latest Action" to decide if the action achieved its intended goal.

Prompt in stage 2 (User prompt):

Latest Action
[]
Visual Changes After Action
[]
— Carefully examine the information provided above to determine whether the last action produced the expected behavior. If the action was successful, update the progress status accordingly. If the action failed, identify the failure mode and provide reasoning on the potential reason causing this failure.
Pro Tip: In rare cases, the UI might not visibly change even if a click action is performed correctly for example, when clicking on a color before drawing. In such situations, you can assume the action was successful and proceed for example, by drawing a line. When the user instruction involves adjusting some values (e.g., brightness, contrast, steps), be sure to check if the values meet expectations after the operation. If the adjusted value is very close to the target, treat the action as successful and note the remaining gap in Progress Status.
Provide your output in the following format containing four parts:
Analyse (Describe the criteria for judgment)
Final Visual State:[Describe the state AFTER the action by comparing the before screenshot (first image) and after screenshot (second image), and cross-referencing with the "Visual Changes After Action" text. Remember to absolutely trust the text if there's a conflict. Then describe if the final visual state meets the expected status or makes reasonable progress towards it]
Outcome
Choose from the following options. Give your response as "A", "B" or "C":
A: Successful or Partially Successful. The result of the last action meets the expectation, OR the action was successfully executed and made incremental progress towards the goal (even if the final objective is not completely fulfilled yet).
B: Failed. The reason for the failure is a grounding error.
C: Failed. The reason for the failure is an error in the action itself.
Error Description
If the action failed (Outcome B or C), provide a detailed description of the error and the potential reason causing this failure. If the action succeeded or partially succeeded (Outcome A), put "None" here.
Progress Status
Update the progress status.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper analyze the cost limitation.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: This paper doesn't include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: This paper disclose the experiment setup of models, benchmarks and prompts.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We use public benchmark. We have also released our source code.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We have specified all experimental setting in Evaluation and Appendix sections.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We have reported the standard deviation of three run times.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We analyze the cost of experiments in Evaluation part.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss it in the Conclusion part.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper doesn't have such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have cited all referenced papers and open-source assets.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The assets are well documented.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper doesn't involve these kinds of experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper doesn't involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: This paper describes the usage of LLMs as it is an important component of our method.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.