Interactive Reward Agent: GUI Task Evaluation via Environment-State Verification

Chenrui Shi[1,2], Yuwei Wu[1,3], Yang Liu[2], Ruining Feng[2,4], Zirui Shang[1,2], Zhi Gao[1,2,3*], Lifeng Fan[2], Che Sun[3*]

[1]Beijing Key Laboratory of Intelligent Information Technology, School of Computer Science & Technology, Beijing Institute of Technology, Beijing, China
[2]State Key Laboratory of General Artificial Intelligence, BIGAI, Beijing, China
[3]Guangdong Laboratory of Machine Perception and Intelligent Computing, Shenzhen MSU-BIT University, Shenzhen, China
[4]Tsinghua University, Beijing, China

{shichenrui,wuyuwei,shangzirui,gaozhi}@bit.edu.cn
{liuyang,fengruining}@bigai.ai, sunche@smbu.edu.cn

Project Page

Abstract

Graphical user interface task evaluation aims to determine whether a GUI agent has successfully completed a user instruction. Automated GUI task evaluation has received increasing attention because the evaluation results can serve as reward signals for both test-time scaling and post-training. However, reliable GUI task evaluation remains challenging because the judgments often require access to environment states, such as system configurations, file data, and application settings, beyond the screenshots of execution trajectories. In this paper, we propose an **interactive reward agent (IRA)** based on a propose-then-verify framework to acquire and verify evidence from the post-execution environment. Given a task instruction and a GUI environment after the GUI agent execution, IRA first proposes the task completion conditions and then verifies them by invoking system tools, application tools, and GUI tools. This design combines evidence from both visible interfaces and the environment state in an interactive process. We further introduce **GUI-RewardBench**, a benchmark of 321 GUI task trajectories spanning 10 Ubuntu desktop application categories. Experiments show that IRA achieves 86.9% accuracy on GUI-RewardBench, outperforming existing evaluator baselines. We further apply IRA to reinforcement learning of GUI agents, achieving a 34.0% OSWorld success rate, which demonstrates that IRA can provide effective reward signals for training GUI agents.

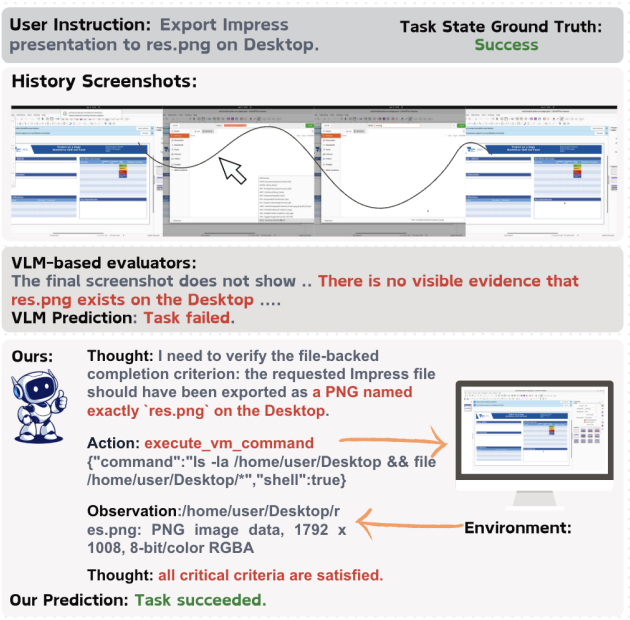


Figure 1: Comparison between the interactive reward agent (ours) and a VLM-based evaluator on a task that requires output-file verification.

Introduction

Graphical user interface (GUI) task evaluation aims to determine whether a GUI agent has successfully completed a user instruction. Recently, automated GUI task evaluation has attracted growing attention because reliable automated evaluation is important for both testing and training. For testing, automated evaluation enables test-time scaling (Lai et al. 2025; Gonzalez-Pumariega et al. 2025). For training, evaluation signals can serve directly as rewards for post-training (Xu et al. 2025; Wang et al. 2026b; Jiang et al. 2026).

Existing methods provide a strong foundation for automated evaluation of GUI tasks. Script-based evaluators used in existing GUI benchmarks can inspect files, system configurations, application states, and environment variables to determine task success (Zhou et al. 2024; Bonatti et al. 2024; Sun et al. 2025; Xie et al. 2025; Zheng et al. 2025; Shi et al. 2017). At the same time, recent progress in vision-language models (VLMs) has enabled more general evaluation methods that judge task completion from screenshots of execution trajectories (Hong et al. 2024; Lu et al. 2024; Zhang et al. 2024; Wu et al. 2025; Qin et al. 2025; Hu et al. 2025). These VLM-based evaluators reduce the need for per-task manually annotated scripts and make scalable evaluation practical. However, task completion may also be reflected in environment-state evidence, such as system configurations, file data, and application settings, beyond what screenshots

*Corresponding authors.

reveal (Zheng et al. 2026; Wang et al. 2025c; Ye et al. 2026; Cheng et al. 2024; Gou et al. 2025; Liu et al. 2026b). As Fig. 1 shows, exporting a LibreOffice Impress presentation as `res.png` to the Desktop creates an output file whose existence and content must be verified outside the application window. This example illustrates that reliable GUI task evaluation requires not only visual understanding but also access to environment states.

This observation motivates us to formulate GUI task evaluation as an interaction process between the evaluator and the environment, in which the evaluator iteratively collects evidence before reaching a final judgment. In doing so, we must address a central challenge for interactive evaluation of GUI task completion. The required evidence is often distributed across heterogeneous sources, including system configurations, file data, and application settings, whose locations and relevance cannot be determined in advance. Different tasks may require different verification strategies, and a fixed checking procedure may fail when interfaces change, applications behave unexpectedly, or relevant evidence is only partially observable. As a result, reliable GUI evaluation requires not only identifying what conditions should be verified, but also dynamically determining how the corresponding evidence should be obtained from the environment and validated.

In this paper, we propose an interactive reward agent (IRA) built on a propose-then-verify framework and equipped with tools that interact with the environment. Given a task instruction and the post-execution GUI environment, IRA first generates explicit task completion conditions and then verifies them by invoking these tools. Concretely, IRA uses a VLM to propose task-specific completion conditions, rather than directly judging task success. It then verifies these conditions by collecting evidence from the actual environment through system tools, application tools, and GUI tools. This framework retains the scalability of VLM-based evaluators across diverse tasks while enabling evidence-grounded checks similar to those used by script-based evaluators (Wei et al. 2026; Chen et al. 2026).

We further introduce **GUI-RewardBench**, a benchmark of 321 task trajectories spanning 10 Ubuntu desktop application categories, to evaluate how accurately evaluators judge GUI task completion in practical desktop environments. GUI-RewardBench covers visible-state tasks that can be verified from screenshots, hidden-state tasks that require inspecting application configurations or system settings, and artifact-verification tasks that require checking generated files. The benchmark includes applications with substantially different verification characteristics.

Experiments on GUI-RewardBench show that IRA provides strong and consistent evaluation performance. IRA achieves 86.9% overall accuracy, outperforming existing VLM-based evaluators. The improvement is especially clear in categories where completion evidence frequently resides in hidden environment states, such as VLC, Thunderbird, and multi-application workflows. The human–IRA agreement analysis on trajectories produced from 100 automatically generated tasks indicates that IRA's judgments are strongly aligned with those of human annotators. These results support the value of the propose-then-verify framework for scalable GUI task evaluation. We further use IRA to provide rewards for reinforcement learning of GUI agents. The resulting experiment suggests that IRA can offer training signals comparable to ground-truth evaluation scripts.

Our contributions are summarized as follows:

- We propose an interactive reward agent (IRA), an interactive GUI task evaluator based on a propose-then-verify framework that combines the scalability of VLM-based evaluators with the evidence-grounded verification of script-based evaluators.
- We introduce GUI-RewardBench, a benchmark of 321 GUI task trajectories whose final states are stable under replay across 10 Ubuntu desktop application categories.
- Experimental results show that IRA achieves 86.9% accuracy on GUI-RewardBench, outperforming existing evaluator baselines.

Related Work

GUI Agents. GUI agents have progressed from web agents based on structured DOM or HTML observations (Shi et al. 2017; Deng et al. 2023) to VLM-based systems that operate across web, mobile, and desktop environments. Recent work has improved visual grounding using screenshots or parsed interface elements (Hong et al. 2024; Cheng et al. 2024; Lu et al. 2024; Wu et al. 2025; Zhang et al. 2026), extended GUI agents to end-to-end computer use (Qin et al. 2025; Wang et al. 2026c; Agashe et al. 2025; Lai et al. 2025), and enhanced long-horizon execution through reinforcement learning, verifier feedback, adaptive data curation, and task-specific knowledge (Xu et al. 2025; Zhang et al. 2025a; Ding et al. 2023; Zhang et al. 2025b; Wang et al. 2025b; Liu et al. 2026a; Xie et al. 2026). As GUI agents become increasingly capable, reliably evaluating whether they have actually satisfied a user instruction becomes correspondingly important. However, many recent training and evaluation pipelines infer task success primarily from screenshot trajectories, which provide only a partial view of the environment (Bai et al. 2024; Wang et al. 2025a; Qi et al. 2025). They may miss changes to files, configurations, or backend states, and the displayed interface may not reflect the current system state. Consequently, a visually plausible trajectory or a final screenshot does not necessarily confirm task completion. IRA addresses this limitation by deriving completion conditions from the instruction and actively verifying them against the final environment state.

GUI Benchmarks. Existing GUI benchmarks cover a broad range of capabilities, including static interface grounding (Li et al. 2025a), GUI knowledge (Shi et al. 2025), cross-application mobile navigation (Lu et al. 2025), realistic web interaction (Zhou et al. 2024; Deng et al. 2023; Chezelles et al. 2025), and executable mobile, desktop, and computer-use environments (Wu et al. 2026; Wang et al. 2026a; Xie et al. 2025; Bonatti et al. 2024; Zheng et al. 2025; Li et al. 2026). Cross-platform suites further evaluate capabilities ranging from interface understanding and grounding to task automation and collaboration (Wang et al.

2025c). Despite their diversity, these benchmarks primarily ask whether a GUI agent can execute a task successfully. GUI-RewardBench studies a different problem. It asks whether an evaluator can correctly determine task completion. This shifts the evaluation target from action generation to completion verification, requiring evaluators to reason over heterogeneous evidence such as screenshots, files, configurations, generated artifacts, and application states. GUI-RewardBench therefore complements existing agent-capability benchmarks by directly evaluating the reliability of reward evaluators.

GUI Task Evaluation. Existing GUI task evaluation methods can be broadly categorized as script-based, VLM-based, and agent-based. Script-based evaluators directly inspect system or application states and can provide precise task-specific checks, but they require substantial manual labor for annotating each task (Zhou et al. 2024; Sun et al. 2025; Xie et al. 2025; Zheng et al. 2025; Shi et al. 2017). VLM-based evaluators are easier to scale across tasks, but typically judge completion from screenshots or trajectories and are therefore limited to visible evidence (Wang et al. 2025a; Bai et al. 2024; Yang et al. 2025). Agent-based evaluators rely on the reasoning abilities of language models and procedural rubrics (Zhuge et al. 2024; Wadhwa et al. 2025), but generally assume that the evidence to be judged has already been provided. GUI task evaluation presents a different challenge. The decisive evidence may not be contained in the given trajectory and can instead be distributed across screenshots, application states, files, configuration values, and command outputs (Wang et al. 2025c; Zheng et al. 2026; Ye et al. 2026; Liu et al. 2026b). IRA therefore treats evidence acquisition itself as part of evaluation. It first derives completion conditions, then determines which environment states and tools are relevant, and actively gathers evidence to verify each condition. This distinguishes IRA from prior agent-based judges that reason over fixed observations and grounds IRA's final judgment in evidence collected from the actual post-execution environment.

Interactive Reward Agent

Background and Problem Setting

Figure 2 shows the overall IRA pipeline. Given a task instruction T, screenshot evidence S_{traj}, and the post-execution environment state S_{env}, GUI task evaluation aims to predict a reward $r \in [0, 1]$, where r indicates the degree to which the task has been completed. We jointly denote the available evaluation evidence as $S = (S_{\text{traj}}, S_{\text{env}})$, where $S_{\text{traj}} = (I_{\text{init}}, I_{\text{final}})$ contains the initial and final screenshots, and S_{env} denotes interactive environment states, such as files, configuration values, application states, command outputs, system settings, and structured interface information such as accessibility trees. Evaluation paradigms differ mainly in which part of S they can access and how they verify task completion based on S.

Script-based evaluators assign a reward r_{script} using manually annotated task-specific verification functions F_T in scripts, which can be formulated as

$$r_{\text{script}} = F_T(S_{\text{env}}), \tag{1}$$

where F_T contains predefined checking rules for task T. This paradigm can provide accurate verification when task completion conditions are clear, but it requires separate scripts for different tasks.

VLM-based evaluators assign a reward r_{vlm} using a vision-language model G_θ based on visible observations, which can be formulated as

$$r_{\text{vlm}} = G_\theta(T, S_{\text{traj}}), \tag{2}$$

where G_θ predicts the task completion reward conditioned on the instruction T and the passive trajectory evidence S_{traj}. This paradigm is more general across tasks, but its verification is mainly limited to information that is visually observable or implicitly inferable from S_{traj}, making its reward prediction potentially inaccurate for tasks that require non-visual evidence or complex reasoning.

Propose-then-Verify Framework

IRA combines the scalability of VLM-based evaluators with the evidence-grounded reliability of script-based evaluators. Instead of directly predicting whether the task is complete, the agent first proposes a set of completion conditions from the task instruction T and the initial and final screenshots I_{init} and I_{final}:

$$C = P_\theta(T, I_{\text{init}}, I_{\text{final}}) = \{C_i\}_{i=1}^{N}, \tag{3}$$

where P_θ denotes the condition proposer. Each condition C_i describes one concrete requirement that should hold in the final environment state, such as the existence of an output file, the value of a configuration entry, the state of an application, or the visibility of a GUI element.

After proposing the conditions, the agent verifies each one through a ReAct-style reasoning-action-observation process. For a condition C_i, verification is not treated as a single-step query. Instead, the agent maintains a condition-specific interaction history H_i^t, where t denotes the current verification step. At step t, the agent reasons over the condition and the accumulated history, decides what additional evidence is needed, and selects a tool action:

$$a_{i,t} = \text{IRA}(C_i, H_i^{t-1}). \tag{4}$$

The selected tool then queries the final environment state and returns an observation:

$$o_{i,t} = \text{Tool}(a_{i,t}, S_{\text{env}}). \tag{5}$$

For read-only tools, the environment state remains unchanged, whereas GUI tools or executable commands may cause state transitions. The observation is appended to the history:

$$H_i^t = H_i^{t-1} \oplus (a_{i,t}, o_{i,t}), \tag{6}$$

where $\oplus$ denotes ordered concatenation.

If the returned evidence is insufficient or ambiguous, the agent does not immediately mark the condition as satisfied. Instead, it searches for an alternative evidence source and issues another tool call. In this way, verification is an iterative process: the observation from one tool call determines the next reasoning step and the next tool action.

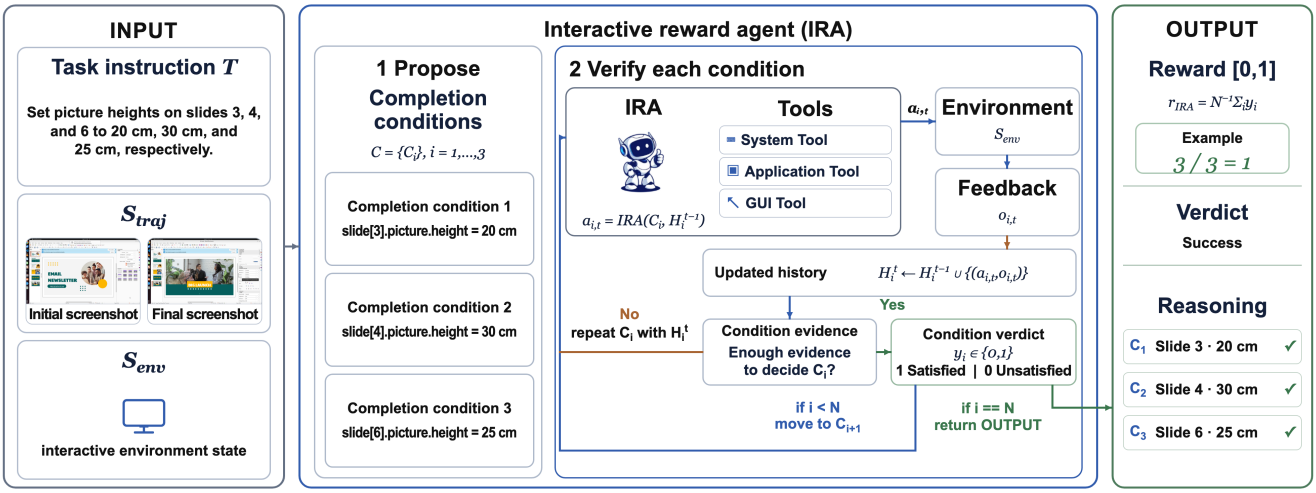


Figure 2: Overview of the Interactive Reward Agent (IRA). Given a task instruction T, the initial and final screenshots I_{init} and I_{final}, and an interactive environment state S_{env}, IRA first proposes a set of task-grounded completion conditions $\{C_i\}_{i=1}^{N}$. For each condition, IRA iteratively selects tool actions based on the condition and interaction history, receives environment feedback, and updates the history until sufficient evidence is available to assign a binary verdict y_i. The condition-level verdicts are then aggregated into the final reward, overall verdict, and reasoning.

Let K_i denote the number of verification steps performed for condition C_i. The final judgment for this condition is computed from the accumulated verification history:

$$y_i = \text{IRA}(C_i, H_i^{K_i}) \in \{0, 1\}. \tag{7}$$

We set $y_i = 1$ only when the accumulated observations in $H_i^{K_i}$ provide explicit environment-state evidence that C_i is satisfied; otherwise, we set $y_i = 0$.

The final reward is computed by aggregating condition-level judgments across all conditions:

$$r_{\text{ira}} = \frac{1}{N} \sum_{i=1}^{N} y_i. \tag{8}$$

This scalar reward represents the fraction of satisfied conditions. For binary evaluation, rewards greater than 0.8 are classified as success.

Evidence Acquisition and Tool Design

IRA obtains explicit evidence through three complementary tool groups: system tools inspect files, configurations, command outputs, and structured interface state; application tools verify generated artifacts such as documents, spreadsheets, and presentations; and GUI tools expose otherwise inaccessible interface states through interactive navigation. During verification, IRA dynamically selects the tool that best matches each completion condition, incorporates the returned evidence into its interaction history, and updates its judgment. Supplementary Sec. A.1 provides the core tool summary, complete specifications, and application-specific extensions.

GUI-RewardBench

GUI-RewardBench is designed to evaluate the accuracy of reward evaluators in judging GUI task completion from environment-state evidence. GUI-RewardBench emphasizes cases where completion evidence may be distributed across screenshots, files, application preferences, system settings, and generated artifacts.

Benchmark Construction

We construct GUI-RewardBench in Ubuntu desktop environments through trajectory replay. The task sources include OSWorld-derived tasks and manually created tasks that cover common desktop applications and multi-application workflows. The final benchmark contains 321 labeled task trajectories.

For each candidate task, we use UI-TARS-1.5 (Qin et al. 2025) and EvoCUA-8B (Xue et al. 2026) to generate trajectories. We then replay the recorded action sequence across multiple virtual-machine instances and compare the resulting final states. A trajectory is included only when the replay produces a relatively stable final state. We initially sample 327 candidates, replay them three times, and remove 6 unstable ones, resulting in 321 stable trajectories.

Trajectory-Length and Outcome Distribution

As shown in Fig. 3, GUI-RewardBench contains a relatively balanced mix of successful and failed trajectories, with diverse failure modes. Success trajectories satisfy the requested instruction, while failure trajectories include incorrect executions, partial completions, missing artifacts, wrong preference values, and visually plausible but state-inconsistent outcomes. Each benchmark trajectory is paired with the task-specific evaluation script for its underlying task.

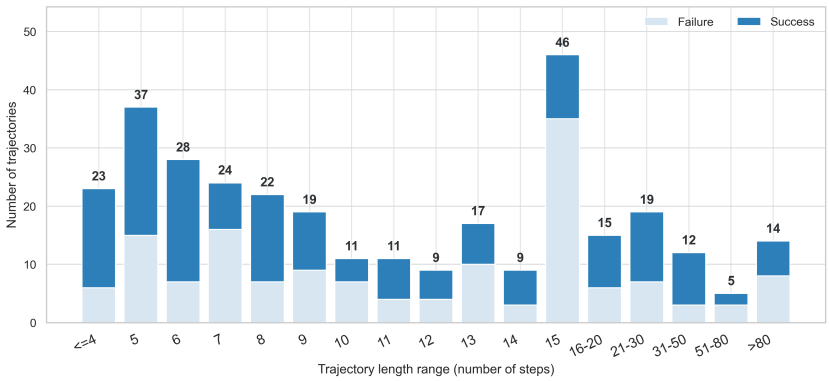


Figure 3: Distribution of trajectory length and outcome (success or failure).

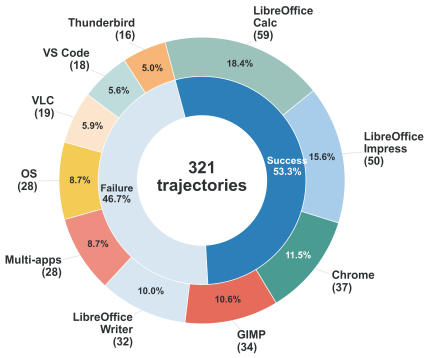


Figure 4: Task-category distribution (outer ring) and success/failure distribution (inner ring).

Benchmark Statistics

GUI-RewardBench consists of 321 stable evaluation trajectories across 10 Ubuntu desktop application categories. As shown in Fig. 3, the benchmark covers a broad range of trajectory lengths and workflow complexities. Trajectory lengths range from fewer than 5 steps to more than 80 steps, with 50 trajectories requiring over 20 steps. This distribution reflects both short interface-level operations and longer workflows involving file editing, export, and cross-application interaction.

The category distribution, shown in Fig. 4, covers office productivity, graphics editing, web browsing, communication, media playback, development, OS-level configuration, and multi-application workflows. Beyond the category diversity, GUI-RewardBench is characterized by three verification types. Artifact-verification tasks form the largest group, accounting for 192 tasks, and require inspecting generated documents, spreadsheets, presentations, images, PDFs, and other output files. Hidden-state tasks account for 89 tasks, requiring evidence beyond the visible screenshot, such as application preferences, configuration files, default handlers, or system settings. Visible-state tasks comprise the remaining 40 tasks and can typically be assessed directly from the interface.

This composition makes GUI-RewardBench suitable for evaluating desktop task verification beyond screenshot-level evidence. In particular, it stresses persistent state inspection, artifact correctness, and cross-application reasoning, which are essential for reliably judging real-world desktop workflows.

Evaluation Protocol

All evaluation methods receive the same task instruction and evaluate the environment state produced by the same evaluation pipeline: environment initialization, trajectory replay, evaluator inspection, and task-specific script verification. Each method outputs a scalar reward, which is then converted to a binary prediction using the same threshold. Rewards greater than 0.8 are classified as success. Success is treated as the positive class when computing precision, recall, and F1. Although GUI-RewardBench trajectories are selected to be relatively replay-stable, evaluation is still conducted in a live desktop environment. UI drift, asynchronous application behavior, network variability, replay imperfections, and evaluator interactions may cause the replayed state to differ from the originally recorded state. Therefore, after each live evaluation run, we execute the task-specific evaluation script and use its output as the ground-truth label for computing TP, FP, TN, and FN. This protocol compares each evaluator against the actual environment state it inspected, while keeping the task instruction, recorded trajectory, replay procedure, and verification script fixed across methods.

Experiments

Experimental Setup

Comparison methods. We organize the comparison by evaluator formulation. Existing VLM-based reward evaluators, including WebRL (Qi et al. 2025), ZeroGUI (Yang et al. 2025), DigiRL (Bai et al. 2024), and DistRL (Wang et al. 2025a), are passive evaluators. They directly judge task completion from screenshots or visual trajectory evidence. IRA is an interactive evaluator and follows the propose-then-verify framework, allowing the evaluator to inspect environment state and use tools to verify task completion. We evaluate IRA using Qwen3.6-35B-A3B (hereafter Qwen3.6) (Qwen Team 2026), GPT-5.4 (OpenAI 2026a), and GPT-5.5 (OpenAI 2026b).

Metrics and evaluation rules. We report accuracy, precision, recall, F1, TP, FP, TN, and FN. Following the evaluation protocol described above, we convert each scalar reward into a binary prediction, classifying rewards greater than 0.8 as success. Because interactive evaluators may alter the environment they inspect, the resulting positive/negative totals differ slightly across methods, even though the trajectory and verification script are held fixed.

Table 1: Performance of VLM-based reward evaluators and the interactive reward agent on GUI-RewardBench.

Method	Mode	Acc.↑	Prec.↑	Rec.↑	F1↑	TP↑	FP↓	TN↑	FN↓
VLM-based reward evaluators									
WebRL (Qi et al. 2025)	PASSIVE	47.0%	100.0%	0.6%	1.2%	1	0	150	170
ZeroGUI (Yang et al. 2025)	PASSIVE	67.6%	77.2%	55.6%	64.6%	95	28	122	76
DigiRL (Bai et al. 2024)	PASSIVE	78.5%	93.2%	64.3%	76.1%	110	8	142	61
DistRL (Wang et al. 2025a)	PASSIVE	78.8%	92.6%	65.5%	76.7%	112	9	141	59
Interactive reward agent (ours)									
IRA with Qwen3.6-35B-A3B (Qwen Team 2026)	INTERACTIVE	85.0%	88.7%	**79.6%**	83.9%	125	16	148	32
IRA with GPT-5.4 (OpenAI 2026a)	INTERACTIVE	85.4%	94.0%	76.2%	84.2%	125	8	149	39
IRA with GPT-5.5 (OpenAI 2026b)	INTERACTIVE	**86.9%**	93.7%	77.6%	**84.9%**	118	8	161	34

Quantitative Results

Results are reported in Table 1. The category-wise comparison is provided in Supplementary Sec. A.5.

IRA improves evaluation by collecting task-relevant evidence. All three IRA variants outperform all four passive VLM-based reward evaluators. The strongest passive evaluator, DistRL, achieves 78.8% accuracy and 76.7% F1, whereas every IRA variant achieves at least 85.0% accuracy and 83.9% F1. A VLM-based reward evaluator must infer task completion from fixed visual observations, which may not show whether a file was saved correctly, a setting was updated, or an output was produced in another application. IRA instead checks these states directly through tools. The results show that accurate GUI task evaluation depends not only on the VLM's reasoning ability but also on the evaluator's access to the evidence needed to verify task completion.

IRA performs consistently across the three evaluated backbones. VLM-based reward evaluators rely on the backbone model to judge task completion directly from visual observations. Their performance is therefore limited by how well the backbone generalizes beyond its training domain. For example, WebRL (Qi et al. 2025) is designed for web-related tasks, which may contribute to its low recall on GUI-RewardBench. In contrast, IRA performs consistently across the three evaluated backbones. The open-source Qwen3.6 backbone achieves results close to the proprietary GPT-5.4 and GPT-5.5 backbones and obtains the highest recall. These results show that IRA reduces the dependence on the backbone's existing ability to generalize. By giving different models the same procedure for collecting and verifying evidence, IRA enables both open-source and proprietary backbones to serve as effective GUI task evaluators.

Qualitative Results

Figure 5 compares a VLM-based reward evaluator and IRA using the same GPT-5.5 backbone. In the LibreOffice Calc example, the task is completed through a command-line update, but the spreadsheet view does not refresh. The VLM evaluator therefore predicts failure from the outdated screenshot, whereas IRA inspects the spreadsheet file and verifies the updated values. In the VLC example, the screenshot shows the current interface mode but cannot confirm whether the setting has been saved. IRA instead locates the configuration file and checks the stored value. These examples show that IRA can verify task states that are missing or unclear in screenshots. Additional examples and full trajectories are provided in Supplementary Secs. A.6 and A.10. IRA's

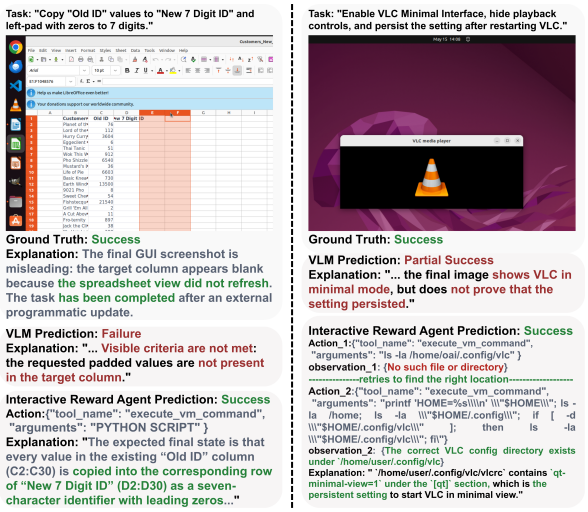


Figure 5: Qualitative comparisons between a VLM-based reward evaluator and the interactive reward agent (IRA) using the same GPT-5.5 backbone. In both cases, IRA produces the correct success verdict while the screenshot-only evaluator does not.

remaining errors mainly arise from misaligned completion conditions, including incorrect granularity, overly literal interpretations, and missed persistence requirements. Detailed cases are provided in Supplementary Sec. A.11.

Ablation Study

We compare three evaluator formulations under the same backbone. In the VLM-only setting, the evaluator receives the task instruction and the initial and final screenshots, and directly predicts whether the task is completed. The GUI-only setting additionally allows the evaluator to interact with the environment to make a judgment. The IRA setting uses the full propose-then-verify framework. Fig. 6 reports the resulting precision and recall under matched backbones.

In these experiments, GUI interaction helps only when the backbone uses it reliably. For GPT-5.4 and GPT-5.5, adding GUI exploration improves both precision and recall, showing that active inspection can reveal evidence missing from the input screenshots. Qwen3.6 shows a different pattern. Its precision improves, but its recall drops sharply, indicating a more conservative evaluation mode. GUI-only verification requires the backbone to plan interactions, ground

Table 2: Accuracy and verification cost under matched backbones. IRA token counts are reported as median/mean.

Backbone	VLM Acc.	IRA Acc.	Δ Acc. (pp)	IRA Avg. Steps	VLM Tokens	IRA Tokens (Med./Mean)
GPT-5.5	78.5%	**86.9%**	+8.4	3.34	3.05K	**14.9K / 59.5K**
GPT-5.4	76.3%	85.4%	+9.1	**3.27**	3.11K	23.8K / 120.0K
Qwen3.6-35B-A3B	78.8%	85.0%	+6.2	8.20	2.98K	25.9K / ∼139.1K

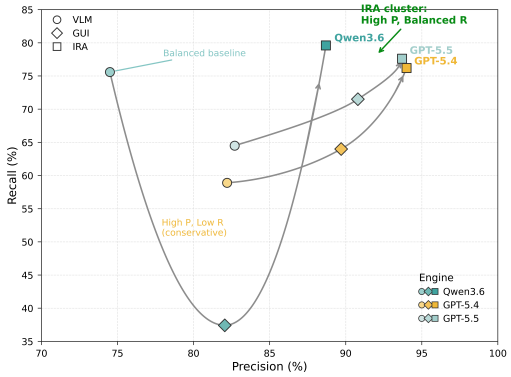


Figure 6: Precision–recall trajectories under matched backbones. Each curve connects the VLM-only, GUI-only, and IRA settings for the same backbone. The VLM-only setting judges task completion directly from the initial and final screenshots; the GUI-only setting adds environment interaction; and IRA uses the full propose-then-verify framework with GUI, system, and application tools. Circles, diamonds, and squares denote VLM-only, GUI-only, and IRA, respectively.

interface elements, recover from errors, and interpret the resulting state. When these steps are unreliable, failure to find evidence may be mistaken for evidence of task failure, making GUI interaction less effective than direct visual judgment. **IRA turns interaction into structured verification.** The full IRA setting improves the precision–recall trade-off across all three backbones and reverses the degradation observed with GUI-only Qwen3.6. This contrast shows that interaction access alone is not sufficient. IRA first defines the completion conditions and then selects system, application, or GUI tools to verify each condition with explicit evidence. By reducing reliance on open-ended and error-prone GUI exploration, IRA enables the open-source Qwen3.6 backbone to perform strongly, highlighting the importance of the verification framework and tool interface.

Verification Cost and Tool Usage

Table 2 shows that IRA improves accuracy by 6.2–9.1 percentage points. The two GPT backbones obtain these gains in about 3.3 verification steps on average, while Qwen3.6 uses more steps and a heavier-tailed token budget. This additional cost provides environment evidence that static visual reasoning cannot recover, including file contents, saved configurations, and application state. The GPT backbones also use fewer tool calls per task (2.34 and 2.53) than Qwen3.6 (7.31), favoring a small number of command-oriented checks over broader GUI exploration. Complete tool-level distributions are reported in Supplementary Sec. A.9.

Automated Task Generation and Human–IRA Agreement

To extend evaluation beyond GUI-RewardBench, we group validated tasks by their initial environment configuration and use an LLM to generate feasible instruction variants, producing 855 tasks across nine application categories. We randomly sample 100 of these tasks, execute them with a Qwen3.6-based GUI agent, and compare GPT-5.5-based IRA verdicts against human annotations. IRA agrees with human labels on 94.0% of the trajectories, yielding Cohen's $\kappa = 0.84$. See Supplementary Secs. A.7 and A.8 for more details of the generation pipeline and human-IRA agreement experiments.

Reinforcement Learning with IRA Rewards

Table 3: RL training settings and resulting OSWorld success rates.

Setting	Task Source	Evaluator	Success (%)
A	OSWorld tasks	Script	34.9
B	OSWorld tasks	IRA	34.0
C	Generated tasks	IRA	33.5

We test whether IRA can replace task-specific reward scripts and extend RL training to automatically generated tasks. Setting A trains on OSWorld tasks using ground-truth script rewards. Setting B uses the same tasks but replaces the scripts with IRA. Setting C trains on automatically generated tasks using IRA, where no task-specific evaluation script is available. We adopt the RL training method of DART (Li et al. 2025b) and use Qwen3.6-35B-A3B as the backbone for IRA. Replacing script rewards with IRA on the same OSWorld tasks yields 34.0% success, comparable to 34.9%. More notably, training on generated out-of-distribution (OOD) tasks with IRA achieves 33.5%, only 1.4 percentage points below the script-based setting. These results show that IRA converts generated tasks without evaluation scripts into effective, transferable RL supervision, enabling scalable training beyond curated benchmark tasks.

Discussion and Conclusion

Our results indicate that reliable GUI reward modeling is primarily an evidence-verification problem rather than merely

a visual classification problem. IRA separates task interpretation from evidence acquisition by decomposing GUI instructions into explicit completion conditions and verifying them through system, application, and GUI tools. This allows IRA to inspect evidence that passive evaluators cannot observe. Its consistent gains across backbones, together with the ablation and efficiency results, show the benefit of selecting decisive evidence. On the 321-trajectory GUI-RewardBench, IRA reaches 86.9% accuracy with GPT-5.5 and strong human agreement on automatically generated tasks ($\kappa = 0.84$). As an RL reward evaluator, it achieves a 34.0% OSWorld success rate, close to the 34.9% obtained with ground-truth script rewards. These results support environment-state verification as a practical approach to scalable GUI-agent evaluation and training.

References

Agashe, S.; Wong, K.; Tu, V.; Yang, J.; Li, A.; and Wang, X. E. 2025. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*.

Bai, H.; Zhou, Y.; Cemri, M.; Pan, J.; Suhr, A.; Levine, S.; and Kumar, A. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 37: 12461–12495.

Bonatti, R.; Zhao, D.; Bonacci, F.; Dupont, D.; Abdali, S.; Li, Y.; Lu, Y.; Wagle, J.; Koishida, K.; Bucker, A.; et al. 2024. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*.

Chen, C.; Shao, J.; Lu, D.; Hu, H.; Liu, X.; Yao, H.; and Liu, W. 2026. GUI-Eyes: Tool-Augmented Perception for Visual Grounding in GUI Agents. *arXiv preprint arXiv:2601.09770*.

Cheng, K.; Sun, Q.; Chu, Y.; Xu, F.; YanTao, L.; Zhang, J.; and Wu, Z. 2024. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. In *Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, 9313–9332.

Chezelles, T. L. S. D.; Gasse, M.; Drouin, A.; Caccia, M.; Boisvert, L.; Thakkar, M.; Marty, T.; Assouel, R.; Shayegan, S. O.; Jang, L. K.; Lu, X. H.; Yoran, O.; Kong, D.; Xu, F. F.; Reddy, S.; Cappart, Q.; Neubig, G.; Salakhutdinov, R.; Chapados, N.; and Lacoste, A. 2025. The BrowserGym Ecosystem for Web Agent Research. *arXiv preprint arXiv:2412.05467*.

Deng, X.; Gu, Y.; Zheng, B.; Chen, S.; Stevens, S.; Wang, B.; Sun, H.; and Su, Y. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems (NeurIPS)*, 36: 28091–28114.

Ding, S.; Du, W.; Ding, L.; Zhang, J.; Guo, L.; and An, B. 2023. Multiagent reinforcement learning with graphical mutual information maximization. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*.

Gonzalez-Pumariega, G.; Tu, V.; Lee, C.-L.; Yang, J.; Li, A.; and Wang, X. E. 2025. The unreasonable effectiveness of scaling agents for computer use. *arXiv preprint arXiv:2510.02250*.

Gou, B.; Wang, D. R.; Zheng, B.; Xie, Y.; Chang, C.; Shu, Y.; Sun, H.; and Su, Y. 2025. Navigating the digital world as humans do: Universal visual grounding for gui agents. In *International Conference on Learning Representations (ICLR)*, volume 2025, 30851–30883.

Hong, W.; Wang, W.; Lv, Q.; Xu, J.; Yu, W.; Ji, J.; Wang, Y.; Wang, Z.; Dong, Y.; Ding, M.; et al. 2024. Cogagent: A visual language model for gui agents. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 14281–14290.

Hu, X.; Xiong, T.; Yi, B.; Wei, Z.; Xiao, R.; Chen, Y.; Ye, J.; Tao, M.; Zhou, X.; Zhao, Z.; et al. 2025. Os agents: A survey on mllm-based agents for general computing devices use. *arXiv preprint arXiv:2508.04482*.

Jiang, G.; Hong, S.; Imani, M.; Bastian, N. D.; and Lan, T. 2026. Recovering Reward Functions From Distributed Expert Demonstrations via Bi-Level Maximum-Likelihood Optimization. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*.

Lai, H.; Liu, X.; Zhao, Y.; Xu, H.; Zhang, H.; Jing, B.; Ren, Y.; Yao, S.; Dong, Y.; and Tang, J. 2025. Computerrl: Scaling end-to-end online reinforcement learning for computer use agents. *arXiv preprint arXiv:2508.14040*.

Li, J.; Li, Y.; Zhao, C.; Xu, Z.; Hu, B.; and Zhang, M. 2026. WindowsWorld: A Process-Centric Benchmark of Autonomous GUI Agents in Professional Cross-Application Environments. In *Findings of the Association for Computational Linguistics: ACL 2026*, 15262–15280.

Li, K.; Meng, Z.; Lin, H.; Luo, Z.; Tian, Y.; Ma, J.; Huang, Z.; and Chua, T.-S. 2025a. Screenspot-pro: Gui grounding for professional high-resolution computer use. In *ACM International Conference on Multimedia (ACM MM)*, 8778–8786.

Li, P.; Hu, Z.; Shang, Z.; Wu, J.; Liu, Y.; Liu, H.; Gao, Z.; Shi, C.; Zhang, B.; Zhang, Z.; et al. 2025b. Efficient multi-turn rl for gui agents via decoupled training and adaptive data curation. *arXiv preprint arXiv:2509.23866*.

Liu, Q.; Jiang, Z.; Yang, H.-J.; Khosravi, M.; Waite, J. R.; and Sarkar, S. 2026a. Enhancing ppo with trajectory-aware hybrid policies. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*.

Liu, Y.; Li, P.; Wei, Z.; Xie, C.; Hu, X.; Xu, X.; Zhang, S.; Han, X.; Yang, H.; and Wu, F. 2026b. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL), Volume 1: Long Papers*, 1035–1051.

Lu, Q.; Shao, W.; Liu, Z.; Du, L.; Meng, F.; Li, B.; Chen, B.; Huang, S.; Zhang, K.; and Luo, P. 2025. Guiodyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 22404–22414.

Lu, Y.; Yang, J.; Shen, Y.; and Awadallah, A. 2024. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*.

OpenAI. 2026a. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/. Accessed: 2026-06-30.

OpenAI. 2026b. Introducing GPT-5.5. https://openai.com/index/introducing-gpt-5-5/. Accessed: 2026-06-30.

Qi, Z.; Liu, X.; Iong, I. L.; Lai, H.; Sun, X.; Sun, J.; Yang, X.; Yang, Y.; Yao, S.; Xu, W.; et al. 2025. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations (ICLR)*, volume 2025, 79791–79821.

Qin, Y.; Ye, Y.; Fang, J.; Wang, H.; Liang, S.; Tian, S.; Zhang, J.; Li, J.; Li, Y.; Huang, S.; et al. 2025. UI-TARS: Pioneering Automated GUI Interaction with Native Agents. *arXiv preprint arXiv:2501.12326*.

Qwen Team. 2026. Qwen3.6-35B-A3B: Agentic Coding Power, Now Open to All. https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Shi, C.; Yu, Z.; Gao, Z.; Feng, R.; Liu, E.; Wu, Y.; Jia, Y.; Xiang, L.; He, Z.; and Li, Q. 2025. GUI Knowledge Bench: Revealing the Knowledge Gap Behind VLM Failures in GUI Tasks. *arXiv preprint arXiv:2510.26098*.

Shi, T.; Karpathy, A.; Fan, L.; Hernandez, J.; and Liang, P. 2017. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning (ICML)*, 3135–3144. PMLR.

Sun, Q.; Cheng, K.; Ding, Z.; Jin, C.; Wang, Y.; Xu, F.; Wu, Z.; Jia, C.; Chen, L.; Liu, Z.; et al. 2025. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, 5555–5579.

Wadhwa, M.; Sprague, Z.; Malaviya, C.; Laban, P.; Li, J. J.; and Durrett, G. 2025. Evalagent: Discovering implicit evaluation criteria from the web. *arXiv preprint arXiv:2504.15219*.

Wang, B.; Lu, D.; Wang, J.; Bai, T.; Liu, S.; Zhang, Z.; Wang, H.; Hu, H.; Xie, T.; Bai, S.; Liu, D.; Shen, Q.; Lin, J.; and Yu, T. 2026a. CUA-Gym: Scaling Verifiable Training Environments and Tasks for Computer-Use Agents. *arXiv preprint arXiv:2605.25624*.

Wang, S.; Huang, W.; Yu, X.; Yang, Z.; Lin, H.; Wu, K.; Xiao, C.; Chen, C.; Wang, W.; Zhu, B.; Zhang, Y.; and Qin, C. 2026b. Beyond SFT-to-RL: Pre-alignment via Black-Box On-Policy Distillation for Multimodal RL. *arXiv preprint arXiv:2604.28123*.

Wang, T.; Wu, Z.; Liu, J.; Hao, J.; Wang, J.; and Shao, K. 2025a. Distrl: An asynchronous distributed reinforcement learning framework for on-device control agent. In *International Conference on Learning Representations (ICLR)*, volume 2025, 74757–74782.

Wang, X.; Wang, B.; Lu, D.; Yang, J.; Xie, T.; Wang, J.; Deng, J.; Guo, X.; Xu, Y.; Wu, C.; et al. 2026c. Opencua: Open foundations for computer-use agents. *Advances in Neural Information Processing Systems (NeurIPS)*, 38: 139756–139806.

Wang, X.; Wang, J.; Qian, Z.; and Zhang, B. 2025b. Online Adaptable Offline RL With Guidance Model. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*.

Wang, X.; Wu, Z.; Xie, J.; Ding, Z.; Yang, B.; Li, Z.; Liu, Z.; Li, Q.; Dong, X.; Chen, Z.; et al. 2025c. MMBench-GUI: Hierarchical Multi-Platform Evaluation Framework for GUI Agents. *arXiv preprint arXiv:2507.19478*.

Wei, J.; Ma, Q.; Zhao, Y.; Zhou, X.; Ni, K.; Gan, G.; and Cohan, A. 2026. OpenComputer: Verifiable Software Worlds for Computer-Use Agents. *arXiv preprint arXiv:2605.19769*.

Wu, D.; Hao, R.; Wang, H.; Wu, S.; Xiao, H.; Li, Z.; Zhou, B.; Ju, Z.; Liu, Z.; Fan, L.; and Zhang, Z. 2026. MobileGym: A Verifiable and Highly Parallel Simulation Platform for Mobile GUI Agent Research. *arXiv preprint arXiv:2605.26114*.

Wu, Z.; Wu, Z.; Xu, F.; Wang, Y.; Sun, Q.; Jia, C.; Cheng, K.; Ding, Z.; Chen, L.; Liang, P. P.; et al. 2025. OS-ATLAS: Foundation action model for generalist GUI agents. In *International Conference on Learning Representations (ICLR)*, volume 2025, 5090–5108.

Xie, R.; Gao, Z.; Shi, C.; Shang, Z.; Chen, L.; and Li, Q. 2026. GUIDE: Resolving Domain Bias in GUI Agents through Real-Time Web Video Retrieval and Plug-and-Play Annotation.

Xie, T.; Zhang, D.; Chen, J.; Li, X.; Zhao, S.; Cao, R.; Toh, J. H.; Cheng, Z.; Shin, D.; Lei, F.; et al. 2025. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37: 52040–52094.

Xu, Y.; Lu, D.; Shen, Z.; Wang, J.; Wang, Z.; Mao, Y.; Xiong, C.; and Yu, T. 2025. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations (ICLR)*, volume 2025, 79822–79843.

Xue, T.; Peng, C.; Huang, M.; Guo, L.; Han, T.; Wang, H.; Wang, J.; Zhang, X.; Yang, X.; Zhao, D.; et al. 2026. Evocua: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*.

Yang, C.; Su, S.; Liu, S.; Dong, X.; Yu, Y.; Su, W.; Wang, X.; Liu, Z.; Zhu, J.; Li, H.; et al. 2025. Zerogui: Automating online gui learning at zero human cost. *arXiv preprint arXiv:2505.23762*.

Ye, B.; Li, R.; Yang, Q.; Liu, Y.; Yao, L.; Lv, H.; Xie, Z.; An, C.; Li, L.; Kong, L.; et al. 2026. Claw-Eval: Towards Trustworthy Evaluation of Autonomous Agents. *arXiv preprint arXiv:2604.06132*.

Zhang, B.; Shang, Z.; Gao, Z.; Zhang, W.; Xie, R.; Ma, X.; Yuan, T.; Wu, X.; Zhu, S.-C.; and Li, Q. 2026. Tongui: Internet-scale trajectories from multimodal web tutorials for generalized gui agents. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, 12367–12375.

Zhang, C.; He, S.; Qian, J.; Li, B.; Li, L.; Qin, S.; Kang, Y.; Ma, M.; Liu, G.; Lin, Q.; et al. 2024. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279*.

Zhang, F.; Li, J.; Li, Y.-C.; Zhang, Z.; Yu, Y.; and Ye, D. 2025a. Improving sample efficiency of reinforcement learning with background knowledge from large language models. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*.

Zhang, G.; Feng, L.; Chen, X.; Tang, K.; and Tan, K. C. 2025b. Enhancing Reinforcement Learning With Cross-Domain Knowledge Transfer via Seeded Graph Matching. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*.

Zheng, C.; Mo, X.; Ma, X.; Lin, Q.; Zhao, Y.; Zhu, J.; Lou, X.; Wang, J.; Wang, Z.; Liu, W.; et al. 2026. Adaptive Milestone Reward for GUI Agents. *arXiv preprint arXiv:2602.11524*.

Zheng, L.; Huang, Z.; Xue, Z.; Wang, X.; An, B.; and Yan, S. 2025. Agentstudio: A toolkit for building general virtual agents. In *International Conference on Learning Representations (ICLR)*, volume 2025, 8044–8085.

Zhou, S.; Xu, F. F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Ou, T.; Bisk, Y.; Fried, D.; et al. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, volume 2024, 15585–15606.

Zhuge, M.; Zhao, C.; Ashley, D.; Wang, W.; Khizbullin, D.; Xiong, Y.; Liu, Z.; Chang, E.; Krishnamoorthi, R.; Tian, Y.; et al. 2024. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*.

A Appendix

This appendix provides implementation details, complete tool specifications, system prompts, hyperparameters, configurations, and baseline implementation details.

A.1 Complete Tool Set

The interactive reward agent (IRA) can support a much larger tool set spanning system-level inspection, application-specific verification, and GUI interaction capabilities. However, the complete tool set is not loaded for every evaluation episode.

During the experiments, only a subset of tools was activated depending on the target application. System tools and GUI interaction tools were always available, while application-specific tools were loaded on demand. In particular, `check_word_file`, `check_excel_file`, `check_ppt_file`, and `get_ppt_xml` were only enabled for the corresponding LibreOffice Writer, Calc, and Impress tasks. The full tool set was available only in multi-application settings where multiple software environments could be involved simultaneously.

The complete tool set includes several additional application-specific tools that were designed to provide structured access to application states and reduce the reliance on visual reasoning. These tools were introduced as optional capabilities to offload potentially expensive VLM-based inspection and verification steps. In practice, however, our experiments showed that IRA was often able to complete evaluations effectively using a much smaller core tool set. Consequently, many application-specific tools were invoked infrequently or not at all in the reported benchmarks.

This observation reflects a broader design principle of IRA: maintaining a minimal and general-purpose tool interface whenever possible, while retaining specialized tools as optional extensions for applications that may benefit from additional structured access.

- **System Tools**
  - `execute_vm_command`: Execute shell commands to inspect system state, check processes, verify system settings, or query command outputs.
    Parameters: `command` (string)
  - `get_vm_file`: Read file contents from the virtual machine filesystem. Useful for inspecting configuration files, generated outputs, logs, and text artifacts.
    Parameters: `file_path` (string)
  - `get_terminal_output`: Retrieve previous terminal command outputs when task success is reflected in command-line feedback.
  - `get_directory_listing`: List files and directories at a specified path. Verify file existence and check directory structure.
    Parameters: `directory_path` (string)
- **Chrome Tools**
  - `get_bookmarks`: Retrieve browser bookmark list with titles and URLs.
  - `get_browser_history`: Access browsing history entries with URLs, titles, and timestamps.

Table 4: Core interactive reward agent (IRA) tool set used in the main experiments.

Tool	Purpose
System Tools (Generic)	
execute_vm_command	Execute shell commands to check system state
get_vm_file	Retrieve file contents from the virtual machine
get_terminal_output	Access command history and outputs
get_host_file_content	Read host-side file contents
get_accessibility_tree	Inspect UI structure, labels, roles, and values
Application Tools	
check_word_file	LibreOffice Writer document verification
check_excel_file	LibreOffice Calc spreadsheet verification
check_ppt_file	LibreOffice Impress presentation verification
get_ppt_xml	PPT XML structure extraction
GUI Tools	Interactive GUI navigation, including clicking, typing, and scrolling

  - `get_active_tab_info`: Get current active tab's URL, title, and status.
  - `get_cookie_data`: Inspect browser cookies for a specified domain.
    Parameters: `domain` (string)
  - `get_default_search_engine`: Query the currently configured default search engine.
- **Thunderbird Tools**
  - `get_thunderbird_prefs`: Read Thunderbird preferences file (`prefs.js`) to verify configuration values.
  - `get_thunderbird_timezone`: Check the configured timezone setting in Thunderbird.
  - `get_thunderbird_accounts`: List all configured email accounts with associated settings.
- **VLC Tools**
  - `get_vlc_config`: Read VLC configuration file to inspect settings such as recording path, subtitle preferences, and audio defaults.
  - `get_default_video_player`: Query the system's default video player application.
  - `get_vlc_playing_info`: Retrieve current playback state, including file path, play/pause status, and position.
- **VSCode Tools**
  - `get_vscode_settings`: Read VSCode `settings.json` to verify editor configurations, theme, font size, and keybindings.
  - `get_vscode_extensions`: List all installed VSCode extensions with names, versions, and status.
- **LibreOffice Tools**
  - `check_word_file`: Verify LibreOffice Writer documents (`.odt`, `.docx`) by extracting text content, checking formatting, and inspecting document structure.
    Parameters: `file_path` (string)

- `check_excel_file`: Verify LibreOffice Calc spreadsheets (`.ods`, `.xlsx`) by reading cell values, formulas, and sheet structure.
  Parameters: `file_path` (string)
- `check_ppt_file`: Verify LibreOffice Impress presentations (`.odp`, `.pptx`) by extracting slide content, text, and layout structure.
  Parameters: `file_path` (string)

- **GIMP Tools**
  - `get_gimp_config_file`: Read GIMP configuration to verify tool settings, brush defaults, and preferences.
- **Interaction Tools**
  - `observe_current_state`: Capture a screenshot of the current GUI state and optionally query a vision language model about specific visual elements.
    Parameters: `query` (string, optional)
  - `get_accessibility_tree`: Retrieve a structured accessibility tree showing UI elements, roles, labels, values, and states.
  - `GUI Tools`: Perform interactive GUI actions including mouse clicks, keyboard input, scrolling, and navigation.
    Parameters:
    * `action` (string)
    * `coordinate` ([x,y])
    * `text` (string)
    * `keys` (list)
    * `pixels` (int)
    * `time` (float)
- **Final Output Tool**
  - `final_answer`: Output the final evaluation result with reward score, verdict, and detailed reasoning. This tool must be called exactly once at the end of evaluation.
    Parameters:
    * `reward` (float, 0–1)
    * `verdict` (Success / Partial Success / Failure)
    * `reasoning` (string)

A.2 System Prompt Structure

The IRA system prompt enforces a reasoning-action-observation loop and contains five key components:

Workflow Structure. The agent operates in a ReAct-style loop:

- **Thought** — The agent outputs reasoning in natural language: what condition to verify, what evidence is needed, which tool to use
- **Action** — The agent calls exactly one tool via function calling API
- **Observation** — The system executes the tool and returns the result

This pattern continues until all conditions are verified or the agent outputs a final answer.

Evaluation Protocol. The prompt enforces a five-step evaluation process:

- **Initial Analysis** — Understand task requirements, compare initial and final screenshots
- **Verification Strategy** — Choose the strongest verification path for each condition
- **Define Completion Conditions** — Decompose task into atomic, necessary completion conditions
- **Verify Each Condition** — Use tools to gather evidence, mark each condition as satisfied/unsatisfied
- **Calibrate Verdict** — Aggregate condition results into final reward

Core Principles. Three principles guide evaluation:

- **Final-State Only** — Judge based solely on final environment state; intermediate progress is ignored
- **Task-Grounded Conditions** — Every condition must be strictly necessary for task completion; do not introduce requirements beyond task instruction
- **Conservative Evaluation** — Ambiguous or unverifiable conditions are treated as NOT satisfied

Evidence Selection Policy. The prompt specifies a strict evidence hierarchy:

- For visible states → use attached screenshots
- For application states → use specialized tools (e.g., `get_bookmarks`, `get_vscode_settings`)
- For hidden states → use CLI/file tools (e.g., `execute_vm_command`, `get_vm_file`)
- For GUI inspection → use `computer` tool as a last resort

This hierarchy encourages the agent to use more reliable evidence sources first.

Strict Prohibitions. The prompt explicitly forbids common failure modes:

- Do NOT repeat tool calls after receiving results
- Do NOT speculate beyond observable evidence
- Do NOT skip writing Thought before Action
- Do NOT make multiple tool calls per turn
- Do NOT assume success without explicit verification
- Do NOT add conditions beyond task requirements

A.3 Hyperparameters and Configuration

Model Configuration. We report details of model configurations in Table 5.

A.4 Baseline Implementation Details

GUI-agent-as-reward evaluator Baselines

GPT-5.5 GUI-agent-as-reward evaluator: Use GPT-5.5 with GUI tool access. Prompted as a verification agent that can interact with the GUI to check task completion. No structured condition-centric protocol.

GPT-5.4 GUI-agent-as-reward evaluator: Same as GPT-5.5 but with GPT-5.4 backbone.

Table 5: Model configurations for IRA experiments.

Parameter	Value / Description
Backbone Models	GPT-5.5, GPT-5.4, Qwen3.6-35B-A3B
Temperature	0.0
Tool call format	OpenAI function calling API
Vision input	Screenshots encoded as base64 images
Screenshot retention	Max 5 most recent screenshots in context
Max evaluation steps	30
Retry Policy	
Max retries	3 (for API errors)
Backoff	Exponential (1s, 2s, 4s)

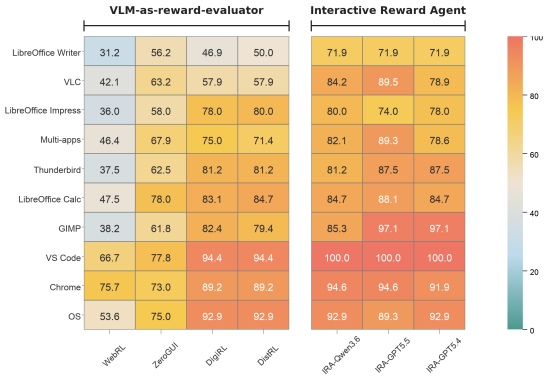


Figure 7: Per-category accuracy comparison between VLM-based reward evaluators and IRA. Each cell reports accuracy (%) for one method on one application category, with warmer colors indicating higher accuracy.

Table 6: IRA tool usage across backbones. Percentages cover tool invocations; the final row reports the average number per task.

Tool / Category	GPT-5.5	GPT-5.4	Qwen3.6
execute_vm_command	83.4%	65.6%	45.9%
GUI tools	6.4%	10.3%	50.5%
get_accessibility_tree	7.3%	16.3%	2.0%
get_terminal_output	0.0%	1.6%	0.6%
get_vm_file	1.1%	3.4%	0.3%
Other tools	1.9%	2.7%	0.7%
Avg. tools per task	**2.34**	**2.53**	**7.31**

A.5 Category-wise Evaluation Results

Figure 7 reports the per-category accuracy of the passive VLM-based reward evaluators and IRA variants. The category breakdown complements the overall results in the main paper and shows that IRA's improvement is distributed across diverse applications rather than being driven by a single category.

A.6 Additional Qualitative Results

The two qualitative comparisons summarized in the main paper use the same GPT-5.5 backbone for the VLM-based reward evaluator and IRA. In the LibreOffice Calc example, the task has been completed through a command-line update, but the visible spreadsheet view did not refresh. The screenshot-only evaluator therefore produces a false negative, whereas IRA queries the underlying file and verifies that the values were copied and zero-padded to seven digits. In the VLC example, the final screenshot shows the current interface mode but cannot establish whether the setting persists. IRA verifies the stored configuration value and searches for the correct path after its first command returns no matching file. These cases show how IRA converts incomplete visual observations into explicit evidence from files, configurations, and application states.

A.7 Automated Task Generation Details

To extend evaluation beyond the 321 trajectories in GUI-RewardBench, we develop an automated task generation pipeline to synthesize diverse GUI task instructions. The pipeline takes existing validated tasks and generates semantically similar variants through LLM-based instruction synthesis in two stages:

1. **Configuration Extraction and Grouping.** Given a set of validated tasks, the pipeline extracts the `config` field from each task, which specifies the initial environment setup, including file downloads, application states, and system configurations. Tasks with identical configurations are grouped because they share the same starting environment state and can support similar instruction variants.
2. **Instruction Generation.** For each configuration group, a language model generates new instructions that are feasible under the given configuration, require interactions similar to the reference tasks, test different aspects of the same application features, or maintain comparable complexity. The prompt includes the configuration specification, reference instructions, and application-specific constraints, and the model returns a JSON array of instruction candidates.

The pipeline processes the GUI-RewardBench task configurations and generates 855 instructions across nine application categories. Table 7 summarizes their distribution. The unusually high number of Thunderbird tasks per configuration comes from one rich mail-profile initialization containing the required inbox messages, folders, attachments, account settings, and writable preference, filter, and contact data stores. These tasks vary mainly in their objectives rather than their initial environment.

A.8 Human–IRA Agreement Details

We randomly sample 100 tasks from the 855 generated tasks, ensuring coverage across major application categories. For each task, a Qwen3.6-based GUI agent executes the instruction, IRA with GPT-5.5 evaluates the resulting trajectory, and

Table 7: Distribution of automatically generated tasks by category. The average is computed over tasks with identical initialization `config` fields.

Task Category	Generated Tasks	Avg. tasks/configuration
LibreOffice Calc	227	5.4
LibreOffice Writer	105	4.8
LibreOffice Impress	204	4.6
Chrome	97	3.9
VSCode	24	4.0
Thunderbird	54	54.0
VLC	19	3.2
GIMP	33	2.5
OS	92	5.8
Total	**855**	**4.9**

Table 8: Contingency table between human reference labels and IRA verdicts.

	IRA Success	IRA Failure	Total
Human Success	23	6	29
Human Failure	0	71	71
Total	23	77	100

human reviewers independently annotate task success. Human annotations serve as reference labels and IRA's binary verdicts as model labels.

The observed agreement rate is

$$p_o = \frac{n_{SS} + n_{FF}}{N}, \tag{9}$$

where n_{SS} denotes tasks labeled successful by both human reviewers and IRA, n_{FF} denotes tasks labeled as failures by both, and N is the total number of evaluated tasks. Cohen's kappa adjusts this agreement for chance:

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \tag{10}$$

where, for binary success/failure judgments,

$$p_e = \frac{n_{S*}}{N}\frac{n_{*S}}{N} + \frac{n_{F*}}{N}\frac{n_{*F}}{N}. \tag{11}$$

Here, n_{S*} and n_{F*} are the numbers of human success and failure labels, and n_{*S} and n_{*F} are the corresponding IRA verdict totals. The coefficient ranges from -1 (perfect disagreement) to 1 (perfect agreement), with 0 indicating chance-level agreement.

Table 8 gives an observed agreement rate of 94.0% and $\kappa = 0.84$, conventionally interpreted as almost perfect agreement. Table 9 reports the category-wise results. Cohen's κ is N/A for Chrome because all sampled Chrome tasks are labeled as failures by both human reviewers and IRA; in this single-class case, $p_e = 1$ and the denominator $1 - p_e$ is zero.

A.9 Verification Tool-Usage Details

Table 6 summarizes how the three IRA backbones acquire evidence. VM commands are the primary evidence source for the GPT backbones and still account for nearly half of Qwen3.6's calls because they directly expose files, configurations, and system state. GPT-5.5 and GPT-5.4 require only 2.34 and 2.53 tool invocations per task, whereas Qwen3.6 uses 7.31 and relies much more heavily on GUI tools. Thus, the GPT backbones tend to select a small number of decisive checks, while Qwen3.6 compensates with broader interactive exploration.

Table 9: Human–IRA agreement on generated tasks. "LibreOffice" combines Calc, Impress, and Writer.

Category	Sample Size	Agreement %	κ
LibreOffice	35	85.7%	0.62
Chrome	15	100.0%	N/A
VSCode	12	100.0%	1.00
Thunderbird	11	90.9%	0.82
GIMP	8	100.0%	1.00
OS	19	100.0%	1.00
Overall	**100**	**94.0%**	**0.84**

A.10 Qualitative trajectory visualization

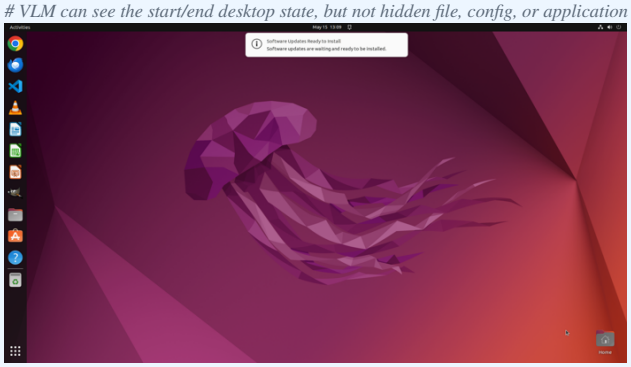
We visualize representative IRA evaluation trajectories to illustrate how the agent reaches its final judgment. Each trajectory records the initial and final screenshots available to the visual reward model, followed by the IRA's step-by-step thoughts, tool calls, GUI interactions, observations, and final verdict. These examples highlight cases where the final screenshot alone is insufficient and the correct judgment depends on hidden document contents, spreadsheet cells, browser preferences, application configuration files, or cross-application evidence.

Hidden document content: receipt unit price I have an image of my receipt located in /home/user. Please take a closer look at it and record the unit price of a new set of pedal arms in a new doc file named Details.docx on the desktop. record the number only.

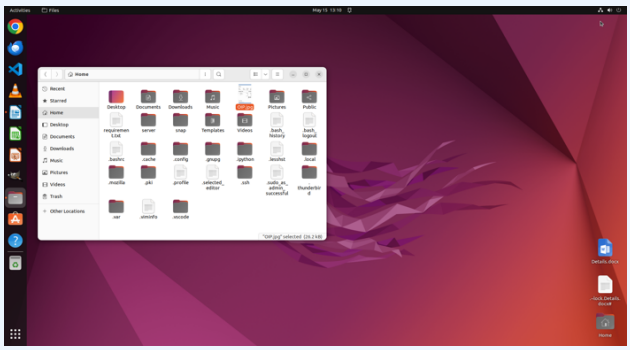
case: receipt_details_docx | model: gpt-5.5 | steps: 17 | Reward Agent: Success (1.00) | VLM: Partial Success (0.55)

Screenshots provided to the reward model

 # VLM can see the start/end desktop state, but not hidden file, config, or application state.



A. initial state



B. attempted end state

[01] execute_vm_command

```
$ find /home/user -maxdepth 2 -type f -printf '%p ' | sort
```

 # thought: I need to verify both the created document and the source receipt image. The screenshot suggests a Details.docx file exists on the desktop, but the content and correctness of the recorded unit price are hidden. I will first list the relevant files in /home/user and on the Desktop to identify the receipt image and the produced document.

```
=> /home/user/.bash_history /home/user/.bash_logout /home/user/.bashrc
/home/user/.cache/event-sound-cache.tdb.9ec62d02362e449e91284972203df915.x86_64-pc-linux-gnu...
```

[02] get_vm_file

```
$ get_vm_file /home/user/Desktop/Details.docx -> Details.docx
```

 # thought: The file listing confirms the relevant source image is likely /home/user/OIP.jpg and that /home/user/Desktop/Details.docx exists. Next I need to inspect the DOCX contents to see what number was recorded. I will download the document for structured Word verification.

```
=> cache/e76c3496-4150-40c5-a306-344519ecf87c/Details.docx
```

[03] checkwordfile

```
$ checkwordfile Details.docx -check 'Document contains the text "1450" and no other substantive text besides this number.'
```

 # thought: Now I have the produced Details.docx on the host. The task requires it to contain only the unit price number for a new set of pedal arms. I will inspect the Word file text content.

```
=> {"passed": false, "reason": "The document does not contain the number \"1450\".", "llm_code_path":...
```

[04] execute_vm_command

```
$ python3 extract word/document.xml text from Details.docx
```

 # thought: The targeted Word check failed for the guessed value, so I should not rely on that assumption. I need to determine the actual text in Details.docx and independently inspect the receipt image to identify the correct unit price. I will extract the document text directly.

```
=> 15
```

[05] get_vm_file

```
$ get_vm_file /home/user/OIP.jpg -> OIP.jpg
```

 # thought: The document contains only "15", which satisfies the "number only" formatting if the correct unit price is 15. I must verify the receipt image itself to determine the unit price of a new set of pedal arms. I will retrieve the source image from /home/user/OIP.jpg for inspection.

```
=> cache/e76c3496-4150-40c5-a306-344519ecf87c/OIP.jpg
```

[06] execute_vm_command

```
$ inspect /home/user/OIP.jpg metadata and OCR availability
```

 # thought: I need to read the receipt image. Since OCR may be available locally, I will first use read-only CLI image metadata/OCR attempts to inspect the image contents without altering anything.

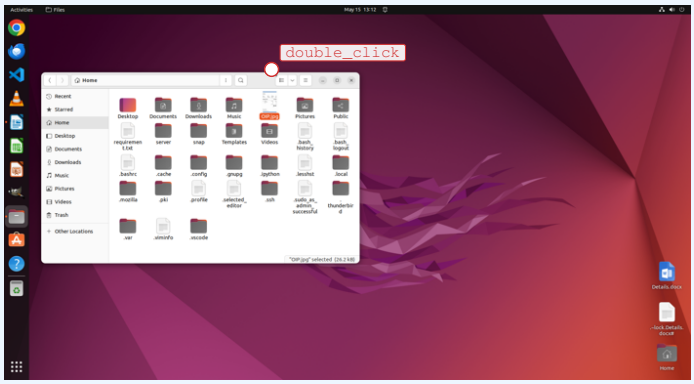
```
=> (474, 670) RGB python packages check pytesseract no No module named 'pytesseract'
```

[07] computer

```
computer[double_click] at [390, 172]
```

 # thought: OCR is not available through the initial command, so I need a visual inspection path. The file manager in the current screenshot shows OIP.jpg selected, so I will open it in the image viewer to read the receipt.

```
=> Computer action executed: double_click
```



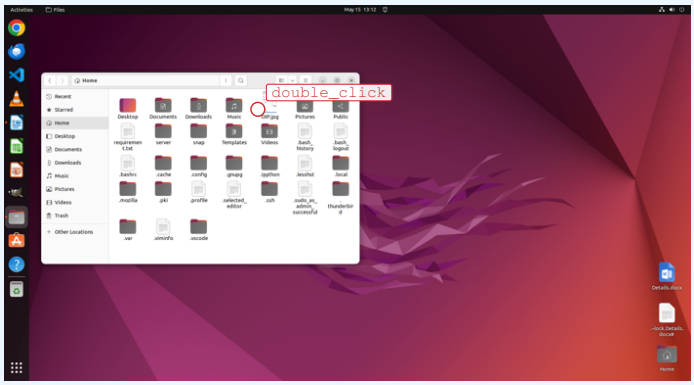
receipt_details_docx_step-007-after.png

[08] computer

```
computer[double_click] at [370, 277]
```

 # thought: The previous double-click did not open the image; the screenshot still shows the file manager with OIP.jpg selected. I likely clicked above the icon area. I will double-click the selected OIP.jpg icon more centrally to open it for visual inspection.

```
=> Computer action executed: double_click
```



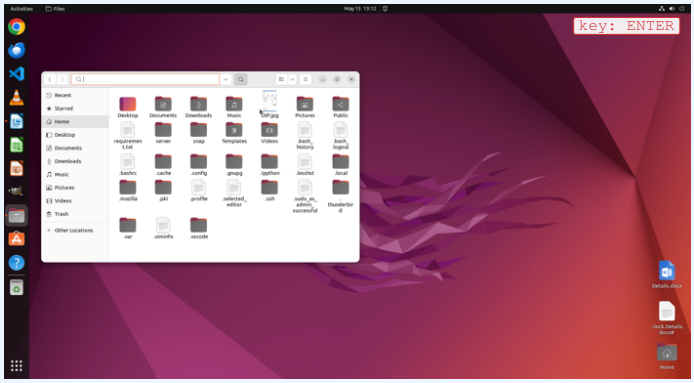
receipt_details_docx_step-008-after.png

[09] computer

```
computer[key] ENTER
```

 # thought: The image appears selected but still not opened. Since selecting a file and pressing Enter is a standard way to open it, I will use Enter to open the selected OIP.jpg and inspect the receipt visually.

```
=> Computer action executed: key
```



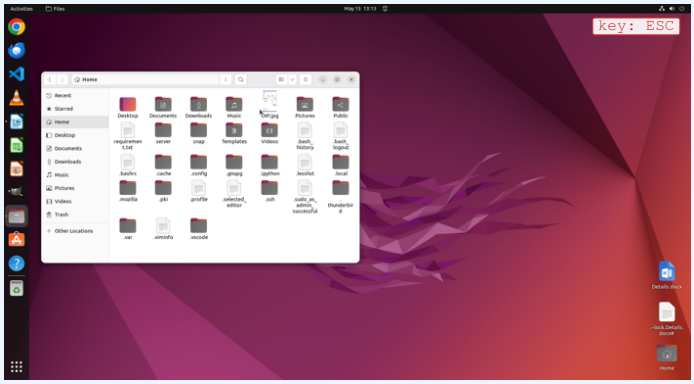
receipt_details_docx_step-009-after.png

[10] computer

computer[key] ESC

 # thought: The file did not open because my earlier coordinates were off; the OIP.jpg icon is near the upper middle of the file manager, and the search/location field is now focused. I will press Escape to leave the field, then select/open the correct OIP.jpg icon area.

=> Computer action executed: key



receipt_details_docx_step-010-after.png

[11] execute_vm_command

$ which xdg-open eog gimp display convert python3; ls target files

 # thought: I need more reliable visual reading of the image. Instead of struggling with GUI coordinates, I can use command-line image processing to enlarge/convert the image, perhaps extract embedded text using available tools like ImageMagick or save a zoomed version, then open that. First I will list whether image display tools and OCR binaries are installed.

=> /usr/bin/xdg-open /usr/bin/eog /usr/bin/gimp /usr/bin/python3 -rw-rw-r- 1 user user 4245 May 15 13:09 /home/user/Desktop/Details.docx -rw-r-r- 1 user user 26173 May 15 13:08...

[12'] invalid

 # thought: The document content is known, but I still need strict evidence for the receipt value. The file manager coordinates have been unreliable, so I will inspect the accessibility tree to locate the OIP.jpg item and understand the current UI structure before using further GUI interaction.

=> Model returned text without calling any tool

[12] execute_vm_command

$ eog /home/user/OIP.jpg >/tmp/eog_receipt.log 2>&1 &

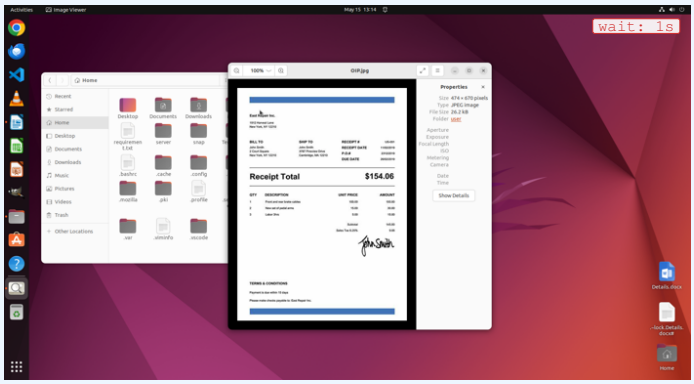
 # thought: eog is available, so I can open the image directly via read-only xdg/eog command for visual inspection. Although launching a viewer changes UI state, it is acceptable for verification and does not alter the image.

[13] computer

computer[wait] 1s

 # thought: I launched the image viewer in the background. I need to wait briefly for it to appear and then use the updated screenshot as visual evidence to read the receipt.

=> Waited 1 seconds



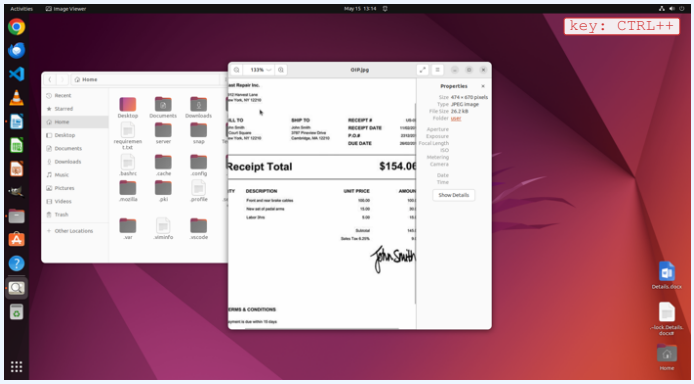
receipt_details_docx_step-013-after.png

[14] computer

computer[key] CTRL++

 # thought: The receipt image is now open and visible, but the item
table text is still small. I can see the target row appears in the
table, but I need higher confidence about the unit price format/value. I will zoom in on the image viewer so the row for "new
set of pedal arms" can be read more clearly.

=> Computer action executed: key



receipt_details_docx_step-014-after.png

[15] final_answer

reward=1 verdict=Success

 # thought: The evidence now shows the source receipt and the produced document. The receipt row for "New set of pedal arms" shows a unit price of 15.00, and the created /home/user/Desktop/Details.docx contains only "15", which is the requested number without currency or extra text. All critical criteria are satisfied.

=> Evaluation completed

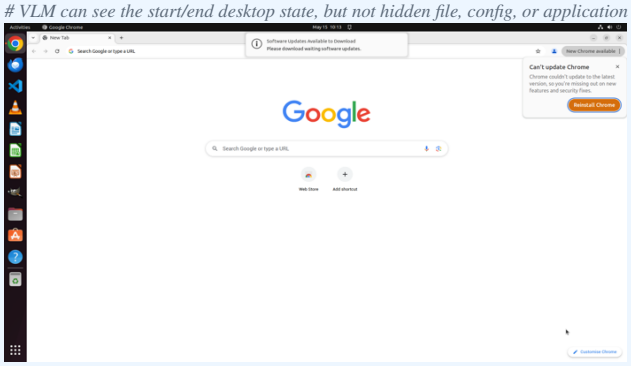
The required document exists at /home/user/Desktop/Details.docx. Direct extraction of the DOCX text showed it contains only the text "15". Visual inspection of the receipt image /home/user/OIP.jpg shows the item "New set of pedal arms" with unit price 15.00, so recording the number only as "15" satisfies the task.

Hidden browser state: startup URL removed On my surface pro whenever I launch Chrome it always opens "funbrain.com." I don't want this. I cleared my cache but it still happens. What should I do?

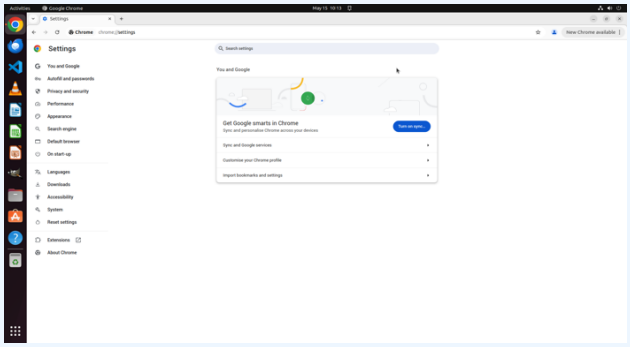
case: chrome_funbrain_startup | model: gpt-5.5 | steps: 10 | Reward Agent: Success (1.00) | VLM: Failure (0.10)

Screenshots provided to the reward model

 # VLM can see the start/end desktop state, but not hidden file, config, or application state.



A. initial state



B. attempted end state

[01] execute_vm_command

```
$ ls -la /home/oai/.config/google-chrome && find /home/oai/.config/google-chrome -maxdepth 2 -type f -name
'Preferences' -o -name 'Secure Preferences'
```

 # thought: I need to verify Chrome's actual startup configuration, because the task is about Chrome opening funbrain.com on launch and the relevant completion condition is hidden in browser preferences rather than visible on the main Settings page. I will inspect the Chrome profile configuration files to see whether funbrain.com remains configured as a startup page or related homepage setting.

=> {'output': '', 'error': "ls: cannot access '/home/oai/.config/google-chrome': No such file or directory\n"}

[02] execute_vm_command

```
$ echo $HOME; ls -la /home; find /home -maxdepth 4 -type f \( -name 'Preferences' -o -name 'Secure Preferences' \)
2>/dev/null
```

 # thought: The expected Chrome profile path was not present, so I cannot conclude from that location. I need to

identify the actual user/config location and then inspect the relevant Chrome/Chromium preference files without assuming the path.

=> /home/user total 12 drwxr-xr-x 3 root root 4096 Dec 17 2023 . drwxr-xr-x 20 root root 4096 Oct 18 2024 .. drwxr-x-- 23 user user 4096 Aug 5 2025 user /home/user/.config/Code/Preferences

[03] execute_vm_command

```
$ ls -la /home/user/.config; ps -eo pid,args | sed -n '1,200p'
```

 # thought: The home directory is /home/user, and the shallow search did not find Chrome Preferences. Since Chrome is visibly running, its profile may be in a nonstandard directory or deeper path. I will list the relevant configuration directories and running command line to locate the active user data directory.

=> total 120 drwx---- 23 user user 4096 Aug 5 2025 . drwxr-x-- 23 user user 4096 Aug 5 2025 .. drwxrwxr-x 2 user user 4096 Mar 17 2024 autostart drwx---- 19 user user 4096 Aug 5 2025 Code...

[04] execute_vm_command

```
$ find /home/user/.config/google-chrome -maxdepth 3 -type f \( -name 'Preferences' -o -name 'Secure Preferences' -o -name 'Local State' \) -print -exec python3 - «'PY' {} \; import json,sys,os p=sys.argv[1] print(' --',p,'--') try: data=json.load(open(p)) except Exception as e: print('ERR',e); sys.exit() # Print full top-level keys and selected relevant sections if present print('top_keys=', sorted(data.keys())) for key in ['session','homepage','homepage_is_new
tabpage','startup_urls','browser','profile','extensions','default_search_provider','restore_on_startup','restore_on_st
artup_migrated']: if key in data: print(key, json.dumps(data[key], indent=2)[:12000]) PY
```

 # thought: I see the google-chrome config directory exists under /home/user. I need the preference details, especially startup URLs, restore behavior, homepage, and possibly extension settings that could force funbrain.com. I will print the relevant preference files in a structured way.

=> /home/user/.config/google-chrome/Local State -- /home/user/.config/google-chrome/Local State -- top_keys= ['accessibility', 'autofill', 'background_tracing', 'breadcrumbs', 'browser',...

[05] execute_vm_command

```
$ python3 - «'PY' import json, os
paths=['/home/user/.config/google-chrome/Default/Preferences','/home/user/.config/google-chrome/Default/Secure Preferences'] for p in paths: print(' --',p,'exists',os.path.exists(p),'--') if not os.path.exists(p): continue txt=open(p,errors='replace').read() print('contains funbrain:', 'funbrain' in txt.lower()) try: data=json.loads(txt) except Exception as e: print('json error',e); continue print('top_keys=', sorted(data.keys())) for key in ['session','
homepage','homepage_is_newtabpage','startup_urls','browser','profile','extensions','default_search_provider','restore_
on_startup','restore_on_startup_migrated']: if key in data: out=json.dumps(data[key], indent=2, sort_keys=True) print(' '+key+':') print(out[:20000]) PY
```

 # thought: The previous command only displayed Local State and did not successfully show the Default profile preference contents, which are the main evidence. I need to inspect the Default profile Preferences and Secure Preferences directly for startup/homepage settings and any occurrences of funbrain.com.

=> -- /home/user/.config/google-chrome/Default/Preferences exists True -- contains funbrain: False top_keys= ['NewTabPage', 'accessibility', 'account_tracker_service_last_update',...

[06] execute_vm_command

```
$ printf ' - Chrome policy dirs - '; for d in /etc/opt/chrome /etc/chromium /etc/google-chrome /usr/share/google-chrome; do echo "DIR $d"; find "$d" -maxdepth 4 -type f 2>/dev/null | sort; done; printf ' - desktop entries containing chrome/google - '; find /usr/share/applications /home/user/.local/share/applications /home/user/Desktop -maxdepth 2 -type f 2>/dev/null | sort | while read f; do case "$f" in *chrome*|*Chrome*|*google*|*Google*) echo "FILE $f"; sed -n '1,220p' "$f"; echo; esac; done
```

 # thought: The Default Preferences show no occurrence of funbrain and no printed startup URL settings, which suggests the startup page may have been removed. However, launch behavior can also be controlled by Chrome policies or desktop shortcuts/command-line arguments, so I need to inspect those system-backed sources before judging success.

=> - Chrome policy dirs - DIR /etc/opt/chrome DIR /etc/chromium DIR /etc/google-chrome DIR /usr/share/google-chrome - desktop entries containing chrome/google - FILE...

[07] execute_vm_command

```
$ python3 - «'PY' import json, os, re base='/home/user/.config/google-chrome/Default' for name in ['Preferences','Secure Preferences']: p=os.path.join(base,name) print(' --',p,'--') data=json.load(open(p)) def walk(x,path=''): if isinstance(x, dict): for k,v in x.items(): np=f'{path}.{k}' if path else k s=k.lower() if any(term
in s for term in ['startup','restore','homepage','home_page','session']): print(np,'=',repr(v)[:1000]) walk(v,np) elif
isinstance(x, list): for i,v in enumerate(x): walk(v,f'{path}[{i}]') else: if isinstance(x,str): lx=x.lower() if any(term in lx for term in ['funbrain','http://','https://','chrome://newtab','chrome://settings']):
print(path,'=',repr(x)[:1000]) walk(data) PY
```

 # thought: The main profile and desktop launcher do not show funbrain.com, but I still need to verify the exact Chrome startup-related preference values rather than infer from missing printed sections. I will recursively enumerate

keys/values containing startup, restore, homepage, and any URL-like occurrences in the active Chrome profile.
`=> -- /home/user/.config/google-chrome/Default/Preferences --`
`extensions.settings.ahfgeienlihckogmohjhadlkjgocpleb.manifest.app.launch.web_url =`
`'https://chrome.google.com/webstore'...`

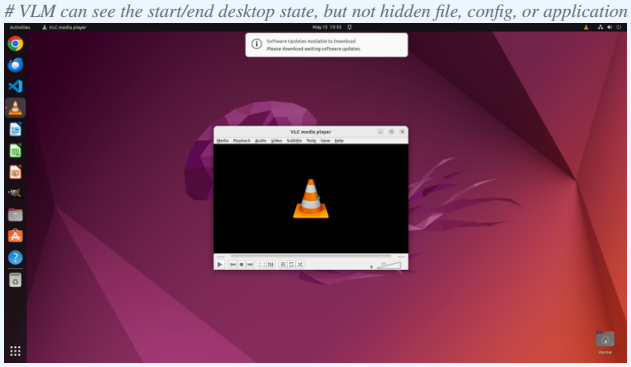
[08] execute_vm_command
`$ grep -RIl -exclude='*.log' -exclude='*.ldb' -exclude='*.sqlite*' -exclude='History' -exclude='Cookies'`
`-exclude='Cache_Data' 'funbrain' /home/user/.config/google-chrome /home/user/.local/share/applications`
`/home/user/Desktop /usr/share/applications /etc/opt/chrome /etc/chromium 2>/dev/null || true`
 # thought: The active profile's actual preference values do not contain funbrain.com or startup URL keys, while only MAC hashes remain. To rule out another persistent source such as history/session files, extension state, or shortcuts containing funbrain.com, I will search the Chrome config and common launch locations for any occurrence of "funbrain".

[09] final_answer
`reward=1 verdict=Success`
 # thought: The task requires the final Chrome launch/startup configuration to no longer open funbrain.com. Although the before/after screenshots look visually unchanged on the general Chrome Settings page, the decisive state is hidden in Chrome's profile, policy, and launcher configuration. I verified the active Chrome profile under /home/user/.config/google-chrome/Default: Preferences and Secure Preferences contain no funbrain.com, no session startup_urls setting, and no homepage setting pointing to funbrain; policy directories and the Google Chrome desktop launcher also do not contain funbrain.com. There are only built-in extensions, with no apparent extension override. Therefore the persistent sources that would make Chrome open funbrain.com on launch appear removed/absent, satisfying the objective.
`=> Evaluation completed`
Final state verifies the required end condition: Chrome is no longer configured to launch funbrain.com. The active Chrome profile Preferences contain no occurrence of funbrain.com and no startup_urls/homepage setting pointing to it; Secure Preferences also contain no occurrence. Chrome policy/launcher locations inspected did not contain funbrain.com, and the installed extensions shown in Preferences are only built-in components with no override. The screenshots themselves are visually unchanged on the main Settings page, but the hidden configuration evidence confirms the unwanted startup page is not configured.

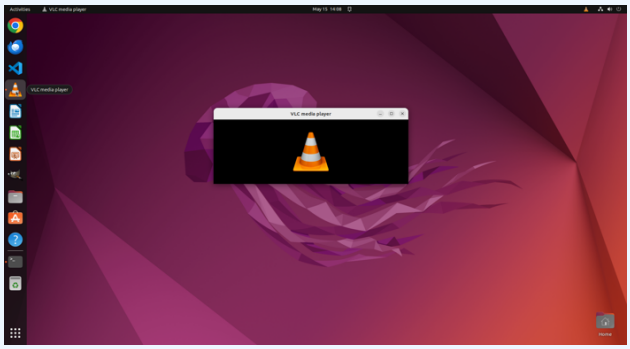
Persistence check: VLC minimal interface Enable VLC Minimal Interface in window mode so the bottom playback controls are hidden, and make sure the setting persists after restarting VLC. I often multitask on my computer, and the persistent toolbar in VLC is very distracting.

case: vlc_minimal_persistence | model: gpt-5.5 | steps: 9 | Reward Agent: Success (1.00) | VLM: Partial Success (0.75)

Screenshots provided to the reward model
 # VLM can see the start/end desktop state, but not hidden file, config, or application state.



A. initial state



B. attempted end state

[01] execute_vm_command
`$ ls -la /home/oai/.config/vlc && sed -n '1,260p' /home/oai/.config/vlc/vlcrc`
 # thought: I need to verify both critical requirements: VLC is currently in windowed minimal interface with the bottom controls hidden (visible in Image B), and the preference is persisted so it will remain after VLC restarts. The visible screenshot strongly suggests the controls and menu are hidden, but persistence is a hidden configuration criterion, so I will inspect VLC's configuration files rather than rely on appearance alone.
`=> {'output': '', 'error': "ls: cannot access '/home/oai/.config/vlc': No such file or directory\n"}`

[02] execute_vm_command

```
$ printf 'HOME=%s ' "$HOME"; ls -la /home; ls -la "$HOME/.config"; if [ -d "$HOME/.config/vlc" ]; then ls -la
"$HOME/.config/vlc"; fi
```

 # thought: The attempted config inspection used the wrong home path, as the directory '/home/oai/.config/vlc' does not exist. I need to locate the actual user home and inspect the VLC configuration there, without assuming the username.

```
=> HOME=/home/user total 12 drwxr-xr-x 3 root root 4096 Dec 17 2023 . drwxr-xr-x 20 root root 4096 Oct 18 2024 ..
drwxr-x-- 23 user user 4096 Aug 5 2025 user total 120 drwx---- 23 user user...
```

[03] execute_vm_command

```
$ cat /home/user/.config/vlc/vlc-qt-interface.conf
```

 # thought: The correct VLC config directory exists under '/home/user/.config/vlc'. To verify persistence, I need to inspect the Qt interface configuration file where VLC stores window/interface state, especially whether minimal view is enabled after restart.

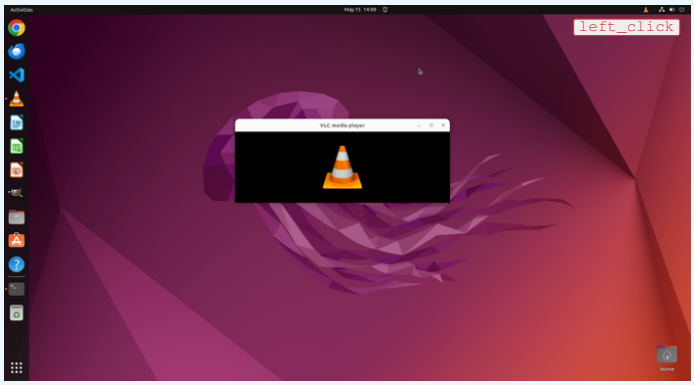
```
=> [General] filedialog-path=@Variant(\0\0\0\x11\0\0\0\n/home/user)...
```

[04] computer

```
computer[left_click]
```

 # thought: The Qt interface config confirms status bar is hidden, but it does not explicitly show the minimal-interface toggle or whether the bottom playback controls will remain hidden after a restart. Because persistence after restart is a central task requirement, I should verify it behaviorally by restarting VLC through the GUI and checking whether the window reopens in minimal interface. First I will close the current VLC window.

```
=> Computer action executed:
left_click
```



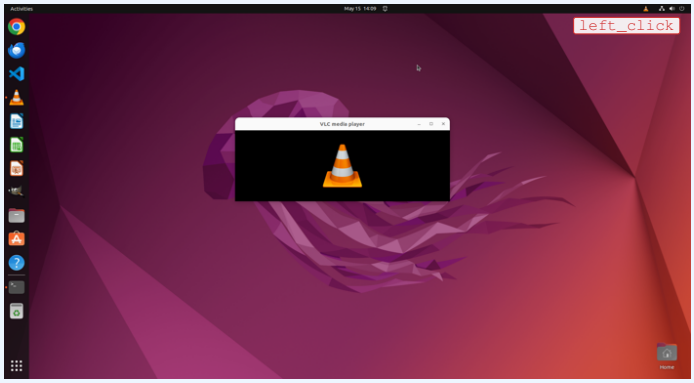
vlc_minimal_persistence_step-004-after.png

[05] computer

```
computer[left_click]
```

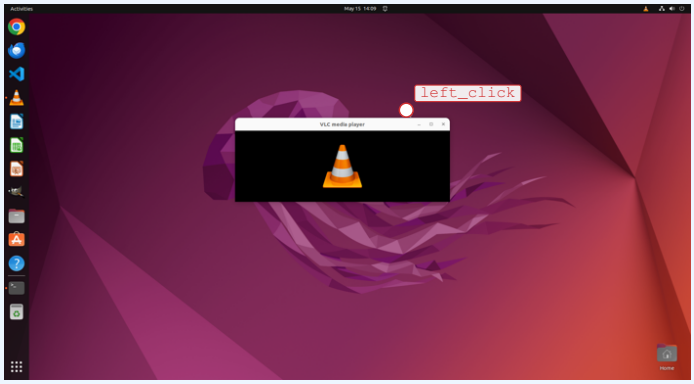
 # thought: The previous click did not close VLC; the window is still visible in minimal interface. I need to close it using the close button at the top-right of the VLC window so that I can restart VLC and verify persistence.

```
=> Computer action executed:
left_click
```



vlc_minimal_persistence_step-005-after.png

[06] computer
computer[left_click]
 # thought: I realize my previous close clicks used incorrect relative coordinates (too high on the screen); the VLC close button
is near the window's top-right at about x 450, y 122 in the 768x432
screenshot, which corresponds to roughly [586, 282]. I will click
that actual close button now to restart VLC for persistence verification.
=> Computer action executed: left_click



vlc_minimal_persistence_step-006-after.png

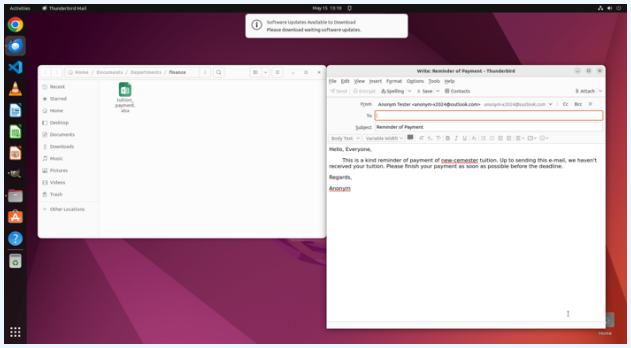
[07] execute_vm_command
$ ps -ef | grep -i '[v]lc'; printf ' -- vlc qt interface -- '; cat /home/user/.config/vlc/vlc-qt-interface.conf; printf ' -- vlcrc lines mentioning qt/minimal/interface/controls/status/toolbar -- '; sed -n '1,2600p' /home/user/.config/vlc/vlcrc | awk 'BEGIN{IGNORECASE=1} /qt|minimal|interface|controls|status|toolbar/ {print NR ":" $0}'
 # thought: The GUI close attempts have not closed the window, so instead of repeatedly clicking, I will use read-only process inspection/config inspection to gather more evidence. I need to see whether VLC is running and inspect relevant configuration entries in both vlcrc and qt config for interface settings that would indicate a persistent minimal interface.
=> user 9678 1005 0 14:08 ? 00:00:00 /usr/bin/vlc -started-from-file user 10258 2168 0 14:10 ? 00:00:00 /bin/sh -c ps -ef | grep -i '[v]lc'; printf '\n-- vlc qt interface --\n'; cat...

[08] final_answer
reward=1 verdict=Success
 # thought: The visible final state shows VLC in a window with only the video area and title bar, with the menu bar and bottom playback controls hidden, satisfying the visible minimal-interface requirement. The hidden persistence requirement is also supported by VLC's configuration: '/home/user/.config/vlc/vlcrc' contains 'qt-minimal-view=1' under the '[qt]' section, which is the persistent setting to start VLC in minimal view. Therefore both critical criteria are satisfied.
=> Evaluation completed
Success. Image B/current screenshots show VLC in windowed minimal interface with no bottom playback controls or menu bar visible. The persistent setting is verified in '/home/user/.config/vlc/vlcrc', where the Qt interface section contains 'qt-minimal-view=1', indicating VLC will start in minimal view after restart. The task's visible and persistence requirements are both satisfied.

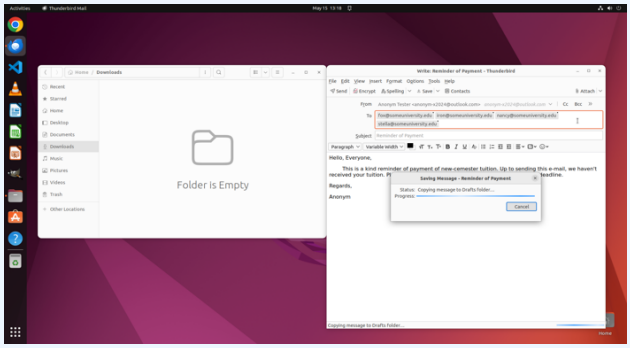
Cross-app evidence: tuition reminder recipients I've drafted an e-mail reminder for those who haven't paid tuition. Please help me to check out their e-mails from the payment record and add to the receiver field.

case: thunderbird_tuition_recipients | model: gpt-5.5 | steps: 7 | Reward Agent: Success (1.00) | VLM: Partial Success (0.75)

Screenshots provided to the reward model
 # VLM can see the start/end desktop state, but not hidden file, config, or application state.



A. initial state



B. attempted end state

[01] execute_vm_command

```
$ find /home -maxdepth 4 -type f | sed 's#^#/##' | sort
```

 # thought: I need to verify the required final state: the Thunderbird draft's recipient field should contain the email addresses of people who have not paid tuition, as determined from the payment record. The screenshots show the compose window and a receiver field, but they do not reveal the payment record or allow reliable extraction of all recipients. I will inspect the filesystem for the payment record or related files so I can compare the draft recipients against the source data.

```
=> //home/user/.bash_history //home/user/.bash_logout //home/user/.bashrc //home/user/.cache/babl/babl-fishes
//home/user/.cache/dconf/user...
```

[02] execute_vm_command

```
$ ls -laR /home/user/Documents /home/user/Downloads /home/user/Desktop 2>&1
```

 # thought: The broad file listing did not show the tuition payment spreadsheet even though Image A showed it under Documents/Departments/Finance. I need a focused directory listing to determine whether the payment record still exists and where; if it is missing, that is important evidence because the task depended on it, but I need not assume absence from the broad listing alone.

```
=> /home/user/Desktop: total 8 drwxr-xr-x 2 user user 4096 Mar 15 2024 . drwxr-x-- 23 user user 4096 May 15 13:09 ..
/home/user/Documents: total 12 drwxr-xr-x 3 user user 4096 May 15 13:09 ....
```

[03] execute_vm_command

```
$ python3 - «'PY' from openpyxl import load_workbook
path='/home/user/Documents/Departments/finance/tuition_payment.xlsx' wb=load_workbook(path, data_only=True) for ws
in
wb.worksheets: print('SHEET', ws.title) for row in ws.iter_rows(values_only=True): print('\t'.join(" if v is None
else str(v) for v in row)) PY
```

 # thought: Now I have confirmed the payment record exists at /home/user/Documents/Departments/finance/tuition_payment.xlsx. I need to inspect its contents to identify exactly which students have not paid and their email addresses, then compare that expected set with the Thunderbird recipient field. A read-only Python command can dump the workbook contents.

```
=> {'output': '', 'error': 'Traceback (most recent call last):\n File "<stdin>", line 1, in
<module>\nModuleNotFoundError: No module named \'openpyxl\'\n'}
```

[04] execute_vm_command

```
$ python3 - «'PY' import zipfile, xml.etree.ElementTree as ET
path='/home/user/Documents/Departments/finance/tuition_payment.xlsx' with zipfile.ZipFile(path) as z: print('FILES')
for n in z.namelist(): print(n) ns={'a':'http://schemas.openxmlformats.org/spreadsheetml/2006/main'} shared=[] if
'xl/sharedStrings.xml' in z.namelist(): root=ET.fromstring(z.read('xl/sharedStrings.xml')) for si in
root.findall('a:si', ns): texts=[] for t in si.iter('{http://schemas.openxmlformats.org/spreadsheetml/2006/main}t'):
texts.append(t.text or '') shared.append(''.join(texts)) print('SHARED_STRINGS') for i,s in enumerate(shared):
print(i, repr(s)) # workbook sheets and relationships sheets=[n for n in z.namelist() if
n.startswith('xl/worksheets/sheet') and n.endswith('.xml')] print('SHEETS') for sheet in sheets: print('--', sheet)
root=ET.fromstring(z.read(sheet)) for row in root.findall('.//a:sheetData/a:row', ns): vals=[] for c in
row.findall('a:c', ns): ref=c.attrib.get('r') typ=c.attrib.get('t') v=c.find('a:v', ns) val='' if v is None or v.text
is None else v.text if typ=='s' and val!='': val=shared[int(val)] elif typ=='inlineStr': texts=[] for t in
c.iter('{http://schemas.openxmlformats.org/spreadsheetml/2006/main}t'): texts.append(t.text or '') val=''.join(texts)
vals.append(f'{ref}={val}') print(' | '.join(vals)) PY
```

 # thought: The openpyxl library is unavailable, so I need another read-only way to inspect the xlsx. Since an .xlsx is a ZIP archive of XML files, I can list and parse its XML contents directly to recover the table values.

```
=> FILES _rels/.rels xl/_rels/workbook.xml.rels xl/workbook.xml xl/theme/theme1.xml xl/styles.xml
```

xl/worksheets/sheet1.xml xl/sharedStrings.xml docProps/core.xml docProps/app.xml...

[05] get_accessibility_tree

{}

 # thought: The spreadsheet shows the unpaid students are Fox, Iron, Nancy, and Stella, with emails fox@someuniversity.edu, iron@someuniversity.edu, nancy@someuniversity.edu, and stella@someuniversity.edu. Image B visibly appears to contain those four addresses in the Thunderbird To field, but I want stronger structured evidence for the compose window's recipient field before final judgment. The accessibility tree should expose the exact text in the active UI.

=> <desktop-frame xmlns:st="https://accessibility.ubuntu.example.org/ns/state" xmlns:attr="https://accessibility.ubuntu.example.org/ns/attributes"...

[06] final_answer

reward=1 verdict=Success

 # thought: I have verified the critical final-state requirements. The payment record lists "No" for Fox, Iron, Nancy, and Stella, whose emails are fox@someuniversity.edu, iron@someuniversity.edu, nancy@someuniversity.edu, and stella@someuniversity.edu. The Thunderbird compose window's To field contains exactly those four address pills and no additional recipient pills. Although there is a draft-save error dialog, saving/sending was not part of the task, and the required receiver field has been correctly populated.

=> Evaluation completed

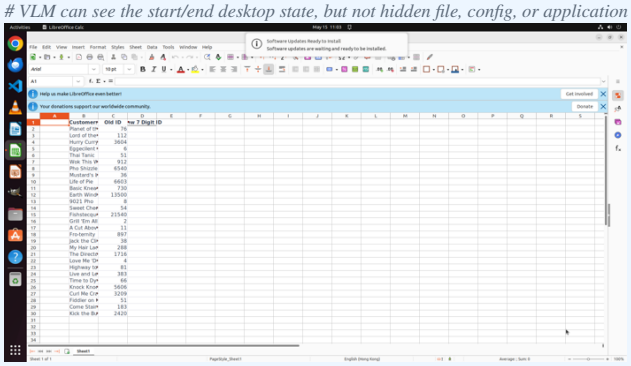
The task was to find the unpaid students' emails from the payment record and add them to the email receiver field. Inspecting the spreadsheet showed unpaid entries for Fox, Iron, Nancy, and Stella, with emails fox@someuniversity.edu, iron@someuniversity.edu, nancy@someuniversity.edu, and stella@someuniversity.edu. The Thunderbird compose window's To field contains exactly these four recipient address pills. No extra or missing recipients were found. The visible draft-save error is not relevant to the requested receiver-field update.

Spreadsheet structure beats misleading pixels I would like to copy all the numbers in the 'Old ID' column to the 'New 7 Digit Id' column, and pad them with zeros in front, to fill them up to seven digits. Finish the work and don't touch irrelevant regions, even if they are blank.

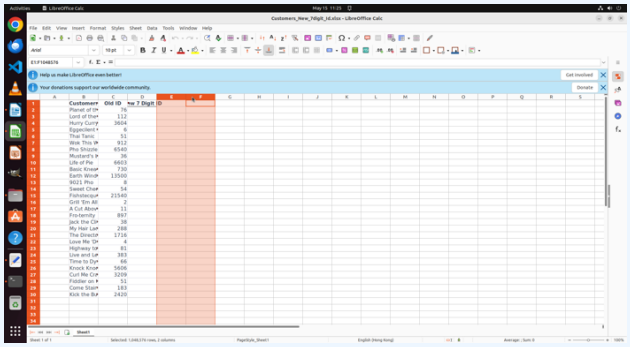
case: calc_old_id_padding | model: gpt-5.5 | steps: 4 | Reward Agent: Success (1.00) | VLM: Failure (0.00)

Screenshots provided to the reward model

 # VLM can see the start/end desktop state, but not hidden file, config, or application state.



A. initial state



B. attempted end state

[01] execute_vm_command

```
$ find /home -maxdepth 4 -type f | sed 's#^#/##' | head -200
```

 # thought: I need to verify the actual spreadsheet contents, not just the visible screenshot, because padding with leading zeros may be represented as text/formatting and many rows are involved. The active document title indicates a file named "Customers_New_7digit_id.xlsx", so I will inspect the filesystem to locate the spreadsheet and then read its cell values.

=> //home/user/.gnupg/pubring.kbx //home/user/.gnupg/trustdb.gpg //home/user/.bashrc //home/user/server/main.py //home/user/server/screenshots/screenshot.png //home/user/server/out.log...

[02] execute_vm_command

```
$ python3 - «'PY' from openpyxl import load_workbook path='/home/user/Customers_New_7digit_Id.xlsx' wb=load_workbook(path,data_only=False) ws=wb.active print('sheet', ws.title, 'max_row', ws.max_row, 'max_col', ws.max_column) for r in range(1, min(ws.max_row,40)+1): vals=[] for c in range(1, min(ws.max_column,8)+1):
```

```
cell=ws.cell(r,c) vals.append((cell.coordinate, cell.value, cell.number_format, cell.data_type)) print(vals) PY
# thought: The file is present at '/home/user/Customers_New_7digit_Id.xlsx'. I need to inspect the workbook cells to
confirm whether the "New 7 Digit Id" column contains the corresponding Old ID values padded to seven digits, and
whether unrelated columns were changed. I will read the workbook values and formats with Python.
=> sheet Sheet1 max_row 32 max_col 4 [('A1', None, 'General', 'n'), ('B1', 'Customers', 'General', 's'), ('C1', 'Old
ID', 'General', 's'), ('D1', 'New 7 Digit ID', 'General', 's')] [('A2',...
```

[03] final_answer

reward=1 verdict=Success

 # thought: The expected final state is that every value in the existing "Old ID" column (C2:C30) is copied into the corresponding row of "New 7 Digit ID" (D2:D30) as a seven-character identifier with leading zeros, while other customer data and irrelevant blank regions remain unchanged/empty. Visual evidence shows D was populated, and file-backed verification confirms each D value is exactly the zero-padded form of the corresponding C value, with no data added beyond column D. Therefore all critical criteria are satisfied.

=> Evaluation completed

The workbook '/home/user/Customers_New_7digit_Id.xlsx' was inspected directly. Rows D2:D30 under "New 7 Digit ID" contain the exact seven-character zero-padded versions of the corresponding Old ID values in C2:C30, e.g. 76 -> 0000076, 3604 -> 0003604, 13500 -> 0013500, and all remaining rows similarly match. The customer names and old IDs remain present, and the workbook has no data beyond the intended columns; irrelevant blank regions are empty. The final state satisfies the task fully.

A.11 Interactive Reward Agent Error Case Analysis

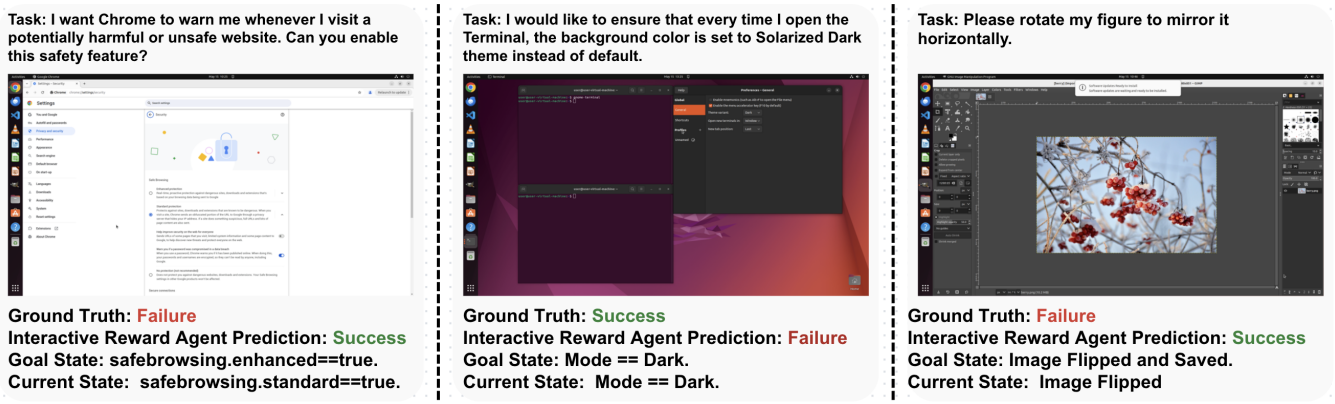


Figure 8: Representative IRA failures caused by misaligned completion conditions: incorrect protection granularity, an overly literal theme interpretation, and failure to verify that an edited artifact was saved.

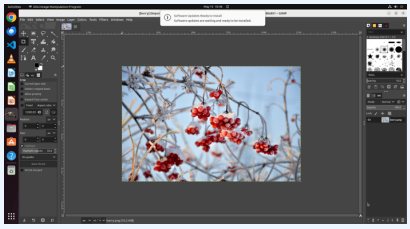
Fig. 8 demonstrates failures caused by misaligned completion conditions rather than insufficient evidence acquisition. IRA incorrectly accepts standard protection for a task requiring enhanced protection, rejects a valid dark-mode state by interpreting "Solarized Dark" too literally, and infers completion from a historical horizontal-flip operation without verifying the implicit requirement that the flipped image be saved. These cases show that user instructions may be ambiguous or omit expectations such as acceptable semantic equivalents and persistence. Consequently, IRA may formulate incorrect completion conditions even when relevant evidence is available. Robust evaluation therefore requires better alignment between the literal instruction, the user's underlying intent, and the completion conditions used for verification. More cases and detailed trajectories are provided in the following.

False positive: operation evidence without saved artifact Please rotate my figure to mirror it horizontally.

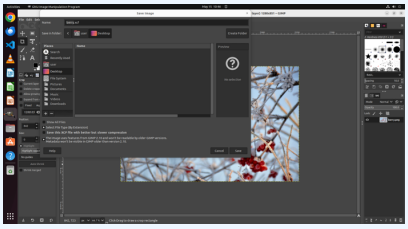
case: gimp_mirror_unsaved | model: gpt-5.5 | steps: 6 | category: Implicit completion requirement | Reward Agent: Success (1.00) | GT: (0.00)

Screenshots provided to the reward model

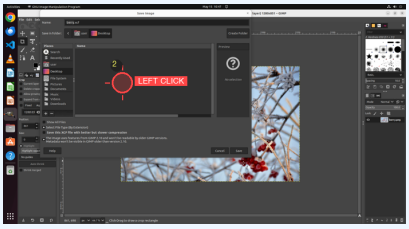
VLM can see the start/end desktop state, but not hidden file, config, post-execution configuration state, or application state.



A. Initial image state



B. Attempted end state: save dialog still visible



C. Reward Agent GUI verification attempt

[Task reading] What the task should require

Mirror the figure horizontally and leave the edited figure saved/exported as the final artifact. An undo-history entry alone is not enough.

[Error mechanism] Why the Reward Agent judgment is wrong

Reward Agent relied on GIMP state evidence such as 'Undo Flip Layer' and treated the open save dialog as irrelevant. The failure is task-semantics calibration: the model did not infer the persistence requirement hidden in the user's editing request.

[Key verification steps] Evidence used by the judge

1. **Step 1-3**: Reward Agent tried to dismiss the save dialog to inspect the visible canvas.
2. **Step 4**: It inspected the accessibility tree and found evidence of a flip operation in the undo history.
3. **Final**: It returned Success because the visual operation occurred, explicitly saying no save/export requirement was specified.

[Final judgment] Reward Agent output and paper takeaway

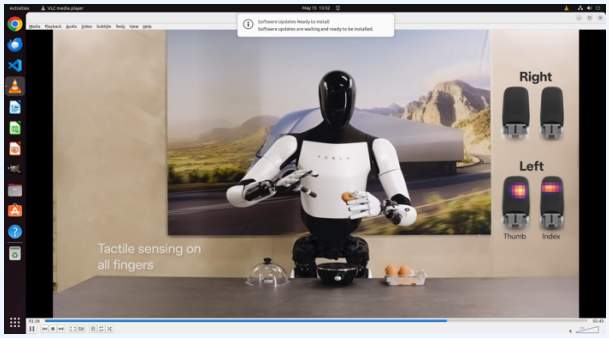
Reward Agent returned Success with reward 1.00. The model verified that the operation happened, but missed that an image-editing request usually requires a persisted output.

False negative: transient controls mistaken for non-fullscreen state Can you enable fullscreen mode in VLC so that the video fills up the whole screen?

case: vlc_fullscreen_overlay | model: gpt-5.5 | steps: 2 | category: Visual overconfidence | Reward Agent: Failure (0.00) | GT: True (1.00)

Screenshots provided to the reward model

VLM can see the start/end desktop state, but not hidden file, config, post-execution configuration state, or application state.



A. Initial VLC state

B. Attempted end state with visible overlay

[Task reading] What the task should require

Enable VLC fullscreen. Temporary playback controls or progress overlays do not necessarily imply the video is windowed.

[Error mechanism] Why the Reward Agent judgment is wrong

Reward Agent stopped after comparing screenshots. It treated the visible menu/progress controls as decisive evidence of non-fullscreen, although the benchmark oracle checked VLC window size against screen size.

[Key verification steps] Evidence used by the judge

1. **Final only**: Reward Agent made the failure decision directly from the screenshot.
2. **Missed check**: It did not query window size or application fullscreen state.

[Final judgment] Reward Agent output and paper takeaway

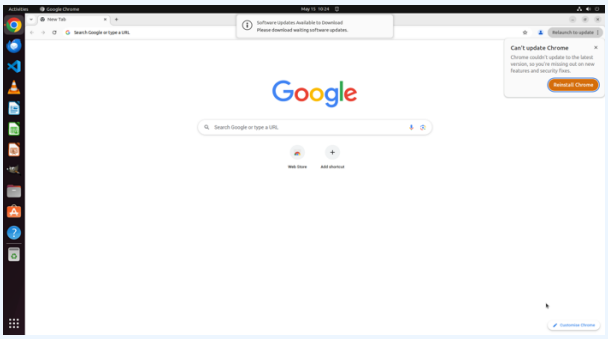
Reward Agent returned Failure with reward 0.00. The model over-trusted visible controls and did not verify the actual window/screen geometry.

False positive: correct setting family but wrong protection tier I want Chrome to warn me whenever I visit a potentially harmful or unsafe website. Can you enable this safety feature?

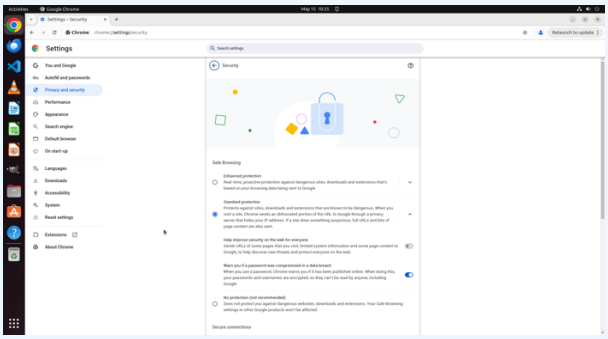
case: chrome_standard_vs_enhanced | model: gpt-5.5 | steps: 6 | category: Wrong task granularity | Reward Agent: Success (1.00) | GT: (0.00)

Screenshots provided to the reward model

VLM can see the start/end desktop state, but not hidden file, config, post-execution configuration state, or application state.



A. Initial Chrome state



B. Security settings inspected by Reward Agent

[Task reading] What the task should require

The target criterion is Enhanced Safe Browsing, i.e., 'safebrowsing.enhanced == true'; Standard Safe Browsing is insufficient.

[Error mechanism] Why the Reward Agent judgment is wrong

Reward Agent correctly navigated to Chrome security settings, but collapsed Standard and Enhanced protection into the same semantic bucket. This is not a tool-access failure; it is a criterion-level interpretation error.

[Key verification steps] Evidence used by the judge

1. **Step 1**: It inspected Chrome preference files for Safe Browsing state.
2. **Step 2-4**: It opened 'chrome://settings/security' for direct UI evidence.
3. **Final**: It accepted Standard protection as satisfying the safety-warning request.

[Final judgment] Reward Agent output and paper takeaway

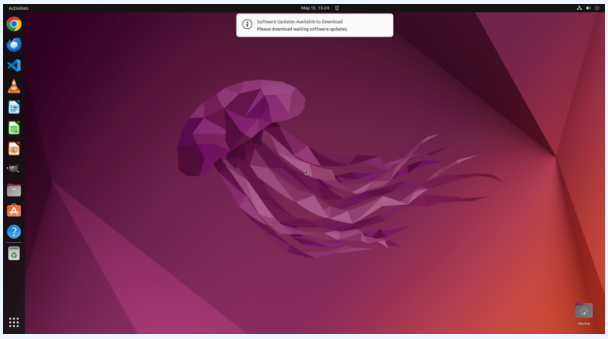
Reward Agent returned Success with reward 1.00. The model matched the broad concept of safety browsing but not the exact required tier.

False negative: exact theme identity over a functional dark-theme criterion I would like to ensure that every time I open the Terminal, the background color is set to Solarized Dark theme instead of default. Please guide me to set it permanently.

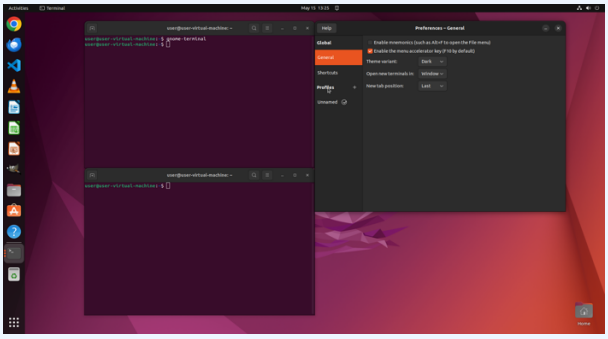
case: terminal_solarized_dark | model: gpt-5.5 | steps: 7 | category: Over-literal interpretation | Reward Agent: Failure (0.00) | GT: True (1.00)

Screenshots provided to the reward model

VLM can see the start/end desktop state, but not hidden file, config, post-execution configuration state, or application state.



A. Initial terminal preferences state



B. Attempted end state

[Task reading] What the task should require

For this task family, the accepted success condition is that Terminal opens in a dark theme permanently; exact Solarized palette matching is not required by the oracle.

[Error mechanism] Why the Reward Agent judgment is wrong

Reward Agent correctly inspected gsettings and found the app theme was dark but the profile palette was not exactly Solarized Dark. The failure is over-specific interpretation

of a flexible user request.

[Key verification steps] Evidence used by the judge

1. **Step 1-3**: It tried dconf dumps and treated empty output as inconclusive.
2. **Step 4-5**: It used gsettings to inspect the profile and theme settings.
3. **Final**: It returned Failure because the profile was not exact Solarized Dark.

[Final judgment] Reward Agent output and paper takeaway

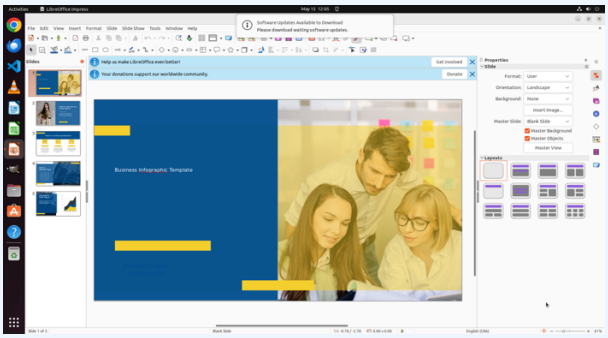
Reward Agent returned Failure with reward 0.00. The model enforced a stricter theme identity than the benchmark's effective requirement.

Thresholded partial: title formatting correct but scope over-expanded Bold the text on slide 1. Make the title of size 44pt and underline it on slide 1. The file to be evaluated is located on the VM at the following path: "/home/user/Desktop/39_2.pptx".

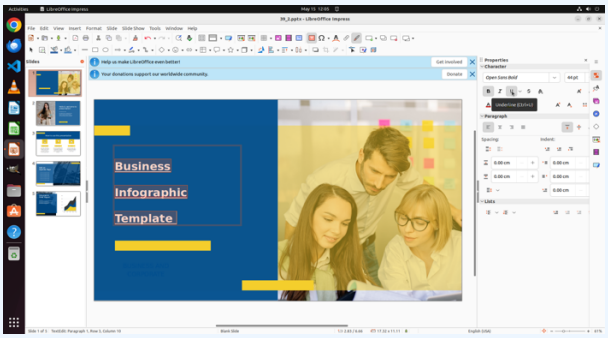
case: ppt_bold_title_scope | model: gpt-5.5 | steps: 3 | category: Scope over-expansion | Reward Agent: Partial Success (0.67) | GT: True (1.00)

Screenshots provided to the reward model

VLM can see the start/end desktop state, but not hidden file, config, post-execution configuration state, or application state.



A. Initial Impress state



B. Formatted slide state

[Task reading] What the task should require

The likely intended focus is the slide title formatting plus visible slide text; the oracle accepts the edited PPTX as matching the gold file.

[Error mechanism] Why the Reward Agent judgment is wrong

Reward Agent inspected PPTX XML and correctly found the title bold/44pt/underlined, but also found another text object not bold. The issue is scope expansion in criterion parsing.

[Key verification steps] Evidence used by the judge

1. **Step 1**: It opened slide XML for file-backed formatting evidence.
2. **Final**: It gave 0.667 because one non-title text object was not bold.

[Final judgment] Reward Agent output and paper takeaway

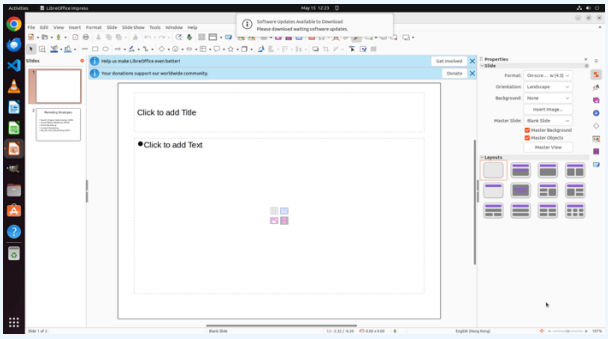
Reward Agent returned Partial Success with reward 0.67. The model interpreted 'text on slide 1' more broadly than the evaluator and penalized an extra text box.

Thresholded partial: correct image and size, strict coordinate criterion Insert the image 'none.png' from the Desktop onto slide 2 at the top-left corner with size 4cm*3cm.

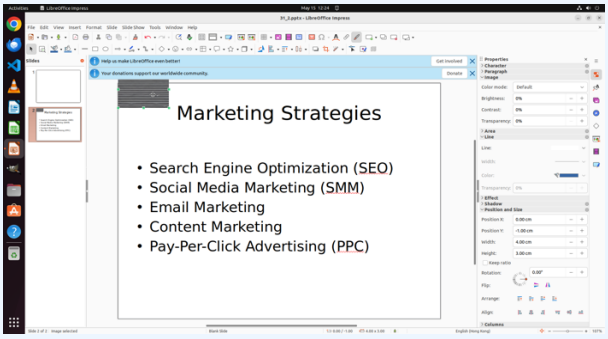
case: ppt_image_position_strict | model: gpt-5.5 | steps: 5 | category: Over-strict spatial criterion | Reward Agent: Partial Success (0.67) | GT: True (1.00)

Screenshots provided to the reward model

VLM can see the start/end desktop state, but not hidden file, config, post-execution configuration state, or application state.



A. Initial slide state



B. Inserted image selected in Impress

[Task reading] What the task should require

Insert the requested image on slide 2 at top-left with 4 cm by 3 cm size. Depending on UI coordinate conventions, a near-edge placement may still satisfy the practical goal.

[Error mechanism] Why the Reward Agent judgment is wrong

Reward Agent compared embedded-image hash and EMU size successfully, then penalized ‘y=-1cm‘. This is a strict spatial interpretation rather than failure to verify the artifact.

[Key verification steps] Evidence used by the judge

1. **Step 1**: It located the PPTX and source image files.
2. **Step 2-3**: It inspected slide XML and compared image hashes.
3. **Final**: It returned 0.667 because the vertical offset was -1 cm.

[Final judgment] Reward Agent output and paper takeaway

Reward Agent returned Partial Success with reward 0.67. The model validated the image and size exactly but treated a small coordinate offset as decisive.