VeriCUA: Learning Proactive Verification for Reliable GUI Agent Rewards

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement learning for computer-use agents increasingly relies on proactive reward agents that interact with the post-execution GUI environment to verify task completion beyond what recorded screenshots reveal. The reliability of such verification hinges on two coupled decisions, namely what to check and when to stop. Existing methods leave these decisions to prompted foundation models, yet neither model scale nor few-shot examples help, as prompted Judges ranging from 4B models to GPT-5.4 all reach between 68% and 70% macro-average agreement. These decisions are hard to prompt because what a check yields and when the evidence suffices depend on the environment rather than the task, and become apparent only once verification is executed. Learning them from the environment is nevertheless difficult, since verification starts from the Actor's terminal state, which cannot be faithfully replicated during training, so each state admits a single attempt. We therefore introduce VeriCUA, which makes each single attempt informative and compares attempts across terminal states. A trainable Judge decides what to check and when to stop, while a frozen Probe executes the checks in the live environment. The Judge is trained with online reinforcement learning on one verification trajectory per terminal state, with a reward reflecting both verdict correctness and Probe usage and returns normalized across terminal states. Trained only on OSWorld and evaluated without further training on MobileWorld and WebArena-Verified, VeriCUA attains the highest reward agreement on all three benchmarks and lies on the agreement–cost Pareto frontier among proactive verifiers, outperforming few-shot GPT-5.4 by 5.36 macro-average points with 47.6%–59.8% fewer Probe tokens. As the reward for policy training, it raises task success by 3.21 points overall and 5.40 points on held-out multi-application tasks.

1 Introduction

Training computer-use agents (CUAs) with reinforcement learning requires reliable rewards for task completion (Lv et al., 2026; Lai et al., 2025; Wang et al., 2026). Rule-based rewards are precise but brittle and difficult to scale (Xie et al., 2024; El hattami et al., 2025; Kong et al., 2026), while passive model-based rewards are more flexible but can only verify what is exposed in the available observations (Qi et al., 2025; Bai et al., 2024; Wang et al., 2025; Yang et al., 2025). Decisive evidence, such as the contents of a saved file or other hidden application state, may therefore remain unobserved (Cai et al., 2025). Proactive verification addresses this limitation by interacting with the post-execution environment to acquire the missing evidence before issuing a verdict (Dai et al., 2026; Cui et al., 2026; Shi et al., 2026). Because such verification is performed for every training rollout, it must be not only accurate but also efficient enough to avoid invoking an expensive model for each reward.

Interaction alone, however, does not guarantee a reliable verdict. Reliability is governed by the verification strategy, which comprises two coupled decisions left unspecified by the task, namely **what to check** and **when to stop**. For a task that requires notes on every slide, a single slide without notes suffices to establish failure and terminate the inspection (Figure 1(b)). The prompted verifier in Figure 1(a) instead requests an exhaustive slide-by-slide comparison and continues scrolling and switching slides without reaching a verdict. This deficiency is remedied neither by model scale nor by in-context examples. Under an identical verification loop, prompted Judges ranging from a 4B model to GPT-5.4 all attain between 68% and 70% macro-average agreement, while few-shot ex-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

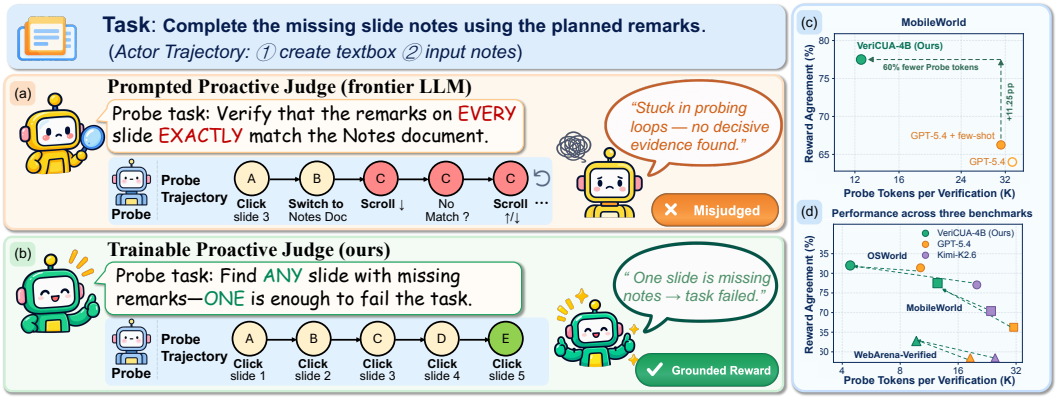

Figure 1: **Learned check selection and stopping make proactive verification reliable, and also cheaper.** Given the same task and frozen PROBE, (a) prompted GPT-5.4 requests an exhaustive comparison and remains inconclusive, whereas (b) VERICUA searches for a slide with missing notes and stops once this counterexample establishes task failure. (c, d) It attains higher agreement with fewer Probe tokens than few-shot GPT-5.4 and Kimi-K2.6.

amples improve agreement by at most 0.6 points and even reduce that of Kimi-K2.6 by 1.0 point (Table 1). Scaling the Judge thus increases cost without improving reliability.

This limitation stems from how prompted Judges make these decisions. They reason over the task description and the recorded screenshots, which reveals what a plausible check looks like but not what evidence it will yield. What a check yields depends on where the evidence resides, the state left by the Actor, and what the PROBE can reach, while the appropriate time to stop depends on how often verdicts based on the screenshots alone prove correct. Neither can be inferred from the task, which explains why scale and examples offer little benefit. The verification strategy must instead be learned by executing it and comparing its verdicts against true outcomes.

Learning from the environment is, however, challenging because of where verification begins. An Actor starts from a resettable, task-specified initial state, allowing RL to compare multiple rollouts from the same state (Shao et al., 2024). A GUI verifier, by contrast, starts from the Actor's terminal state, which results from a single stochastic execution. This state cannot be faithfully replicated during training, since re-execution may not reproduce it, identical screenshots do not guarantee identical hidden state, and probing itself alters it. Each terminal state therefore admits only a single verification attempt, which precludes comparing attempts from the same state.

We introduce VERICUA, which makes each single attempt informative and compares attempts across terminal states. A trainable JUDGE decides what to check and when to stop, while a frozen GUI PROBE executes the checks, so that outcome differences reflect the JUDGE's decisions and the PROBE can be replaced without retraining. The JUDGE is trained with online RL on complete attempts graded by executable outcomes, which grounds each check in the evidence it yields. Its reward also accounts for PROBE usage, so that the same attempt indicates whether stopping was appropriate. Returns are normalized across terminal states within each batch (Hu et al., 2025), obviating environment forking, and initialization on outcome-filtered teacher traces ensures every attempt is a complete verification.

We evaluate VERICUA on OSWorld (Xie et al., 2024), MobileWorld (Kong et al., 2026), and WebArena-Verified (El hattami et al., 2025). **First, the verification strategy matters more than model scale.** VERICUA surpasses few-shot GPT-5.4 by 5.36 macro-average points while consuming 47.6%–59.8% fewer PROBE tokens, placing it on the agreement–cost Pareto frontier (Figure 4). Even VERICUA-1B exceeds prompted GPT-5.4 (OpenAI, 2026) and its teacher Kimi-K2.6 (Kimi Team, 2026) in macro-average agreement, making proactive rewards affordable with small models. **Second, the learned strategy generalizes.** Trained only on OSWorld, the JUDGE achieves the highest agreement on unseen mobile and web environments, as the behaviors it acquires, such as specifying where evidence resides, are not tied to the training environment. **Third, the learned rewards benefit policy learning.** VERICUA rewards improve downstream Pass@4 by 3.21 points overall and by 5.40 points on held-out MultiApp tasks.

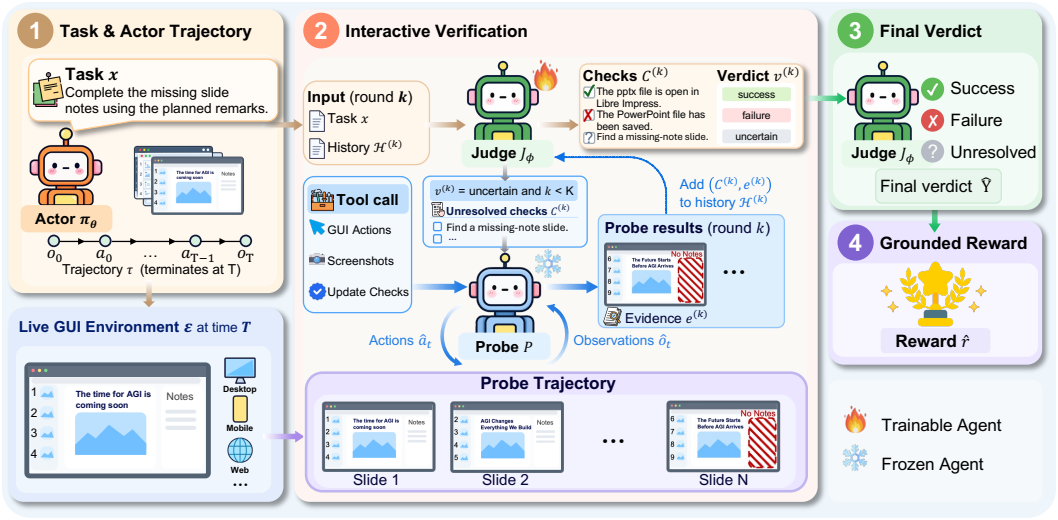

Figure 2: Overview of VERICUA. The JUDGE formulates task-success checks and assesses whether the available evidence supports a terminal verdict. When further inspection is requested, the frozen PROBE gathers evidence for unresolved checks in the live environment and returns status updates with supporting observations.

2 METHOD

2.1 OVERVIEW

As shown in Figure 2, VERICUA verifies the Actor's task outcome using its execution history and live terminal environment. It consists of a trainable JUDGE and a frozen GUI PROBE. The JUDGE initially formulates task-success checks and marks each as satisfied, violated, or unresolved based on the available evidence. If the evidence supports a success or failure verdict, verification terminates. Otherwise, subject to the probing budget, the PROBE gathers evidence for the unresolved checks and returns status updates with supporting observations. Subsequent JUDGE calls retain the checks from the previous round and update their statuses using the returned evidence.

By specifying what the PROBE is asked to inspect and deciding whether to request further interaction, the JUDGE determines the evidence acquired and hence the verdict. Since each Actor terminal state cannot be faithfully replicated during training and thus yields a single verification attempt, we train the JUDGE on complete verification attempts using feedback on verdict correctness and PROBE use, so that each attempt informs both what to check and when to stop.

2.2 THE JUDGE-PROBE VERIFICATION LOOP

Consider an Actor policy π_θ attempting task x in a live GUI environment $\mathcal{E}$ and terminating at time T with trajectory $\tau = (o_0, a_0, \ldots, a_{T-1}, o_T)$, where o_t and a_t denote GUI observations and actions. The JUDGE receives x and an input representation $\psi(\tau)$ containing the Actor's action history and its most recent screenshots (Appendix H). When the JUDGE requests additional evidence, the frozen PROBE interacts with the live terminal environment to inspect unresolved task conditions and returns status updates with supporting observations. Each round therefore involves the two decisions that make up the verification strategy: whether the current evidence supports a terminal verdict and, if not, which checks to send to the PROBE.

JUDGE. Let $\mathcal{H}^{(k)}$ denote the verification history available to the JUDGE at call k, consisting of the initial Actor input and the checks and PROBE evidence retained from earlier rounds. At call $k \in \{0, \ldots, K\}$, where K is the maximum number of PROBE invocations, the trainable JUDGE J_ϕ conditions on the task x and $\mathcal{H}^{(k)}$ to produce a verdict and status-annotated checks:

$$\left(v^{(k)}, C^{(k)}\right) = J_\phi\left(x, \mathcal{H}^{(k)}\right), \qquad \mathcal{H}^{(0)} = \psi(\tau). \tag{1}$$

Here $v^{(k)} \in \{\texttt{success}, \texttt{failure}, \texttt{uncertain}\}$, and $C^{(k)}$ pairs each check with a status: satisfied, violated, or unresolved. The check text is generated at $k = 0$ and the statuses are updated thereafter. The JUDGE returns `success` when all checks are satisfied, `failure` when any is violated, and `uncertain` otherwise; a success or failure verdict terminates verification. When $v^{(k)} = \texttt{uncertain}$ and $k < K$, only the unresolved subset $C_u^{(k)} \subseteq C^{(k)}$ is passed to the PROBE.

PROBE. The frozen PROBE P gathers evidence for the unresolved checks in the current live environment; keeping it fixed lets outcome differences reflect the JUDGE's decisions. Its instructions confine it to inspecting the environment, without completing or repairing the Actor's task (Appendix H). Its interaction trajectory and returned evidence are represented as

$$\left(\tau_P^{(k)}, e^{(k)}\right) \sim P\Big(\cdot \mid C_u^{(k)}; \mathcal{E}^{(k)}\Big), \qquad \mathcal{H}^{(k+1)} = \mathcal{H}^{(k)} \oplus \left(C^{(k)}, e^{(k)}\right). \tag{2}$$

Here $\mathcal{E}^{(0)}$ is the Actor's terminal environment, and each subsequent PROBE invocation continues from the state reached by the preceding PROBE actions. The returned evidence $e^{(k)}$ comprises criterion-status updates and selected supporting observations, and $\oplus$ denotes concatenation. Only $e^{(k)}$ reaches the JUDGE; the full PROBE trace $\tau_P^{(k)}$ does not, and each invocation starts without the interaction history of earlier rounds, so neither component's context grows with the other's execution length.

2.3 OUTCOME FEEDBACK FROM A SINGLE ATTEMPT

We assign a terminal reward to each complete verification attempt, combining verdict correctness with whether probing was used. A reward based only on correctness scores a correct verdict reached directly the same as one reached after probing; with one attempt per terminal state, the reward itself must therefore indicate whether the JUDGE stopped at the right time.

Let $\hat{Y}_i$ denote the final JUDGE verdict and Y_i the executable outcome recorded before PROBE interaction. We set $p_i = 1$ if the verification attempt invokes the PROBE at least once, and $p_i = 0$ otherwise, so the probing term of the reward reflects the JUDGE's stopping decision rather than the frozen PROBE's execution length. The terminal reward is $r_i = R(\hat{Y}_i, Y_i, p_i)$.

Correct verdicts receive higher rewards when reached without probing, encouraging reliable judgments from existing evidence. Incorrect verdicts receive larger penalties when issued without probing, discouraging premature judgments. Unresolved or invalid outputs receive the lowest reward, discouraging attempts that exhaust the probing budget without reaching a terminal verdict. The exact reward schedule is given in Appendix F.

We use this terminal reward to jointly train check formulation and stop-or-probe decisions over verification attempts. The reward preferences teach the JUDGE to stop once the evidence suffices, while the shared return trains check formulation through its effect on the evidence collected and the verdict. We next describe how we collect verification trajectories and optimize the JUDGE.

2.4 ONLINE OPTIMIZATION OF THE JUDGE

Initialization. Before online optimization, we initialize the JUDGE by supervised fine-tuning on teacher verification trajectories whose final verdicts match the executable outcomes. These traces provide examples of structured check generation and verdict prediction, so that each one-shot attempt is a complete, valid verification rather than wasted on malformed outputs. The PROBE remains frozen.

Sampling verification trajectories. We run the JUDGE from the live terminal environment of each Actor rollout and execute its PROBE requests to obtain the evidence. Each completed verification attempt receives the terminal reward defined in Section 2.3. Unlike an Actor's resettable initial state, this terminal environment is produced by one stochastic execution: replaying the Actor need not reproduce it, transient state such as unsaved edits and modal dialogs cannot be confirmed from screenshots, and PROBE actions further alter it. We therefore collect one verification trajectory per Actor terminal environment, without environment forking or same-state verifier rollout groups.

Updating the JUDGE. Comparing attempts across environments, we optimize the JUDGE with REINFORCE++ (Hu et al., 2025), normalizing terminal returns across the independent environments in

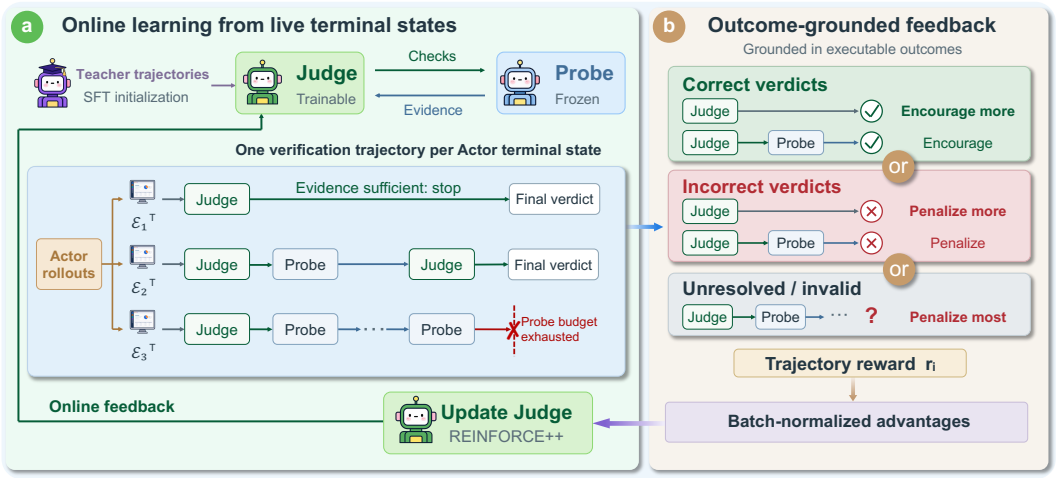

Figure 3: Training procedure of VERICUA. Online optimization uses one verification trajectory per live Actor terminal state. The JUDGE is updated using terminal feedback on verdict correctness and whether probing was used, while the PROBE remains frozen.

each batch. Let $z_i = \big((v_i^{(0)}, C_i^{(0)}), e_i^{(0)}, \ldots, (v_i^{(K_i)}, C_i^{(K_i)})\big)$ denote a complete verification trajectory with $K_i \le K$ PROBE invocations and terminal reward r_i. When $K_i = 0$, the trajectory contains only the initial JUDGE output. For a batch $\mathcal{B}$ of complete verification trajectories, the normalized return supplies the advantage for every active JUDGE token:

$$A_{i,k,t} = \frac{r_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \qquad (i,k,t) \in \mathcal{A}_{\mathcal{B}}. \tag{3}$$

Here $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ are the mean and variance of terminal rewards in the batch, and $\epsilon > 0$ ensures numerical stability. The index set $\mathcal{A}_{\mathcal{B}}$ contains active JUDGE tokens, indexed by trajectory i, JUDGE call k, and token position t. Only newly generated JUDGE tokens contribute to the loss; context tokens and PROBE outputs are excluded. The following log-probability surrogate expresses the on-policy gradient at the rollout policy:

$$\mathcal{L}_{\text{Judge}}(\phi) = -\frac{1}{|\mathcal{A}_{\mathcal{B}}|} \sum_{(i,k,t)\in\mathcal{A}_{\mathcal{B}}} A_{i,k,t} \log \pi_\phi^J \left(y_{i,k,t} \mid h_{i,k,t}\right), \tag{4}$$

where π_ϕ^J is the autoregressive JUDGE policy, $y_{i,k,t}$ is the generated token, and $h_{i,k,t}$ is its conditioning context. The loss averages over active JUDGE tokens across the batch. For parameter updates, we use the clipped likelihood-ratio objective described in Appendix E, with rollout-policy probabilities and normalized returns held fixed.

Using the trained verifier. After training, we freeze VERICUA and map its final verdict to a binary reward $\hat{r}_i = \mathbb{I}\big[\text{VERICUA}(x_i, \tau_i, \mathcal{E}_i^T) = \texttt{success}\big]$, where $\mathcal{E}_i^T$ is the terminal environment of Actor rollout i. We keep VERICUA fixed and optimize the Actor with vanilla GRPO (Shao et al., 2024), grouping Actor rollouts generated from the same task initialization; each rollout is verified once from its own terminal environment, and the resulting verdict determines its reward $\hat{r}_i$.

3 EXPERIMENTS

Our experiments examine whether a learned verification strategy improves reward agreement beyond prompted verifiers of any scale and at what evidence-acquisition cost, whether it generalizes, and whether the resulting rewards improve downstream Actor learning.

3.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate on the OSWorld (Xie et al., 2024) tasks that require neither network access nor Google Drive, the GUI-only subset of MobileWorld (Kong et al., 2026), and

Table 1: Reward reliability across executable environments (%). Results use Avg@4 except ZeroGUI*, which follows its official Majority@4 aggregation. VERICUA leads on all three benchmarks; bold and underlined mark the best and second-best results.

Verifier	Mode	OSWorld desktop	MobileWorld mobile	WebArena-V web	Macro Avg. all envs.
GPT-5.4					proprietary reference
Prompted VERICUA	Prompted	<u>81.88</u>	64.00	58.50	68.13
+ few-shot	Prompted	81.40	66.25	58.50	68.72
Kimi-K2.6					proprietary reference
Prompted VERICUA	Prompted	75.85	74.00	59.00	69.62
+ few-shot	Prompted	77.05	70.25	58.50	68.60
Qwen3-VL-4B-Thinking					4B main backbone
DistRL	Passive	61.78	68.00	57.75	62.51
DigiRL	Passive	63.71	68.25	57.75	63.24
ZeroGUI*	Passive	71.50	73.00	51.00	65.17
VAGEN	Prompted	65.34	70.25	56.25	63.95
Prompted VERICUA	Prompted	73.73	73.50	61.50	69.58
VERICUA (ours)	Trained	**82.00**	**77.50**	**62.75**	**74.08**
Δ vs. Prompted (pp)		+8.27	+4.00	+1.25	+4.50
Qwen3-VL-2B-Thinking					2B main backbone
Prompted VERICUA	Prompted	66.49	65.00	55.00	62.16
VERICUA (ours)	Trained	77.84	72.50	62.00	70.78
Δ vs. Prompted (pp)		+11.35	+7.50	+7.00	+8.62
MiniCPM-V-4.6-Thinking					1B main backbone
Prompted VERICUA	Prompted	52.29	58.75	40.50	50.51
VERICUA (ours)	Trained	77.11	<u>75.50</u>	<u>62.25</u>	<u>71.62</u>
Δ vs. Prompted (pp)		+24.82	+16.75	+21.75	+21.11

WebArena-Verified Hard (El hattami et al., 2025), using 414, 100, and 100 trajectories, respectively. These cover successful executions and diverse failures from five Actors: EvoCUA-32B (Xue et al., 2026), Qwen3.8-27B (Qwen Team, 2026), Holo3.1-35B-A3B (H Company, 2026), Qwen3-VL-4B-Thinking (Bai et al., 2025), and Gemini 3.7 Flash (Google DeepMind, 2026). Each trajectory includes the action–observation history, the clean task snapshot from which its terminal environment can be re-executed, and an executable outcome label hidden from all verifiers.

Training data. We train the JUDGE only on OSWorld and keep it frozen thereafter, so MobileWorld and WebArena-Verified test zero-shot transfer; each environment uses one frozen PROBE shared by all VERICUA variants. We denote trained Judges by size: VERICUA-4B (Qwen3-VL-4B-Thinking), VERICUA-2B (Qwen3-VL-2B-Thinking), and VERICUA-1B (MiniCPM-V-4.6-Thinking (Yu et al., 2026; Yao et al., 2024)). SFT uses Kimi-K2.6 verification traces and online RL uses newly collected OSWorld trajectories, all disjoint from evaluation (Appendix F).

Baselines. Unless noted, all baselines use Qwen3-VL-4B-Thinking, and existing methods follow their published observation and prompting configurations (Appendix D). **(i) Passive verifiers:** DistRL (Wang et al., 2025), DigiRL (Bai et al., 2024), and ZeroGUI (Yang et al., 2025). **(ii) Prompted proactive verifiers:** VAGEN (Cui et al., 2026) and prompted VERICUA, which shares trained VERICUA's JUDGE–PROBE architecture, prompts, and budget of at most $K=2$ PROBE rounds but uses an off-the-shelf Judge. We also evaluate prompted VERICUA with the backbones of VERICUA-2B and VERICUA-1B, and with proprietary Kimi-K2.6 and GPT-5.4, both also tested few-shot.

Evaluation protocol. Both evaluations use the executable task evaluator as ground truth and aggregate four independent runs (Appendix A). **Verifier evaluation:** We keep only Actor trajectories whose outcome is stable across six re-executions from the clean task snapshot (Appendix D); each proactive-verifier trial then runs in its own re-executed terminal environment, scored against the same outcome label, so no verifier's PROBE actions can affect another's, while passive verifiers score the recorded screenshots. We report Avg@4 agreement per benchmark and its unweighted mean (Macro Avg.), and measure cost as frozen-PROBE tokens per judgment (Appendix D). **Actor evaluation:** We train the same Qwen3-VL-4B-Thinking Actor on OSWorld with vanilla GRPO (Shao et al., 2024) under identical tasks, rollout budget, and hyperparameters, varying only the frozen reward source (Ze-

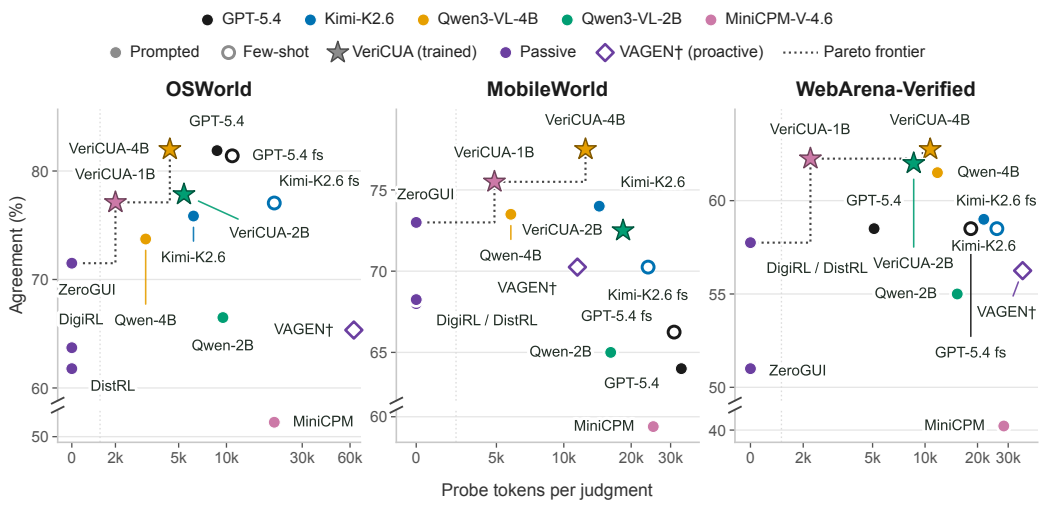

Figure 4: Reward agreement versus Probe tokens per judgment: trained VeriCUA occupies the agreement–cost Pareto frontier among proactive verifiers. VAGEN† is billed for its full loop.

roGUI, prompted VeriCUA, or trained VeriCUA), and report Pass@4 on the 206 single-application tasks used for Actor training (ID) and 74 MultiApp tasks held out from Actor training (OOD).

3.2 Verification Strategy Matters More Than Scale

Agreement and Probe-token cost. Prompted Judges from a 4B model to GPT-5.4 plateau at 68.13%–69.62% macro-average agreement, and few-shot examples shift it by at most 1.02 points. Table 1 and Figure 4 show that VeriCUA-4B achieves the highest reward agreement on all three benchmarks and lies on the agreement–cost Pareto frontier of proactive verifiers, the class that must pay for interaction. Compared with few-shot GPT-5.4, it improves macro-average agreement by 5.36 points (68.72% $\rightarrow$ 74.08%) with 47.6%–59.8% fewer Probe tokens; on OSWorld, where GPT-5.4 is strongest, it still leads with 56.9% fewer, and its margin widens to 11.25 and 4.25 points on MobileWorld and WebArena-Verified. Even VeriCUA-1B (71.62%) exceeds every prompted Judge.

Effect of the learned verification policy. With the Judge backbone, Probe, prompts, and round budget held fixed, training improves agreement on every benchmark, yielding macro-average gains of 4.50, 8.62, and 21.11 percentage points over the prompted Judges for the 4B, 2B, and 1B models, respectively; the gains therefore come from the learned verification policy rather than the architecture. The agreement–cost frontier reflects this: on every benchmark the highest agreement on the frontier is reached by a trained VeriCUA Judge, and VeriCUA-1B stays on the frontier throughout, dominating every prompted counterpart in both agreement and cost (Figure 4).

3.3 Generality and Behavior of the Learned Strategy

Transfer across environments. On MobileWorld and WebArena-Verified, which are unseen during training, VeriCUA-4B and VeriCUA-1B paired with environment-specific Probes rank first and second, ahead of every prompted and passive verifier, including few-shot GPT-5.4, so the learned strategy is not specific to desktop applications.

Contribution of training stages. Figure 5(a) separates SFT initialization from subsequent online RL for the Qwen3-VL-4B-Thinking Judge on OSWorld. Overall agreement increases from 73.73% for the prompted Judge to 79.41% after SFT and 82.00% after online RL. Both stages therefore contribute to the final result, with online RL adding 2.59 percentage points beyond SFT initialization. The reported scores for judgments with and without probing also improve. Notably, every trained Judge surpasses Kimi-K2.6 (69.62%), whose traces initialize it, so a single attempt per terminal state suffices to learn beyond imitation.

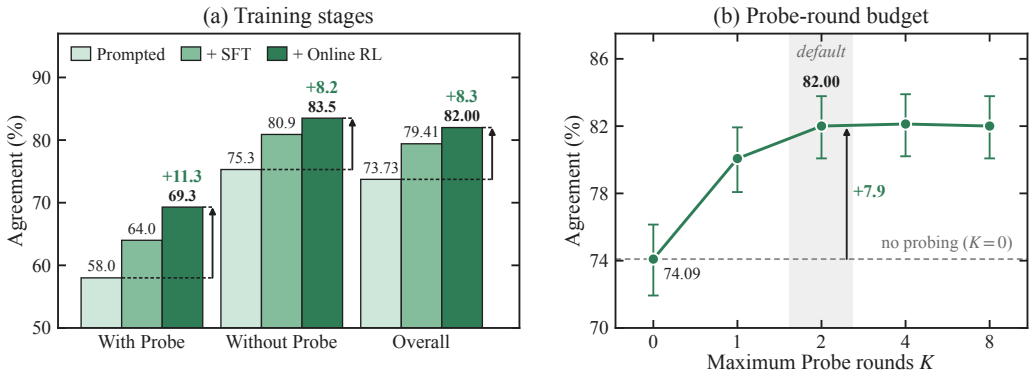

Figure 5: Learning ablations on OSWorld. **(a)** SFT improves all judgments; online RL mainly lifts those that invoke the PROBE. **(b)** Agreement saturates at $K{=}2$, with most gain from the first round.

Effect of the probing budget. Figure 5(b) varies the maximum number of PROBE rounds for the trained 4B Judge. Agreement increases from 74.09% at $K{=}0$ to 82.00% at $K{=}2$, confirming that evidence from the live environment is decisive; the first round accounts for most of the gain, and two rounds suffice. The cap limits available interaction; the JUDGE can still terminate before using it.

Robustness to PROBE size. We keep the trained JUDGE fixed and evaluate Holo3.1 Probes at 35B-A3B, 9B, and 4B scales. OSWorld agreement ranges from 82.00% to 82.73% across these configurations without retraining the JUDGE (Appendix C), so the learned strategy resides in the JUDGE.

Evidence acquisition behavior. We further examine 33 OSWorld cases in which the recorded Actor trajectory omits outcome-determining evidence that remains accessible through GUI interaction (Appendix B). With the same frozen PROBE, VERICUA-4B directs 61.9% of PROBE actions toward task-relevant evidence, compared with 47.6% for prompted GPT-5.4, and aimless scrolling and waiting fall from 16.2% to 5.5%, with fewer repeated targets and dead-end probes. Figure 6 shows why: verifying that Presenter Console is disabled requires a setting absent from the recorded trajectory. VERICUA's check names both the condition and its location (`Tools` → `Options`), and the PROBE inspects it in two steps; given the same condition without the location, the PROBE takes nine steps without finding it.

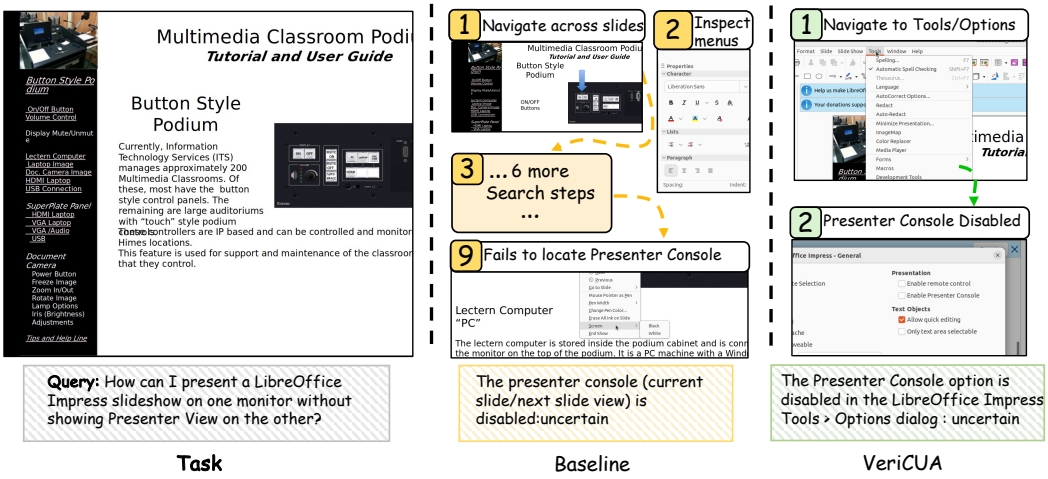

Figure 6: Case study: VERICUA's check names where the decisive evidence lives, so the PROBE settles it in two steps. Baseline: the same PROBE given a check stating the condition but not its location.

3.4 Downstream Actor Learning

Actor performance. Using VeriCUA rewards yields the highest Pass@4 among the compared reward sources on every split (Table 2). Overall Pass@4 rises from 43.93% for the initial Actor to 47.14%, a gain of 3.21 percentage points. On the MultiApp split, it rises from 17.57% to 22.97%, a gain of 5.40 percentage points. The Actor trained with prompted VeriCUA reaches 45.71% overall and 20.27% on MultiApp, below the corresponding results with the trained verifier.

Verification during Actor training. We examine the verification behavior in the runs using prompted and trained VeriCUA rewards (Detailed in Appendix G). The run using the trained verifier has higher agreement with executable outcomes (81.4% versus 77.8%), a lower false-negative rate (4.6% versus 10.0%), and a lower Probe-attempt rate (4.28% versus 5.70%). The trained verifier thus probes less often while missing fewer successes, reflecting a reward that scores both verdict correctness and Probe use.

Table 2: Downstream Policy performance with different reward sources (Pass@4, %). Using VeriCUA as the reward provider yields the best performance across all splits.

Policy	Reward source	All	ID	OOD
Initial Policy	–	43.93	53.40	17.57
Policy + Reward	ZeroGUI	45.36	54.37	20.27
Policy + Reward	Prompted VeriCUA	45.71	54.85	20.27
Policy + Reward	VeriCUA (ours)	**47.14**	**55.83**	**22.97**

4 Related Work

Reward signals for computer-use agents. Task-completion rewards for computer-use agents can be obtained from executable evaluators or model-based judgments. Executable evaluators derive outcome labels from application or backend state (Xie et al., 2024; Kong et al., 2026; El hattami et al., 2025), relying on task-specific checking logic. Environment synthesis automates the construction of verifiable tasks by generating resettable tasks together with executable evaluators (Cao et al., 2026; Zhang et al., 2026b). Model-based evaluators assess recorded observations and action histories using prompted or trained judges (Bai et al., 2024; Wang et al., 2025; Qi et al., 2025; Yang et al., 2025; Xia et al., 2025), while rubric and milestone-based critics further structure this assessment (Rosset et al., 2026; Li et al., 2026). However, since recorded observations obscure uncaptured state, verifiers must acquire additional evidence through active interaction with the environment.

Interactive verification and evidence acquisition. Recent work incorporates evidence acquisition into both post-execution verification and task execution. Post-execution verifiers continue from the Actor's terminal environment and inspect state before issuing a verdict (Dai et al., 2026; Cui et al., 2026; Shi et al., 2026). Other approaches integrate evidence collection, completeness checking, or recovery into the acting policy (Cai et al., 2025; Han et al., 2026). Text-reasoning verifiers can be trained with RL by replicating a problem to compare verification attempts (Zhang et al., 2026a); we study the post-execution GUI setting, where each state yields a single verification attempt.

5 Conclusion

We introduced VeriCUA, a proactive GUI reward agent that learns what to check and when to stop. Prompted verifiers benefit little from larger models or few-shot examples because a check's value is revealed only through environment interaction. Despite having only one verification attempt per Actor terminal state during training, VeriCUA learns this strategy by comparing outcomes across states. Across desktop, mobile, and web, it reaches the agreement–cost Pareto frontier, enables a 1B Judge to surpass prompted frontier models, and improves downstream policy RL. These results identify verification strategy, rather than model scale, as the key to reliable and affordable GUI reward supervision.

Reproducibility Statement

The interactive verification procedure and online training objective are specified in Sections 2.2 and 2.4. Section 3.1 describes the benchmark splits, baselines, and re-execution-based evaluation protocol. The appendix provides evaluation metrics, training and inference settings, the terminal reward schedule, and complete Judge and Probe prompts. Upon acceptance, we will release the VeriCUA implementation and scripts for training and evaluation.

AI Use Statement

Large language model (LLM) tools were used in a supporting role to assist with coding and debugging in the reinforcement-learning framework, and to polish the manuscript's writing for clarity and readability. Kimi-K2.6 was also used to generate candidate verification trajectories for the rejection-sampled SFT cold-start dataset. LLM tools were not used for brainstorming or generating research ideas. All AI-assisted textual edits were reviewed by the authors, and all AI-assisted code changes were reviewed and tested. The authors take full responsibility for the final content of this paper.

References

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. DigiRL: Training in-the-wild device-control agents with autonomous reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 12461–12495, 2024.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report, 2025. arXiv:2511.21631.

Shao-Fei Cai, Yulei Qin, Haojia Lin, Zi-Han Xu, Gang Li, Yuchen Shi, Zongyi Li, Yong Mao, Si-Qi Cai, Xiaoyu Tan, Yitao Liang, Ke Li, and Xing Sun. SmartSnap: Proactive evidence seeking for self-verifying agents, 2025. arXiv:2512.22322.

Yuan Cao, Dezhi Ran, Mengzhou Wu, Yuzhe Guo, Xin Chen, Ang Li, Gang Cao, Gong Zhi, Hao Yu, Linyi Li, Wei Yang, and Tao Xie. GUI-GENESIS: Automated synthesis of efficient environments with verifiable rewards for GUI agent post-training, 2026. arXiv:2602.14093.

Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. Agentic reward modeling: Verifying GUI agent via progressive trajectory-grounded interaction, 2026. arXiv:2602.00575.

Gaole Dai, Shiqi Jiang, Ting Cao, Yuqing Yang, Yuanchun Li, Rui Tan, Mo Li, and Lili Qiu. ProRe: A proactive reward system for GUI agents via reasoner–actor collaboration. In *Proceedings of the 14th International Conference on Learning Representations*, 2026.

Amine El hattami, Megh Thakkar, Nicolas Chapados, and Christopher Pal. WebArena Verified: Reliable evaluation for web agents. In *Workshop on Scaling Environments for Agents*, 2025. URL https://openreview.net/forum?id=94tlGxmqkN.

Google DeepMind. Gemini 3.7 Flash model card, 2026. URL https://deepmind.google/models/model-cards/gemini-3-7-flash/.

H Company. Holo3.1: Fast & local computer use agents, 2026. URL https://huggingface.co/Hcompany/Holo-3.1-35B-A3B.

Qijun Han, Haoqin Tu, Zijun Wang, Haoyue Dai, Yiyang Zhou, Nancy Lau, Alvaro A. Cardenas, Yuhui Xu, Ran Xu, Caiming Xiong, Zeyu Zheng, Huaxiu Yao, Yuyin Zhou, and Cihang Xie. VLAA-GUI: Knowing when to stop, recover, and search, a modular framework for GUI automation, 2026. arXiv:2604.21375.

Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. REINFORCE++: Stabilizing critic-free policy optimization with global advantage normalization, 2025. arXiv:2501.03262.

Kimi Team. Kimi K2.6: Advancing open-source coding. https://www.kimi.ai/blog/kimi-k2-6, 2026.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. MobileWorld: Benchmarking autonomous mobile agents in agent-user interactive and MCP-augmented environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6142–6167, 2026.

Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. ComputerRL: Scaling end-to-end online reinforcement learning for computer use agents, 2025. arXiv:2508.14040.

Zehao Li, Zhenyu Wu, Yibo Zhao, Bowen Yang, Jingjing Xie, Zhaoyang Liu, Zhoumianze Liu, Kaiming Jin, Jianze Liang, Zonglin Li, Feng Wu, Bowen Zhou, Zun Wang, and Zichen Ding. OS-Themis: A scalable critic framework for generalist GUI rewards, 2026. arXiv:2603.19191.

Bowen Lv, Xiao Liu, Yanyu Ren, Hanyu Lai, Bohao Jing, Hanchen Zhang, Yanxiao Zhao, Shuntian Yao, Jie Tang, and Yuxiao Dong. SCALECUA: Scaling computer use agents with verifiable task synthesis and efficient online RL, 2026. arXiv:2607.11185.

OpenAI. GPT-5.4 Thinking System Card. OpenAI, 2026. URL https://openai.com/index/gpt-5-4-thinking-system-card/.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *Proceedings of the 13th International Conference on Learning Representations*, 2025.

Qwen Team. Qwen3.8-27B model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Corby Rosset, Pratyusha Sharma, Andrew Zhao, Miguel Gonzalez-Fernandez, and Ahmed Awadallah. The art of building verifiers for computer use agents, 2026. arXiv:2604.06240.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. arXiv:2402.03300.

Chenrui Shi, Yuwei Wu, Yang Liu, Ruining Feng, Zi-Rui Shang, Zhi Gao, Lifeng Fan, and Che Sun. Interactive reward agent: GUI task evaluation via environment-state verification, 2026. arXiv:2607.25904.

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, Dayiheng Liu, Que Shen, Junyang Lin, and Tao Yu. CUA-Gym: Scaling verifiable training environments and tasks for computer-use agents, 2026. arXiv:2605.25624.

Taiyi Wang, Zhihao Wu, Jianheng Liu, Jianye Hao, Jun Wang, and Kun Shao. DistRL: An asynchronous distributed reinforcement learning framework for on-device control agent. In *Proceedings of the 13th International Conference on Learning Representations*, 2025.

Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. AgentRM: Enhancing agent generalization with reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19277–19290, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, pp. 52040–52094, 2024.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, et al. EvoCUA: Evolving computer use agents via learning from scalable synthetic experience, 2026. arXiv:2601.15876.

Chenyu Yang, Shiqian Su, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, Wenhai Wang, Yu Qiao, Xizhou Zhu, and Jifeng Dai. ZeroGUI: Automating online GUI learning at zero human cost, 2025. arXiv:2505.23762.

Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, et al. MiniCPM-V: A GPT-4V level MLLM on your phone, 2024. arXiv:2408.01800.

Tianyu Yu, Zefan Wang, Chongyi Wang, Fuwei Huang, Wenshuo Ma, Zhihui He, Tianchi Cai, Weize Chen, Yuxiang Huang, Ranchi Zhao, et al. MiniCPM-V 4.5: Cooking efficient MLLMs via architecture, data, and training recipe. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11704–11715, 2026.

Jia-Zheng Zhang, Ziche Fu, Zhiheng Xi, Wenqing Jing, Ming-Xu Chai, Wei He, Guoqiang Zhang, Chenghao Fan, Chenxin An, Wenxiang Chen, Zhicheng Liu, Haojie Pan, Dingwei Zhu, Tao Gui, Qi Zhang, and Xuanjing Huang. AgentV-RL: Scaling reward modeling with agentic verifier, 2026a. arXiv:2604.16004.

Ziyun Zhang, Zezhou Wang, Xiaoyi Zhang, Zongyu Guo, Jiahao Li, Bin Li, and Yan Lu. InfiniteWeb: Scalable web environment synthesis for GUI agent training. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28465–28492, 2026b.

A Evaluation Metrics

For reward evaluation, let $\{(\tau_i, Y_i)\}_{i=1}^{N}$ be a set of Actor trajectories paired with executable evaluator outcomes. We run each verifier independently k times on every trajectory and denote its j-th verdict by $\hat{Y}_{i,j}$. Each proactive-verifier trial starts from a fresh re-execution of an accepted trajectory under the protocol in Section 3.1, while passive verifiers score the fixed recorded observations. We report average agreement with the executable evaluator as

$$\text{Avg@}k = \frac{100}{Nk} \sum_{i=1}^{N} \sum_{j=1}^{k} \mathbb{I}[\hat{Y}_{i,j} = Y_i]. \tag{5}$$

Thus, Avg@k measures reward agreement across repeated verifier judgments rather than task success by the downstream Policy. ZeroGUI follows its official Majority@4 aggregation, as marked in Table 1.

For downstream Policy evaluation, let $\mathcal{D}$ be the task set and let $S_{i,j} \in \{0, 1\}$ indicate whether the j-th Policy rollout on task i succeeds. We report the fraction of tasks solved at least once within k rollouts as

$$\text{Pass@}k = \frac{100}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \mathbb{I}\left[\sum_{j=1}^{k} S_{i,j} > 0\right]. \tag{6}$$

Both metrics are reported as percentages; differences between percentages are reported in points (i.e., percentage points).

B Behavioral Evidence for Targeted Verification

We analyze 33 evidence-required OSWorld cases. In each case, the recorded trajectory leaves outcome-determining state unobserved, while the missing evidence remains accessible through continued GUI interaction. These cases isolate the need for targeted evidence acquisition and evidence-based stopping: identifying an effective verification path and determining when the returned evidence supports a terminal verdict.

We characterize this process through four behavioral proxies (Table 3). These measurements describe how verification interaction is allocated, complementing final reward agreement and the PROBE-budget ablation with evidence about the verification process itself.

What the proxies capture. Direct-evidence probes indicate interaction directed at evidence bearing on task completion, and therefore reflect the specificity of evidence selection. Aimless scrolling and waiting indicate interaction without progress toward such evidence. Repeated targets capture revisiting verification targets, while dead-end probes capture exploration that does not reach useful evidence. Each proxy is measured independently over these PROBE interactions.

Table 3: VERICUA-4B directs more verification toward task-relevant evidence and reduces all three reported forms of unproductive exploration relative to prompted GPT-5.4. Rates are percentages; Δ is VERICUA minus GPT-5.4 in percentage points.

Behavioral proxy	**GPT-5.4**	**VERICUA-4B**	Δ
Direct-evidence probes (%) $\uparrow$	47.6	**61.9**	$+14.3$
Aimless scroll and wait (%) $\downarrow$	16.2	**5.5**	-10.7
Repeated targets (%) $\downarrow$	19.8	**13.6**	-6.2
Dead-end probes (%) $\downarrow$	25.4	**21.7**	-3.7

More focused evidence acquisition. The trained verifier increases the direct-evidence rate by 14.3 points and reduces aimless scrolling and waiting by 10.7 points; repeated targets and dead-end probes also decrease. With the same frozen PROBE, the learned JUDGE directs verification toward more task-relevant evidence.

Together with the broader agreement and interaction gains in Figure 1(c,d), the controlled PROBE-budget ablation in Figure 5(b), and the targeted checks in Figures 1 and 6, these behavioral shifts show that learned verification focuses interaction on evidence that bears directly on task completion.

C REWARD RELIABILITY ACROSS PROBE SCALES

The JUDGE–PROBE decomposition creates a modular scaling axis: the learned JUDGE determines what evidence to seek, while the frozen PROBE executes the request. We hold the trained JUDGE, evaluation task set, and active-verification protocol fixed and vary only the PROBE within the Holo3.1 family, from 35B-A3B to 9B and 4B. Each reported agreement aggregates repeated evaluation trials across multiple random seeds.

Table 4: Reward agreement remains stable across Holo3.1 PROBE scales. The trained JUDGE, evaluation task set, and active-verification protocol are fixed; only the frozen PROBE changes. Confidence intervals are 95% Wilson intervals.

PROBE	**Agreement (%)**	**95% CI**
Holo3.1-35B-A3B	82.00	$[80.08, 83.78]$
Holo3.1-9B	82.73	$[80.83, 84.47]$
Holo3.1-4B	82.07	$[80.14, 83.84]$

Compact Probes preserve reward quality. Agreement stays within 0.73 points across the three Probe models, with closely overlapping confidence intervals and no monotonic dependence on model size. Pairwise two-sample proportion tests give $p = 0.5843$ (35B-A3B vs. 9B), $p = 0.9639$ (35B-A3B vs. 4B), and $p = 0.6158$ (9B vs. 4B). The 4B Probe reaches 82.07%, closely matching the 82.00% agreement of 35B-A3B. Within the Holo3.1 family, VERICUA therefore retains reward reliability across a wide range of PROBE scales. PROBE capacity becomes a deployment choice rather than a prerequisite for reward quality, enabling compact active verification without retraining the JUDGE.

D Evaluation Details

We use fixed inference settings for each component during reward evaluation. The Judge uses a sampling temperature of 1.0. The frozen Probe uses a sampling temperature of 0.0, a maximum output length of 8192 tokens per API request, and at most 15 interaction steps per probing round. The maximum number of probing rounds is specified in the main experimental setup.

Trajectory screening and cost accounting. We re-execute each Actor trajectory six times from the clean task snapshot and retain it only when all six re-executions yield the same executable outcome and the mean pHash similarity over their final three screenshots exceeds 0.95; verifier trials then run in separate fresh re-executions. In cost accounting, trials that never call the Probe count as zero tokens, and image tokens follow Qwen-VL accounting. VAGEN has no separate Probe, so its full active loop is billed. We measure cost in Probe tokens because this is the cost that the verification strategy controls: the Judge decides which checks the Probe executes and when probing stops, and within each benchmark every prompted and trained VeriCUA Judge drives the same frozen Probe, so these tokens are counted on one model with one tokenizer. Judge tokens, by contrast, are not comparable across backbones: tokenizers differ, closed models such as GPT-5.4 report their reasoning only as provider-side counts, and per-token prices differ by orders of magnitude between a locally served 1B–4B model and a proprietary API. Against proprietary Judge models, including Judge cost would only widen the advantage of VeriCUA, whose Judge runs locally at 1B–4B scale.

The Judge is paired with an environment-specific frozen Probe. Within each benchmark, prompted and trained VeriCUA use the same Probe configuration, isolating the effect of learning the verification policy. Table 5 summarizes the default configurations.

Table 5: Default frozen Probe configurations used for reward evaluation.

Benchmark	**Probe**
OSWorld	Holo3.1-35B-A3B
MobileWorld	GUI-Owl-1.5-32B-Think
WebArena-Verified	Holo3.1-35B-A3B

E REINFORCE++ Optimization

We use the single-rollout REINFORCE++ estimator (Hu et al., 2025), as implemented in Slime. It assigns the terminal outcome reward to the generated Judge tokens in a complete verification rollout and normalizes their advantages globally, as in Equation (3). Thus, no same-state rollout group or learned value model is required. The main text states the underlying policy-gradient objective; our implementation uses the standard PPO-style clipped surrogate:

$$\begin{aligned} \rho_{i,k,t}(\phi) &= \frac{\pi_\phi^J(y_{i,k,t} \mid h_{i,k,t})}{\pi_{\text{old}}^J(y_{i,k,t} \mid h_{i,k,t})}, \qquad \bar{\rho}_{i,k,t}(\phi) = \text{clip}(\rho_{i,k,t}(\phi), 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}), \\ \mathcal{L}_{\text{clip}}(\phi) &= -\frac{1}{|\mathcal{A}_\mathcal{B}|} \sum_{(i,k,t)\in\mathcal{A}_\mathcal{B}} \min\{\rho_{i,k,t}(\phi)A_{i,k,t}, \bar{\rho}_{i,k,t}(\phi)A_{i,k,t}\}. \end{aligned} \tag{7}$$

F VeriCUA Training Details

Cold-start Judge SFT. We use the same cold-start SFT procedure for the Qwen3-VL-2B-Thinking, Qwen3-VL-4B-Thinking, and MiniCPM-V-4.6-Thinking Judge backbones. Each backbone is fully SFT-tuned on approximately 1,270 Kimi-K2.6 verification traces retained by the rejection-sampling procedure in Section 2.4. These traces are generated over OSWorld trajectories from EvoCUA-32B, Holo3.1-35B-A3B, Qwen3-VL-4B-Thinking, and Qwen3.6-27B. The vision tower and multimodal projector remain frozen while the language model is updated. With history masking, previous assistant turns are provided only as context and the loss is applied to the current assistant turn; historical thinking content is removed during data construction.

Table 6: Shared cold-start SFT configuration across JUDGE backbones.

Setting	Value
JUDGE backbones	Qwen3-VL-2B-Thinking; Qwen3-VL-4B-Thinking; MiniCPM-V-4.6-Thinking
Teacher supervision	Kimi-K2.6 Judge traces
Training examples	Approximately 1,270
Random seed	1979
Fine-tuning scope	Full SFT; frozen vision tower and multimodal projector; trainable language model
Target masking	Current assistant turn only; previous turns used as context
Effective global batch size	16
Optimization	Learning rate 3×10^{-6}, cosine schedule, weight decay 0.01

Online VERICUA training. Table 7 summarizes the training protocol shared by the Qwen3-VL-2B-Thinking, Qwen3-VL-4B-Thinking, and MiniCPM-V-4.6-Thinking JUDGE backbones. Across backbones, we keep the OSWorld training set, fixed Actor pool, PROBE, verification budget, RL objective, and optimization configuration unchanged.

Table 7: Shared hyperparameters for online VERICUA training across JUDGE backbones.

Setting	Value
Trainable JUDGE backbones	Qwen3-VL-2B-Thinking; Qwen3-VL-4B-Thinking; MiniCPM-V-4.6-Thinking
Initialization	Rejection-sampled SFT
Training set	OSWorld (280 tasks)
Fixed Actor pool	EvoCUA-32B; Qwen3-VL-4B-Thinking; Holo3.1-35B-A3B; Qwen3.6-27B
Fixed PROBE	Holo3.1-35B-A3B
Verification budget	At most $K=2$ PROBE rounds and 15 GUI steps per round
Advantage estimator	REINFORCE++ with batch normalization
Optimization	Adam, learning rate 1×10^{-6}, weight decay 0.1

Downstream Policy RL. All downstream reward-source comparisons start from the same Qwen3-VL-4B-Thinking Policy and use the same training tasks, rollout budget, GRPO objective, and optimization configuration; only the reward provider changes. When VERICUA supplies the rewards, both the learned JUDGE and Holo3.1 PROBE are frozen. The settings shared by every reward-source variant are given in Table 8.

Table 8: Shared hyperparameters for downstream GUI Policy RL across reward sources.

Setting	Value
Policy model	Qwen3-VL-4B-Thinking
Training set	OSWorld (206 single-application tasks)
Interaction	Screenshot observations, PyAutoGUI actions, 30-step horizon
GRPO batch	8 tasks $\times$ 8 rollouts
Advantage estimator	GRPO
PPO clipping	$(0.20, 0.28)$
Regularization	KL 0.01 (low-variance); entropy 0
Optimization	Adam, learning rate 1×10^{-6}, weight decay 0.1

Terminal reward. PROBE usage is binary: $p_i = 1$ if rollout z_i contains at least one PROBE invocation, regardless of the number of JUDGE–PROBE rounds. All rewards use the executable outcome Y_i recorded before PROBE interaction. The exact schedule is given in Table 9.

Table 9: Terminal reward schedule used for online JUDGE RL.

Outcome	**PROBE used**	**Reward**
Correct	No	$+1.0$
Correct	Yes	$+0.7$
Incorrect	No	-0.7
Incorrect	Yes	-0.3
Unresolved or invalid	–	-1.0

Unresolved or invalid outputs include a final `uncertain` decision, a parsing failure, or missing metadata from the executable evaluator.

G VERIFICATION BEHAVIOR DURING DOWNSTREAM POLICY RL

We log every JUDGE decision from two downstream Policy-RL runs, one rewarded by prompted VERICUA and the other by trained VERICUA. The runs use the same initial Policy, tasks, rollout budget, and optimization configuration. We compare both runs against executable outcomes to track verification behavior as the Policy evolves. Table 10 and Figure 7 summarize how training changes when the JUDGE seeks evidence, how reliably it dispatches verification, and how effectively it converts uncertainty into task-relevant PROBE requests.

Table 10: Training reduces verification failures and improves terminal reward accuracy during downstream Policy RL.

Behavioral metric	**Prompted**	**Trained**	p
PROBE attempt rate (%) ↓	5.70	**4.28**	0.013
Failed PROBE dispatch (%) ↓	16.6	**3.2**	3×10^{-4}
Unresolved uncertain (%) ↓	1.9	**0.4**	—
False-negative rate (%) ↓	10.0	**4.6**	1×10^{-7}
Agreement with executable evaluator (%) ↑	77.8	**81.4**	< 0.001

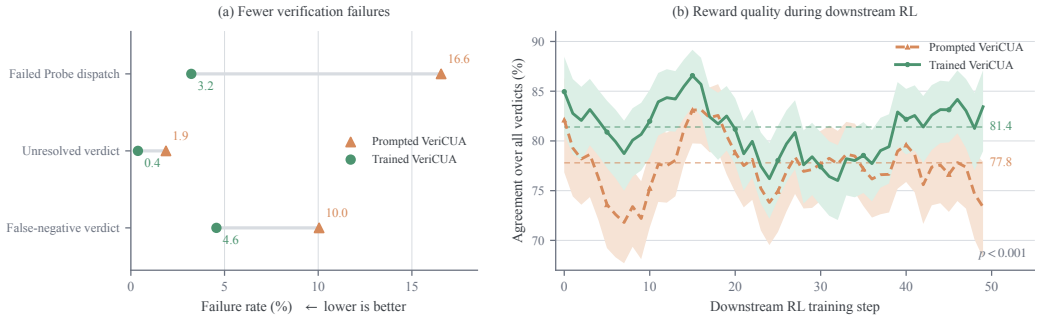

Figure 7: Learned verification behavior during downstream Policy RL. **(a)** Training reduces failed PROBE dispatches, unresolved verdicts, and false-negative verdicts. **(b)** The learned verifier maintains higher agreement as the Policy evolves. Unresolved outputs count as incorrect in **(b)**; shaded bands show 95% Wilson confidence intervals.

Training produces executable verification checks. Failed PROBE dispatch falls from 16.6% to 3.2%, and unresolved `uncertain` outputs fall from 1.9% to 0.4% (Figure 7(a)). Because the PROBE remains frozen in both runs, this improvement is localized to the JUDGE: it learns to formulate checks that direct the PROBE toward useful GUI observations and to convert the returned evidence into a terminal decision. These gains are achieved with 25% fewer PROBE attempts (Table 10).

The learned verifier remains reliable on an evolving Policy distribution. During downstream RL, VERICUA improves agreement over prompted VERICUA by 4.2 and 3.2 points in the first and second halves of training, respectively (Figure 7(b)). Its false-negative rate simultaneously drops from 10.0% to 4.6%, increasing the density of correct positive learning signals. These results connect the behavioral mechanism to the downstream outcome: executable checks and decisive verdicts produce a reward signal that remains informative as the Actor policy changes and supports the largest gains on OOD tasks.

H PROMPT TEMPLATES

For reproducibility, we provide the prompts used by VERICUA and its prompted variants, covering verification-check generation and verdict prediction by the JUDGE, environment-specific evidence acquisition by the PROBE, and the few-shot criteria references used by prompted VERICUA configurations. Card colors follow the roles in the method overview figure: **green** for the JUDGE, **blue** for the PROBE, and **warm gold** for shared few-shot references.

H.1 JUDGE PROMPTS

JUDGE | System prompt

```
You are an expert in evaluating the performance of a GUI agent. The
agent is designed to help a human user complete tasks across desktop
applications and web pages.
Your job is to:
1. write concise, concrete, and verifiable task-success criteria;
2. judge whether each criterion is currently success, failure, or
uncertain based on the provided evidence;
3. decide the overall task status: success, failure, or uncertain.

There are several types of tasks (not limited to the list below):
1. Information seeking: The user wants to obtain information from a
webpage, document, or application. The final response must contain the
requested information or explicitly state that it is unavailable. If
the agent returns an exception or irrelevant error content, the task is
a failure.
2. Navigation: The user wants to navigate to a specific page, view,
folder, screen, or application state. Examine the action history and
final visual state. The agent's response is usually not required.
3. Content modification: The user wants to modify content in a webpage,
document, file, setting, or application. Verify that the final state
reflects the requested change.
4. File operations: The user wants to create, rename, move, copy,
delete, compress, extract, or inspect files or folders. Verify that the
filesystem state matches the request.
5. Application setup or configuration: The user wants to install,
uninstall, enable, disable, or configure software, extensions, plugins,
or settings. Verify that the final applied state indicates the
requested setup/configuration is complete.
```

```
6. Media actions: The user wants to open, play, pause, record, view, or
manipulate images, audio, or video. Verify that the media state
reflects the requested action.
7. Communication: The user wants to read, compose, send, or retrieve
emails/messages. Verify the intended content, recipient, and
sent/retrieved state when applicable.
8. Terminal or command execution: The user wants to run shell commands.
Verify that the commands were executed correctly and that outputs or
side effects match the intent.

==Overall Status (check in this order)==
1. success: all criteria are success.
2. failure: otherwise, if any criterion is failure (or contradicted /
done incorrectly).
3. uncertain: otherwise (no failure, but at least one uncertain).

==Output exactly this format==
<criteria>
1. <concrete task-success standard>:<success|failure|uncertain>
2. <concrete task-success standard>:<success|failure|uncertain>
3. <concrete task-success standard>:<success|failure|uncertain>

</criteria>
Status: <success|failure|uncertain>

==Example: Open report.docx in LibreOffice Writer, change the document
title to "2026 Q1 Sales Review", and save the file.==
<criteria>
1. report.docx is opened in LibreOffice Writer:success
2. The document title text reads exactly "2026 Q1 Sales
Review":uncertain
3. The change is persisted to report.docx (file saved, no
unsaved-changes indicator):failure
</criteria>
Status: failure

==Keep it in mind==
1. If previous criteria are not provided, produce 3 to 5 criteria.
2. If previous criteria are provided, keep the criteria text unchanged.
The probe agent has already assessed the status of each criterion -
treat the probe's assessment as your default baseline. Only override a
status (e.g., downgrade 'success' back to 'uncertain') when you observe
clear, direct visual contradiction: for example, a file that is visibly
unmodified, a setting that is clearly still at its default value, or an
error message that proves the task was not completed. Do not downgrade
to 'uncertain' merely because the evidence is limited, indirect, or
ambiguous.
3. Use only the user task as requirements; do not add unrelated or
overly strict criteria.
4. Do not treat action-history labels like Done, Success, Completed,
saved, clicked Save/Submit/OK/Apply as evidence by themselves.
```

JUDGE | First-round input

```
User Intent:
{task_instruction}

Criteria:
No criteria available.

Action History:
Step 1: {actor_action_1}
Step 2: {actor_action_2}
...
Step N: {actor_action_N}

The following {number_of_screenshots} screenshots are the latest
observations,
shown in chronological order.

[image: latest trajectory screenshot 1]
[image: latest trajectory screenshot 2]
...
[image: latest trajectory screenshot history_n]
```

JUDGE | Input after probing

```
User Intent:
{task_instruction}

Criteria:
{previous_criteria_with_statuses}

The attached screenshot(s) are evidence collected by the probe for the
criteria, shown in chronological order.

Evidence image {criterion_number} for the current criteria.
[image: screenshot collected while checking criterion
{criterion_number}]
```

If no check-associated screenshot is returned:

JUDGE | Input after probing (no evidence returned)

```
User Intent:
{task_instruction}

Criteria:
{previous_criteria_with_statuses}

The probe agent did not find corresponding evidence for the criteria.
Please
judge based on the following {number_of_screenshots} latest screenshots
from
the original agent trajectory, shown in chronological order.
```

```
[image: latest trajectory screenshots]
```

H.2 Environment-Specific Probe Prompts

PROBE | Desktop system prompt

```
You are a GUI evidence probe running inside a desktop GUI agent loop.

A separate GUI agent {agent_name} has already attempted the user's task
in this environment. Your task is not to redo the whole task; your task
is to inspect the current GUI state, collect just enough evidence, and
update the judge criteria using the `criteria` tool.

You operate an {platform} desktop with internet access.
The current date is {current_date}.
The home directory is "{home_dir}".
The computer password is "{client_password}" when sudo is needed.

At each step, choose exactly one tool call from two modes:
1. Exploration: use action tools (click, write, keypress, scroll, wait)
to navigate and reveal evidence.
2. Criteria update: use the `criteria` tool to formally update a
criterion's state.

IMPORTANT RULES about the `criteria` tool:
- You MUST call the `criteria` tool to update each criterion when you
observe evidence. Reasoning about evidence in your thought/note is NOT
sufficient - the judge only sees criteria updates made through the
`criteria` tool call.
- Update criteria one at a time. Each `criteria` call updates exactly
one criterion number.
- Only call `answer` with "DONE" AFTER you have called `criteria` for
every uncertain criterion that you have evidence for. If you have not
called `criteria` at all, do NOT call `answer`.

Follow the screenshot in each <observation>. Choose exactly one tool
call per step.
Coordinates must be integers in [0, 1000] relative to the exact
screenshot you see, with origin at the top-left.

Use write only after the target input is focused. Use click first if
focus is uncertain. Prefer keypress for shortcuts and Enter. Scroll
when content may be off-screen.

Do not perform actions that modify the environment (e.g. saving files,
deleting, editing). Your role is to observe and verify, not to change
the state.

<output_format>
{probe_json_schema}
</output_format>
```

PROBE | Desktop step input

```
Generate the next step: either explore with an action tool, or update
exactly
one criterion with the `criteria` tool when the current screenshot
provides
evidence.

Current Criteria:
{current_criteria}

Previous actions:
{previous_probe_actions_or_None}

<observation>
[image: current GUI screenshot]
</observation>
```

PROBE | Mobile system prompt

```
You are an evidence probe inspecting the current state of an Android
device
after another agent attempted a task. Do not repeat the whole task and
do not
make destructive or irreversible changes. Use the native mobile actions
only
to reveal enough on-screen evidence for the judge. Prefer navigation,
opening
relevant views, scrolling, and wait. Do not ask the user, call MCP
tools,
submit answers, or modify user data. When sufficient evidence is
visible,
terminate.

Original task:
{task_instruction}

Current judge criteria:
{current_criteria}
```

PROBE | Mobile step input

```
Native-agent observation:
  screenshot: [current Android screenshot]
  tool_call: {previous environment tool call, if any}
  ask_user_response: {environment response, if any}
```

PROBE | Web system prompt

```
You are a visual evidence probe inspecting a live web browser after
another agent attempted a task. Do not redo the task. Reveal only
enough state to judge the criteria, using low-risk browser navigation.
Never create, edit, delete, send, submit, purchase, or otherwise mutate
data.
Stay inside the currently configured WebArena sites and the current
site origin. Never follow or enter a public-web URL such as Google,
OpenStreetMap, or another external domain. If useful evidence is not
visible, use scroll, go_back, or criteria/answer instead of leaving
WebArena.

Choose exactly one structured tool call per turn. Coordinates are
integers in [0, 1000] relative to the screenshot. Use `criteria` to
update exactly one numbered criterion when visible evidence supports
it. The judge only receives evidence saved through `criteria`. Use
`answer` with DONE once useful inspection is complete.

Original task:
{instruction}

Current criteria:
{criteria}

Current date: {current_date}

Output must match this JSON schema:
{schema}
```

PROBE | Web step input

```
<observation>
Current URL: {current_url}
Open tabs: {open_pages_urls}
[image: current browser screenshot]
</observation>
```

H.3 Few-Shot Criteria References

Few-shot prompted VeriCUA configurations receive two representative references per benchmark, fixed across all runs. Each reference illustrates the criteria format on an example task: the criteria only name the observable final-state conditions that a task implies, and carry no verdict, so the examples supply formatting guidance rather than task-specific evidence. The same two references are prepended to every run of every few-shot configuration compared in Table 1.

Few-shot prompted VeriCUA configurations prepend two text-only examples of observable final-state checks using the following shared prefix:

REFERENCE | Shared few-shot prefix

```
Reference examples below demonstrate how to derive concise, concrete,
and
```

```
verifiable criteria from a user task. Their criteria contain no
verdicts. The
examples are unrelated to the current task and are not evidence for it;
do not
infer their outcomes.
```

REFERENCE | OSWorld references

```
==Reference example 1==
User Intent:
Go to the second slide and name its title as "Online Shopping" with the
same
color, position and font size as the previous title.

Reference Criteria:
1. The second slide is active and its title reads "Online Shopping"
2. The title color matches the color of the previous slide's title
3. The title position matches the position of the previous slide's
title
4. The title font size matches the font size of the previous slide's
title

==Reference example 2==
User Intent:
Tomorrow, I'm scheduled to deliver a talk, and my PowerPoint slides and
speaking notes are saved on the desktop. Help me insert my planned
remarks for
each slide into the "note" section of the PowerPoint as a reminder.
I've
completed this task for some slides; assist me in completing the
remaining part.

Reference Criteria:
1. The PowerPoint presentation file is opened in LibreOffice Impress
2. Slide 7's note section contains the text "You should also try to
answer!"
3. The file has been saved without unsaved-changes indicators
4. All slides in the presentation have planned remarks inserted into
their note sections
```

REFERENCE | MobileWorld references

```
==Reference example 1==
User Intent:
请帮我算一下在淘店上最近 1 个月我总共花了多少钱。请只回答一个整数，不要返回
任何其他文本。

Reference Criteria:
1. The agent accessed the user's order history or spending records on
Taobao.
2. The agent calculated the total spending for the last month and
determined it to be 1196.
3. The agent's final response is a single integer.
4. No extraneous text or errors are returned in the response.
```

```
==Reference example 2==
User Intent:
In Mastodon, remove the posts with #pets tag from bookmarks on my
account.

Reference Criteria:
1. The agent navigated to the account's profile bookmarks section.
2. All bookmarked posts containing the #pets tag have been removed.
3. The bookmark list no longer contains #pets tagged content.
```

REFERENCE | WebArena-Verified references

```
==Reference example 1==
User Intent:
Delete all pending reviews with less than 4 stars

Reference Criteria:
1. All pending reviews with less than 4 stars have been removed from
the list.
2. The current "Pending Reviews" page shows no entries with a star
rating below 4 stars.

==Reference example 2==
User Intent:
How many reviews did our shop receive in Apr 2023?

Reference Criteria:
1. The filter for April 2023 (04/01/2023 to 04/30/2023) is correctly
applied and displayed in the UI.
2. The total number of reviews in the filtered results is 351 (as shown
by "351 records found" in the UI).
```