GUI-PRA: Process Reward Agents for GUI Tasks

Tao Xiong[1,†,*], **Xueyu Hu**[1,†], **Yurun Chen**[1], **Yuhang Liu**[1],
Changqiao Wu[2], **Pengzhi Gao**[2], **Wei Liu**[2], **Jian Luan**[2], **Shengyu Zhang**[1,‡]

[1]Zhejiang University, [2]MiLM Plus, Xiaomi Inc.
{xiongtao, sy_zhang}@zju.edu.cn

Abstract

Long-horizon GUI automation remains challenging due to error accumulation over extended interaction sequences. Process Reward Models (PRMs) provide dense step-level supervision for mitigating error accumulation, yet standard PRMs are poorly suited to GUI verification. Standard PRM judgments often rely on superficial visual alignment rather than functional correctness, reflecting an *evaluative knowledge gap* caused by missing domain-specific adjudication logic. Standard PRMs also perform passive, single-pass visual assessment, which creates *Visual Ambiguity* when reliable judgment requires actively locating, parsing, or inspecting task-relevant UI evidence. We introduce **GUI-PRA**, a Process Reward Agent that transforms GUI process evaluation from passive scoring into active investigation. GUI-PRA couples **Experience-Injected Criterion Synthesis**, which distills generalized verification principles into state-specific criteria, with **Criterion-Guided Autoregressive Perception**, which uses these criteria to navigate multi-granularity visual tools and gather grounded evidence. On AndroidWorld and Mobile-MiniWoB++, GUI-PRA achieves average gains of **5.0** and **6.5** SR points over standard PRMs across the Qwen-VL backbones, with Qwen3-VL attaining **54.74%** success rate on AndroidWorld. On the offline OS-Critic Bench, GUI-PRA demonstrates strong competitiveness against fully trained critic models. We will release our codes at https://github.com/YuanDaoze/GUI-PRA/.

1 Introduction

Graphical User Interface (GUI) Agents (Hu et al., 2025; Li et al., 2024c; Cao et al., 2026; Ding et al., 2026; Wang et al., 2026b), powered by Multimodal Large Language Models (MLLMs) (Zhang et al., 2025a; Li et al., 2024a; Zheng et al., 2024), have shown promise for automating complex user tasks. Yet these agents frequently fail in long-horizon scenarios (Tian et al., 2025; Wang et al., 2025b; Wu et al., 2025a), where small errors compound over extended interaction sequences. GUI tasks particularly benefit from process supervision because feedback is sparse and a single incorrect action can invalidate the rest of a trajectory. To mitigate this, Process Reward Models (PRMs) (Gandhi et al., 2025; Wanyan et al., 2025; Xiao et al., 2025; Chen et al., 2025a) have gained traction as a mechanism for providing dense, step-level supervision that guides agents toward correct trajectories and

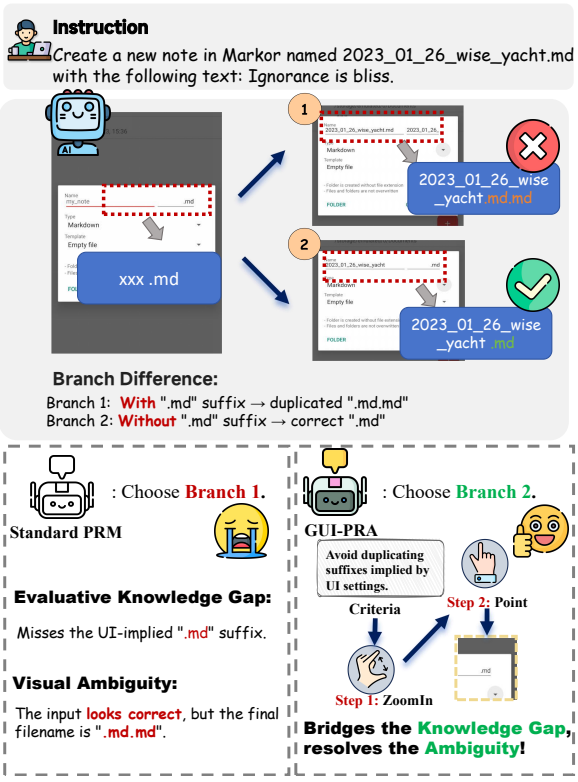


Figure 1: A motivating example of a "Double Suffix" error case, comparing Standard PRM and GUI-PRA. In the figure, "ZoomIn" is shorthand for `ZoomInSubfigure`.

[†]Equal contribution, [‡]Corresponding Author, [*]Work done during an internship at Xiaomi Inc.

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 29695–29716
October 24-29, 2026 ©2026 Association for Computational Linguistics

prunes erroneous paths during inference.

However, directly applying PRMs in GUI domains yields suboptimal supervision due to two fundamental limitations. As illustrated in Figure 1, standard PRMs suffer from an *evaluative knowledge gap* (Zhang et al., 2025b; Dai et al., 2026; Krumdick et al., 2025; Chiu et al., 2025). Unlike domain experts who rely on deep functional knowledge to verify correctness, standard PRMs judge actions by superficial visual alignment. They lack the logic to verify procedural dependencies (e.g., confirming that preconditions are met before an action) or enforce state-transition rules (e.g., checking data persistence). This leads to failures such as the *completion illusion*, where a PRM assigns high reward to an agent that has visually filled a form but failed to execute the functional "Save" action, or misses subtle violations of negative constraints.

Second, standard PRMs are constrained by *Visual Ambiguity* (Fan et al., 2024; Xu et al., 2024; Hu et al., 2024). Standard PRM evaluation is typically passive and single-pass, leaving the judge unable to actively locate, parse, or inspect the UI evidence required for reliable assessment. Fine-grained cues such as checkbox states, dense text fields, or visually similar list items may therefore remain unresolved during scoring. Without an evidence-seeking mechanism, PRMs tend to render verdicts from coarse visual proxies rather than grounded observations, leading to hallucinatory conclusions.

We introduce **GUI-PRA**, a Process Reward Agent that addresses both limitations through a single organizing principle: *decoupling what to verify from how to verify it*. As shown in Figure 2, GUI-PRA transforms a standard PRM from a static scorer into an active investigator. To bridge the evaluative knowledge gap, we propose **Experience-Injected Criterion Synthesis**. Instead of relying on implicit intuition, this mechanism synthesizes explicit adjudication criteria for the current UI state by referencing a repository of verification principles distilled from generalized expert logic. These criteria establish a clear, rule-based standard for what constitutes success at each step. To overcome Visual Ambiguity, we propose **Criterion-Guided Autoregressive Perception**. Guided by the synthesized criteria, this mechanism transforms verification from passive viewing into an active evidence-seeking process. It uses multi-granularity visual tools, including global layout parsing and local inspection, to gather grounded evidence before rendering a verdict.

- We propose **GUI-PRA**, a Process Reward Agent which shifts the supervision paradigm from static, global-view scoring to an active, evidence-seeking investigation process.
- We introduce two coupled mechanisms: **Experience-Injected Criterion Synthesis** for deriving state-specific adjudication criteria, and **Criterion-Guided Autoregressive Perception** for gathering grounded visual evidence through tool use.
- We conduct extensive experiments on AndroidWorld, Mobile-MiniWoB++, and OS-Critic Bench. GUI-PRA improves the Qwen-VL series by **5.0** and **6.5** SR points on average across the two online benchmarks, and remains competitive with trained critic models in offline evaluation.

2 Related Work

(M)LLM-based GUI Agents. Multimodal large language models (Bai et al., 2025; Singh et al., 2026; Team et al., 2025) have substantially expanded the capability of GUI agents. Recent work strengthens GUI agents by post-training MLLMs with supervised fine-tuning (Liu et al., 2025; Hong et al., 2024; Wu et al., 2024) or reinforcement learning (Liu et al., 2025; Xu et al., 2025, 2026; Wang et al., 2026a, 2025a) on curated interaction trajectories. A complementary line designs agent frameworks with specialized modules for planning, grounding, memory, and verification (Ye et al., 2025; Zhang et al., 2023; Xu et al., 2026), yet long-horizon GUI automation still suffers from step-level errors that can cascade into unsafe or irreversible outcomes (Chen et al., 2025b,c). This motivates process-level supervision, where an external verifier checks intermediate candidate actions to avoid unreliable trajectories.

Process Reward Models for GUI Agents. Process-level supervision for GUI agents has emerged from three related sources. One line directly trains dedicated PRMs on GUI interaction data, using (Chen et al., 2025a; Wanyan et al., 2025; Xiao et al., 2025; Wu et al., 2025b) to predict candidate-step validity. These methods require substantial domain-specific data and training resources, which can limit transferability. A second line repurposes general-purpose multimodal models as judges, either in an LLM-as-a-Judge setting

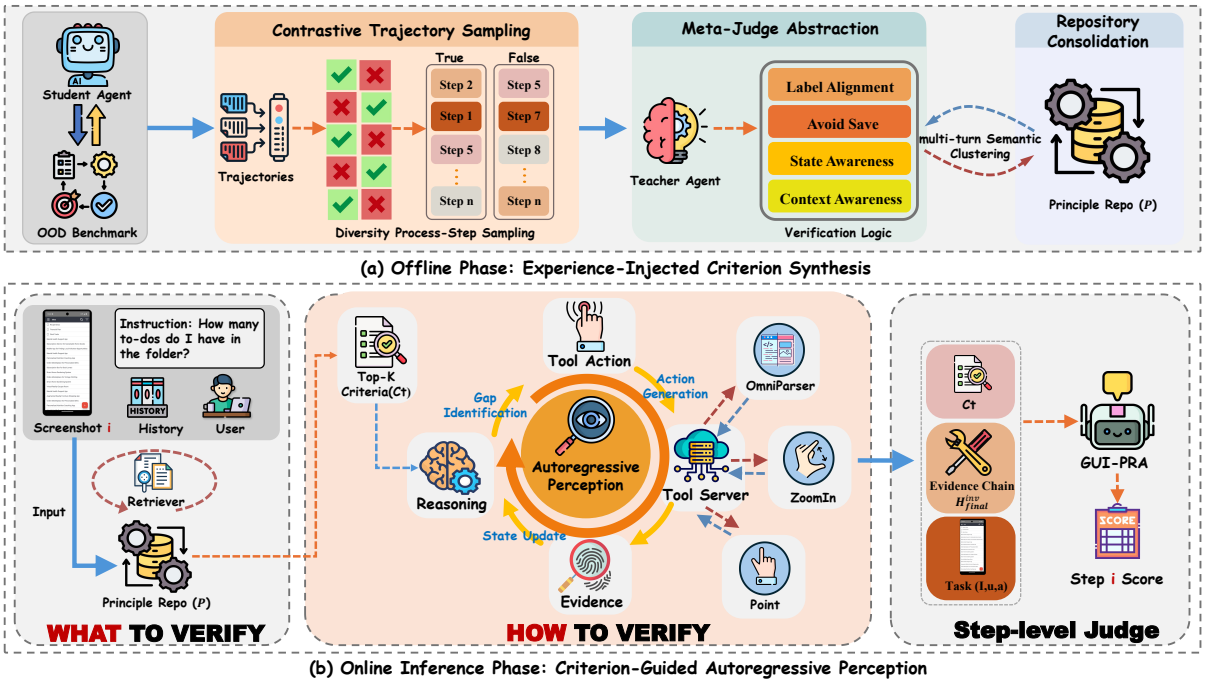


Figure 2: The overall framework of GUI-PRA. **Offline Phase:** We construct a Principle Repository ($\mathcal{P}$) by distilling generalized verification logic from contrastive trajectories using a Meta-Judge. **Online Phase:** The model synthesizes explicit criteria (C_t) based on $\mathcal{P}$, then enters a **Criterion-Guided Autoregressive Loop** to actively gather grounded visual evidence via tools (e.g., `ZoomInSubfigure`, `Point`) for the final evidence-based supervision.

over screenshots and trajectories or in an Agent-as-a-Judge setting over proposed thought-action pairs (Zheng et al., 2026; Dai et al., 2026; Cui et al., 2026). Proactive and agentic approaches can gather additional evidence, but rely largely on free-form reasoning, leaving criteria implicit and evidence acquisition difficult to control. A third line uses PRM-style signals as stepwise feedback inside reinforcement learning, where principle-constrained, curriculum-staged, or retrospective rewards guide policy optimization rather than inference-time verification. Representative examples include Nong et al. (2026); Lu et al. (2025); Tang et al. (2025); Feng et al. (2025). These methods primarily optimize policy parameters rather than provide inference-time supervision for a frozen actor. We propose GUI-PRA , an evidence-grounded Process Reward Agent that guides GUI action selection at inference time using explicit criteria and active visual verification.

3 Preliminary

3.1 GUI Task Automation

We formulate GUI task automation as a sequential decision-making problem. Given a natural language instruction I (formerly denoted as goal g) and an initial GUI state S_0, which comprises a screenshot scr_0 and structural elements e_0 (i.e., $S_0 = (scr_0, e_0)$), the agent must generate a sequence of actions $\hat{\alpha}$ to fulfill the instruction. An action sequence $\hat{\alpha} = (a_1, a_2, \ldots, a_T)$ is valid if the terminal state satisfies a success predicate V, such that $V(\mathcal{T}(S_0, \hat{\alpha}), I) = \texttt{pass}$, where $\mathcal{T}$ denotes the environment transition function $S_{t+1} = \mathcal{T}(S_t, a_t)$.

The agent operates via a ReAct-style loop (Yao et al., 2023). At step t, the interaction history is denoted as:

$$\mathcal{H}_t = \big(u_1, a_1, o_1,\ u_2, a_2, o_2,\ \ldots,\ u_t, a_t, o_t\big),$$

where u_i represents the agent's reasoning thought, a_i is the executable action, and o_i is the observation consisting of the new state S_i. The policy π_θ generates the next thought-action pair $(u_{t+1}, a_{t+1}) \sim \pi_\theta(\cdot \mid \mathcal{H}_t, I)$.

3.2 Supervision with a Standard PRM

To mitigate error accumulation in long-horizon tasks, a Process Reward Model (PRM) provides dense supervision. At step t, the policy generates a set of k candidate thought-action pairs, denoted as $\mathcal{A}_t = \{(u_{t,j}, a_{t,j})\}_{j=1}^{k}$. The standard PRM acts as a discriminator to select the optimal transition.

Formally, it estimates the probability of future success conditioned on the static history $\mathcal{H}_{t-1}$ and the current candidates:

$$(u_t, a_t) = \underset{(u,a)\in\mathcal{A}_t}{\text{argmax}} \ \text{PRM}(I, S_{t-1}, \mathcal{H}_{t-1}, u, a) \tag{1}$$

4 Methodology

4.1 Overview

As illustrated in Figure 2, GUI-PRA redefines the Process Reward Model from a passive scorer to an active investigator. To address two key limitations of standard PRMs, GUI-PRA adopts a three-phase pipeline: (1) **Experience-Injected Criterion Synthesis** (§4.2), which derives *what* to verify from distilled verification principles; (2) **Criterion-Guided Autoregressive Perception** (§4.3), which uses active tools to determine *how* to verify; and (3) **Evidence-Based Supervision** (§4.4), computing the final reward r_t based on grounded evidence rather than static intuition.

4.2 Experience-Injected Criterion Synthesis

Experience-Injected Criterion Synthesis addresses the lack of domain-specific adjudication logic in standard PRMs through the following two-stage process.

Phase I: Offline Verification Principle Distillation. To enable the model to judge like an expert, we first construct a **Synthesized Verification Principle Repository** ($\mathcal{P}$) containing generalized adjudication heuristics. Instead of relying on manual engineering or raw trajectory retrieval, we automate the construction process through a *Generate-Contrast-Abstract* pipeline:

- **Contrastive Trajectory Sampling:** Starting from trajectories produced by a GUI agent on an out-of-distribution benchmark, we construct balanced sets of successful and failed episodes. From these episodes, we further sample diverse intermediate steps to cover different process-level supervision scenarios. This procedure emphasizes **hard negatives**, where the failed action is plausible but logically flawed rather than random noise.

- **Meta-Judge Abstraction:** A strong LLM serves as the Meta-Judge for abstracting verification logic from each contrastive set. Rather than describing only the surface error in a specific interface, the Meta-Judge extracts reusable principles that explain why a transition should be accepted or rejected. For instance, a failed submission with missing required fields can be abstracted into a *State Dependency Validation* principle, where submission actions are valid only after all mandatory fields have been completed.

- **Repository Consolidation:** The extracted logic is consolidated through semantic clustering to keep the repository compact and generalizable. Redundant entries are merged, and overly specific descriptions are rewritten into reusable verification principles. The resulting repository $\mathcal{P}$ covers core GUI verification logic, including state dependencies, functional effectiveness, task-flow compliance, and system feedback validity.

Phase II: Online Principle Instantiation. At inference time, the PRM converts the principle repository into state-specific criteria for the current decision. Given the task instruction I, UI state S_t, and a candidate action (u, a), a dense retriever selects the top-k principles from $\mathcal{P}$ that are most relevant to the current intent. The model then instantiates these principles into concrete verification criteria C_t, specifying what evidence should be checked in the current UI. For example, a retrieved principle on *State Dependency Validation* becomes a requirement to verify that mandatory fields such as Name and Phone are filled before accepting a `Submit` action at (x, y). The resulting criteria C_t are passed to Criterion-Guided Autoregressive Perception, where they guide tool selection and evidence gathering.

4.3 Criterion-Guided Autoregressive Perception

Criterion-Guided Autoregressive Perception turns the criteria C_t into a sequential evidence-gathering process. Guided by the instantiated criteria, the model selects visual tools autoregressively and updates the evidence history until the information needed for judgment is sufficient.

Multi-Granularity Visual Toolset. To enable multi-scale perception, we define a specialized toolset $\mathcal{T}$ comprising `OmniParser` for global layout, `ZoomInSubfigure` for local inspection, `Point` for spatial grounding, and `Terminate` for process control (see Appendix A for technical specifications).

The Autoregressive Investigation Loop. Within this perception module, verification is carried out

as a dynamic inference trajectory that iteratively refines the evidence available for judgment. At each step k, the model conditions on the instruction I, the criteria C_t, and the evolving observation history H_k^{inv}. The loop follows a *Reason-Act-Update* structure:

- **Gap Identification:** The model first maps each criterion in C_t to concrete **Verification Targets** in the current UI. It then identifies which evidence is still missing for validating these targets. For instance, if a criterion requires checking a toggle state, the model treats the target toggle as unresolved until its precise visual state has been inspected.
- **Action Generation:** To bridge the identified gap, the model autoregressively generates the optimal tool action $A_k \in \mathcal{T}$. The selection is strategic: checking existence triggers `Point`, while resolving visual ambiguity triggers `ZoomInSubfigure` to isolate the target region. This step represents an active decision to acquire the most discriminative information required for the verdict.
- **State Update:** The execution result O_k (e.g., a zoomed local view, point coordinates, or parsed UI structure) is appended to the context, forming the updated history $H_{k+1}^{inv} = H_k^{inv} \cup \{A_k, O_k\}$. This updated state provides a grounded foundation for the next iteration.

This loop continues until the model invokes `Terminate` or the maximum number of iterations is reached, ensuring that the final reward signal is not a hallucinated guess, but a logical conclusion derived from a **traceable chain of grounded evidence**.

4.4 Enhanced Supervision with GUI-PRA

In the final stage, GUI-PRA aggregates the synthesized knowledge and collected visual facts to produce the supervisory signal. Unlike the standard PRM (Eq. 1) which relies on opaque probability estimation, we employ an **Evidence-Aware Adjudication Protocol**.

For each candidate $(u, a) \in \mathcal{A}_t$, the Judge Agent rigorously evaluates the alignment between the collected evidence chain H_{final}^{inv} and the explicit criteria C_t. A high reward is assigned only when the evidence affirmatively supports the verification targets. The final action selection is formalized as:

$$(u_t, a_t) = \underset{(u,a)\in\mathcal{A}_t}{\text{argmax}} \ \mathcal{F}_{\text{judge}}(I, C_t, H_{\text{final}}^{inv}, u, a) \quad (2)$$

where $\mathcal{F}_{\text{judge}}$ denotes the scoring function that quantifies the logical consistency between the criteria C_t and the grounded evidence H_{final}^{inv}.

5 Experiment

5.1 Experimental Setup

Benchmarks. We evaluate on AndroidWorld, Mobile-MiniWoB++, and the mobile subset of OS-Critic Bench, using the standard M3A and Pure Vision execution settings for the online benchmarks. Benchmark-specific details are given in Appendix B.2.

Baselines. We compare GUI-PRA with the **Base Model**, which executes tasks without process supervision, and the **Standard PRM**, which passively scores candidate actions without explicit criteria or active tool use.

Metrics. We report **Success Rate (SR)** and **Difficulty-stratified Success Rate (DSR)**, where Easy, Medium, and Hard DSR values sum to the overall SR.[1]

Implementation Details. Online evaluation uses Best-of-N rejection sampling with temperature $\tau = 1.0$, where the supervisor selects the highest-scoring candidate step. Offline evaluation directly ranks pre-collected trajectories. Further protocol, system, and repository details are provided in Appendix B.2 and Appendix B.3.

5.2 Main Results

Table 1 and Figure 3 summarize the performance for online execution and offline discrimination, respectively. Across all settings, GUI-PRA consistently outperforms both the autonomous Base Model and the Standard PRM baseline.

5.2.1 Online Evaluation

Table 1 reports online execution results on AndroidWorld and Mobile-MiniWoB++. GUI-PRA improves over the Standard PRM across all tested backbones and both benchmarks. The strongest result in the main single-backbone comparison comes from Qwen3-VL-8B-Instruct, where GUI-PRA raises SR to **54.74%** and improves over the Standard PRM by 5.17 SR points. More

[1] Values may not sum exactly because each entry is rounded independently.

Setting	DSR (%)			SR (%)	Δ@1	Δ@2
	easy	medium	hard			
AndroidWorld						
InternVL3-8B-Instruct (M3A)	5.17	0.00	0.86	6.03	–	–
w/ PRM	6.90	0.00	**1.72**	8.62	+2.59	–
w/ **GUI-PRA**	**8.62**	0.00	0.86	**9.48**	+3.45	+0.86
InternVL3.5-8B-Instruct (M3A)	6.04	0.00	0.86	6.90	–	–
w/ PRM	9.91	0.43	0.86	11.20	+4.30	–
w/ **GUI-PRA**	**10.34**	**1.72**	0.86	**12.93**	+6.03	+1.73
Qwen2.5-VL-7B-Instruct	15.52	3.88	0.86	20.26	–	–
w/ PRM	17.24	1.72	0.86	19.83	-0.43	–
w/ **GUI-PRA**	**21.55**	**2.16**	0.86	**24.57**	+4.31	+4.74
Qwen3-VL-8B-Instruct	–	–	–	47.60	–	–
w/ PRM	30.60	14.66	**4.31**	49.57	+1.97	–
w/ **GUI-PRA**	**36.21**	**14.66**	3.88	**54.74**	+7.14	+5.17
Mobile-MiniWoB++						
Qwen2.5-VL-7B-Instruct	–	–	–	51.09	–	–
w/ PRM	–	–	–	45.65	-5.44	–
w/ **GUI-PRA**	–	–	–	**53.26**	+2.17	+7.61
Qwen3-VL-8B-Instruct	–	–	–	59.78	–	–
w/ PRM	–	–	–	59.78	0.00	–
w/ **GUI-PRA**	–	–	–	**65.22**	+5.44	+5.44

Table 1: Online execution performance on AndroidWorld and Mobile-MiniWoB++. Each backbone is used as both the execution agent and the reward-model backbone. **SR** denotes the overall Success Rate, and **DSR** reports AndroidWorld Success Rate stratified by task difficulty. **Δ@1** is the SR-point gain over the base model, and **Δ@2** is the SR-point gain of GUI-PRA over the Standard PRM.

importantly, the improvement is not limited to already stable settings. On Qwen2.5-VL, the Standard PRM slightly degrades AndroidWorld performance, while GUI-PRA recovers the loss and improves over the PRM baseline. Mobile-MiniWoB++ shows the same recovery pattern under a denser UI setting. These results indicate that GUI process supervision benefits from criterion-guided active investigation rather than a passive VLM critic alone.

The Difficulty-Stratified Success Rate further clarifies the source of the gains. Improvements are concentrated in the *Easy* setting and selected *Medium* cases, while *Hard* tasks remain largely unchanged. For Qwen3-VL-8B-Instruct on AndroidWorld, Easy-task DSR rises from 30.60% to **36.21%**, whereas Hard-task DSR stays nearly flat. This pattern matches the verifier role of GUI-PRA. Active investigation is most effective when the actor has already generated plausible candidate actions; when the correct action is absent from the candidate set, a verifier has limited room to recover. A task-level actor-coverage analysis is provided in Appendix C.1.

5.2.2 Offline Evaluation

To assess the intrinsic supervision quality of our method, we conduct a static evaluation on the Mobile Subset of the **OS-Critic Bench**. Figure 3 compares GUI-PRA with standard zero-shot PRMs and uses **GUI-Critic-R1** (Wanyan et al., 2025) as a supervised critic reference for the Qwen2.5 series.

The offline results show that active investigation improves transition discrimination without task-specific critic training. On the Qwen2.5 backbone, GUI-PRA improves Accuracy from 58.45% to **59.13%** and F1 from 65.79% to **66.16%**. On Qwen3, GUI-PRA also raises Accuracy, from 63.24% to **64.30%**, although F1 decreases after the updated official evaluation. We therefore treat the offline result as evidence of improved accuracy and competitive discrimination, rather than a uniform gain on every metric.

GUI-PRA also remains competitive with the supervised critic reference. On Qwen2.5, it matches GUI-Critic-R1 in Accuracy and approaches its F1 score. This result complements the online evaluation. Online SR measures whether supervision improves execution, whereas OS-Critic Bench iso-

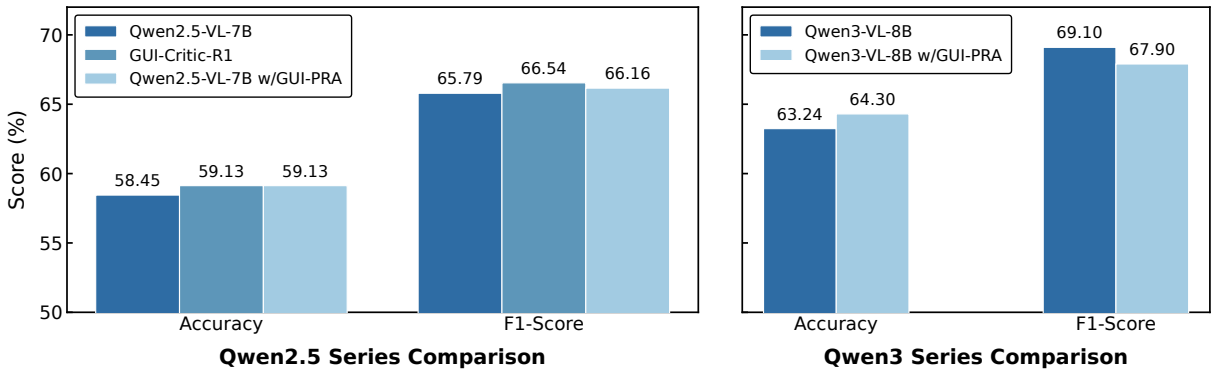


Figure 3: Offline critic performance on the **OS-Critic Bench** (Mobile Subset). We report Accuracy and F1-Score for zero-shot PRMs, GUI-PRA, and the supervised GUI-Critic-R1 reference. All Qwen backbone names denote Instruct variants.

Configuration	Acc (%)	Prec (%)	Rec (%)	F1 (%)
Perceval-3B-policy	55.48	55.96	55.20	55.58
+GUI-PRA	**55.94**	54.73	**73.30**	**62.67**
Δ	+0.46	-1.23	+18.10	+7.09

Table 2: Perceval-3B policy transfer on the OS-Critic Mobile Subset. Values are percentages.

lates the critic's ability to distinguish valid from invalid transitions. The offline results support our central claim that criterion-guided active investigation can provide stronger evidence for step-level GUI verification than passive scoring.

5.2.3 Transfer to a Process-Supervised Policy

We further evaluate whether GUI-PRA transfers to a policy that has already received process supervision. We apply the same frozen-supervisor procedure to the released Perceval-3B-policy checkpoint (Min et al., 2026), a Qwen2.5-VL-3B policy trained with perception-centric process supervision. On the OS-Critic Bench Mobile Subset (Table 2), GUI-PRA improves Accuracy by 0.46 points, Recall by 18.10 points, and F1 by 7.09 points. These results show that GUI-PRA reduces false negatives and helps the model make more reliable transition judgments, with a modest precision trade-off.

5.3 Ablation Study

Table 3 ablates the two core modules of GUI-PRA on AndroidWorld with Qwen3-VL-8B-Instruct. Removing the active perception tools causes the largest drop, reducing SR by 4.74 points and bringing performance close to the Standard PRM. This result shows that retrieved criteria alone are insufficient when the verifier cannot actively inspect task-relevant UI evidence.

Method	DSR (%)			SR (%)	Δ@1
	easy	medium	hard		
PRM	30.60	14.66	4.31	49.57	–
GUI-PRA (Full)	**36.21**	**14.66**	**3.88**	**54.74**	–
w/o component:					
– Principle	36.21	12.93	2.59	51.72	-3.02
– Tool	32.76	12.93	4.31	50.00	-4.74

Table 3: Ablation study on AndroidWorld with Qwen3-VL-8B-Instruct. Δ@1 reports the SR-point change relative to the full GUI-PRA framework.

Setup	Role	SR (%)	Impact (Δ)
Qwen3-VL-32B-Instruct	Agent Only	55.60	–
w/ PRM (8B)	Supervisor	56.03	+0.43
w/ **GUI-PRA** (8B)	Supervisor	**57.76**	**+2.16**
Qwen3-VL-8B-Instruct	Agent Only	47.60	–
w/ Principle & Tool	Integrated Agent	32.76	**-14.84**

Table 4: Efficient supervision and role decoupling on AndroidWorld. **Impact** (Δ) reports the SR-point change relative to the corresponding Agent Only baseline.

Removing the principle repository also degrades performance, reducing SR by 3.02 points. In this setting, the verifier can still inspect the interface, but lacks explicit adjudication logic for deciding what evidence matters. The two modules therefore address complementary failure sources: principles specify what should be verified, while active perception gathers the evidence needed to verify it.

5.4 Efficient Supervision and Role Decoupling

We further investigate two critical questions:

> ***RQ1:*** *Can a smaller* GUI-PRA *supervisor effectively guide a larger, more capable agent?*

Configuration	Acc (%)	F1 (%)	ΔAcc
Qwen3-VL-8B (Zero-shot PRM)	63.24	69.10	–
+ GUI-PRA (GPT-5-mini)	61.64	66.13	-1.60
+ GUI-PRA (GPT-5.1)	**64.30**	67.90	+1.06

Table 5: Principle quality ablation on the OS-Critic Bench (Mobile Subset).

> ***RQ2:*** *Why should* GUI-PRA *operate as a critic rather than as an executor?*

Weak-to-Strong Generalization. For **RQ1**, we use Qwen3-VL-8B-Instruct as the supervisor for a Qwen3-VL-32B-Instruct agent. As shown in Table 4, the standard 8B PRM gives only a marginal gain over the 32B agent. In contrast, the 8B GUI-PRA supervisor improves SR by 2.16 points. This result suggests that active verification can make a smaller model useful as a supervisor, since the supervision task focuses on checking a finite set of candidate actions rather than generating the full trajectory.

Critic vs. Actor Paradigm. For **RQ2**, we test whether the same principle-and-tool mechanism can be integrated directly into the executor. This actor-style deployment performs substantially worse, dropping SR from 47.60% to 32.76%. The result supports keeping GUI-PRA as a critic. As a decoupled supervisor, GUI-PRA can investigate candidate actions and sharpen decision boundaries without mixing verification steps into the actor's generative context.

5.5 Principle Quality Analysis

This experiment evaluates whether the criterion module depends on the quality of distilled principles. We keep the runtime backbone fixed as Qwen3-VL-8B-Instruct and vary only the Meta-Judge used for offline distillation. As shown in Table 5, GPT-5-mini principles reduce both Acc and F1 relative to the zero-shot PRM, while GPT-5.1 principles improve Acc by 1.06 points.

The gap comes from the specificity of the distilled verification logic. GPT-5-mini often produces shallow location rules, such as checking whether a tap falls inside a visible bounding box. These rules repeat surface-level grounding cues and can distract the verifier from functional correctness. GPT-5.1 more often distills task-aware rules. For checkout-related goals, it asks the verifier to check whether the selected control belongs to the cart or checkout flow. For termination decisions, it warns against stopping merely because a header or main title is visible. The F1 drop in Table 5 also shows a limitation: principle injection is not automatically beneficial, and noisy criteria can weaken downstream supervision.

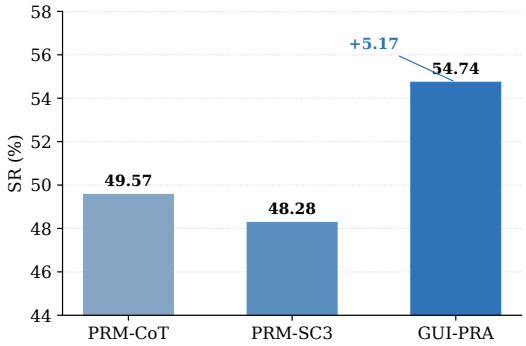


Figure 4: Compute-budget controlled comparison on AndroidWorld (Qwen3-VL-8B-Instruct).

5.6 Compute-Budget Comparison

A natural concern is whether GUI-PRA's gain merely reflects more test-time compute. We control for this by giving the Standard PRM a matched budget. PRM-CoT uses one chain-of-thought pass per candidate. PRM-SC3 is a self-consistency variant that samples each candidate three times at temperature 0.7 and averages the scores. GUI-PRA denotes the full framework.

As shown in Figure 4, self-consistency does not improve the Standard PRM beyond its CoT baseline (48.28 vs. 49.57), and still remains more than 6 SR points below GUI-PRA (54.74). The result suggests that additional reasoning samples cannot substitute for criterion-guided evidence gathering. The bottleneck is not only how many times the verifier reasons, but whether it knows what to verify and can actively collect the required UI evidence.

Latency is bounded and parallelizable. GUI-PRA adds 6.70 s per step on average with ≈1.17 active visual-tool calls per step (1.48 total calls when `Terminate` is included) relative to the single-pass Standard PRM. Since the N candidate evaluations in each Best-of-N step are independent, the wall-clock overhead can be reduced through parallel execution, although the underlying computation remains. Detailed latency, tool-reliability, and recovery analyses are provided in Appendix C.2.

5.7 Additional Analyses

To further assess the scope and practical behavior of GUI-PRA, we conduct additional experiments

on actor coverage, latency, failure modes, tool reliability, and criterion diversity. The results show that GUI-PRA benefits tasks with viable actor candidates, incurs bounded investigation overhead, and can recover from explicit tool failures, while silent semantic misgrounding remains a limitation. Detailed results are provided in Appendices C.1, C.2, D, and E.

5.8 Case Study

Figures 8 and 9 present case studies for *ContactsNewContactDraft* and *NotesTodoItemCount*, exemplifying a "do not save" negative constraint and a counting-completeness requirement respectively. In both cases, a passive verifier can mistake visually plausible progress for task completion or accept premature termination. GUI-PRA instead keeps checking task-specific criteria—field population and the save-prohibition rule for the first task, and item-by-item counting completeness for the second—before adjudicating termination. These cases show how criterion-guided supervision enforces functional correctness beyond surface-level visual completion. Detailed operational steps and intermediate reasoning are provided in Appendix F.

6 Conclusion

We introduced GUI-PRA, a Process Reward Agent that transforms GUI process evaluation from passive scoring into active investigation. By combining Experience-Injected Criterion Synthesis with Criterion-Guided Autoregressive Perception, GUI-PRA derives task-specific criteria and gathers grounded UI evidence before assigning supervision. Experiments across online execution and offline critic evaluation show consistent gains over standard PRMs, supporting active verification as a practical direction for reliable GUI agents.

Limitations

GUI-PRA has shown robust gains across multiple backbones and benchmarks; we note two scope considerations rather than open failure modes. First, like any MLLM-based verifier, the quality of process supervision is upper-bounded by the underlying visual tools and language backbone, although our experiments suggest that the autoregressive investigation loop already absorbs most of the noise from individual tool calls. Second, the Principle Repository is currently distilled offline from contrastive trajectories, which already generalizes well across mobile and web environments; extending it into an online, continually updated repository that adapts to entirely new UI paradigms is a natural direction for future work.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 62402429, U24A20326, and 62441236), the Ningbo Yongjiang Talent Introduction Programme (No. 2023A-397-G), and the Young Elite Scientists Sponsorship Program by CAST (No. 2024QNRC001). The author gratefully acknowledges the support of Zhejiang University Education Foundation Qizhen Scholar Foundation. This work is supported by Xiaomi Inc.'s research program. Conclusions are the authors' and do not represent official funder or government policies.

References

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Wanxia Cao, Chengzhen Duan, Pei Fu, Pengzhi Gao, Niu Lian, Fazhan Liu, Hui Liu, Heng Qu, Qinzhuo Wu, Zhehao Yu, Tongbo Chen, Shiqi Cui, Anan Du, Shukai Jia, Yuanfa Li, Wei Liu, Yike Liu, Wenchao Lu, Zhenbo Luo, and 13 others. 2026. Xiaomi-gui-0 technical report. *Preprint*, arXiv:2606.31410.

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

Yurun Chen, Xavier Hu, Yuhan Liu, Keting Yin, Juncheng Li, Zhuosheng Zhang, and Shengyu Zhang. 2025b. Harmonyguard: Toward safety and utility in web agents via adaptive policy enhancement and dual-objective optimization. *Preprint*, arXiv:2508.04010.

Yurun Chen, Xavier Hu, Keting Yin, Juncheng Li, and Shengyu Zhang. 2025c. Evaluating the robustness of multimodal agents against active environmental injection attacks. *Preprint*, arXiv:2502.13053.

Ming-Chang Chiu, Fuxiao Liu, Karan Sapra, Andrew Tao, Yaser Jacoob, Xuezhe Ma, Zhiding Yu, and Guilin Liu. 2025. Aide: Agentically improve visual language model with domain experts. *Preprint*, arXiv:2502.09051.

Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. 2026. Agentic reward modeling: Verifying gui agent via online proactive interaction. *Preprint*, arXiv:2602.00575.

Gaole Dai, Shiqi Jiang, Ting Cao, Yuqing Yang, Yuanchun Li, Rui Tan, Mo Li, and Lili Qiu. 2026. Prore: A proactive reward system for gui agents via reasoner-actor collaboration. *Preprint*, arXiv:2509.21823.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, YenSung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, and 31 others. 2024. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. *Preprint*, arXiv:2409.17146.

Zihan Ding, Longxu Dou, Qi Gao, Xiangwu Guo, Shengchao Hu, Zilong Huang, Zihang Jiang, Lei Ke, Mengcheng Lan, Weixian Lei, Hanxuan Li, Honglin Li, Xiyun Li, Zaitang Li, Leowei Liang, Xin Luo, Haozhe Ma, Jiayi Mao, Zhoujie Pan, and 10 others. 2026. Ui-mate: Advancing open-weight foundation gui agents with in-context demonstrations. *Preprint*, arXiv:2608.15930.

Yue Fan, Lei Ding, Ching-Chen Kuo, Shan Jiang, Yang Zhao, Xinze Guan, Jie Yang, Yi Zhang, and Xin Eric Wang. 2024. Read anywhere pointed: Layout-aware gui screen reading with tree-of-lens grounding. *Preprint*, arXiv:2406.19263.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for llm agent training. *Preprint*, arXiv:2505.10978.

Shubham Gandhi, Jason Tsay, Jatin Ganhotra, Kiran Kate, and Yara Rizk. 2025. When agents go astray: Course-correcting swe agents with prms. *Preprint*, arXiv:2509.02360.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogagent: A visual language model for gui agents. *Preprint*, arXiv:2312.08914.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, Yuhuai Li, Shengze Xu, Shenzhi Wang, Xinchen Xu, Shuofei Qiao, Zhaokai Wang, Kun Kuang, Tieyong Zeng, Liang Wang, and 10 others. 2025. Os agents: A survey on mllm-based agents for general computing devices use. *Preprint*, arXiv:2508.04482.

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. 2024. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models. *Preprint*, arXiv:2312.03052.

Michael Krumdick, Charles Lovering, Varshini Reddy, Seth Ebner, and Chris Tanner. 2025. No free labels: Limitations of llm-as-a-judge without human grounding. *Preprint*, arXiv:2503.05061.

Dongxu Li, Yudong Liu, Haoning Wu, Yue Wang, Zhiqi Shen, Bowen Qu, Xinyao Niu, Guoyin Wang, Bei Chen, and Junnan Li. 2024a. Aria: An open multimodal native mixture-of-experts model. *Preprint*, arXiv:2410.05993.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024b. On the effects of data scale on computer control agents. *arXiv preprint arXiv:2406.03679*.

Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanjing Xiong, Fan Zhang, Xiang Li, and 6 others. 2024c. Personal llm agents: Insights and survey about the capability, efficiency and security. *Preprint*, arXiv:2401.05459.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. 2025. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *Preprint*, arXiv:2504.14239.

Junyu Lu, Songxin Zhang, Zejian Xie, Zhuoyang Song, and Jiaxing Zhang. 2025. Orcust: Stepwise-feedback reinforcement learning for gui agent. *Preprint*, arXiv:2509.17917.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. Omniparser for pure vision based gui agent. *Preprint*, arXiv:2408.00203.

Yong Min, Kun Zhou, Yifan Li, et al. 2026. Improving vision-language models with perception-centric process reward models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 33099–33109.

Songqin Nong, Xiaoxuan Tang, Jingxuan Xu, Sheng Zhou, Jianfeng Chen, Tao Jiang, and Wenhao Xu. 2026. Craft-gui: Curriculum-reinforced agent for gui tasks. *Preprint*, arXiv:2508.11360.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 467 others. 2026. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Fei Tang, Zhangxuan Gu, Zhengxi Lu, Xuyang Liu, Shuheng Shen, Changhua Meng, Wen Wang, Wenqi Zhang, Yongliang Shen, Weiming Lu, Jun Xiao, and Yueting Zhuang. 2025. Gui-g^2: Gaussian reward modeling for gui grounding. *Preprint*, arXiv:2507.15846.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, and 1332 others. 2025. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Shizuo Tian, Hao Wen, Yuxuan Chen, Jiacheng Liu, Shanhui Zhao, Guohong Liu, Ju Ren, Yunxin Liu, and Yuanchun Li. 2025. Agentprog: Empowering long-horizon gui agents with program-guided context management. *Preprint*, arXiv:2512.10371.

Wenkai Wang, Hongcan Guo, Zheqi Lv, and Shengyu Zhang. 2025a. A rolling stone gathers no moss: Adaptive policy optimization for stable self-evaluation in large multimodal models. *Preprint*, arXiv:2508.09155.

Wenkai Wang, Xiyun Li, Hongcan Guo, Wenhao Yu, Tianqing Fang, Haitao Mi, Dong Yu, and Shengyu Zhang. 2026a. Measure twice, click once: Co-evolving proposer and visual critic via reinforcement learning for gui grounding. *Preprint*, arXiv:2604.21268.

Wenkai Wang, Tao Xiong, Jingchen Ni, Yunpeng Bao, Xiyun Li, Tianqi Liu, Hongcan Guo, Zilong Huang, and Shengyu Zhang. 2026b. Deskcraft: Benchmarking desktop agents on professional workflows and human-in-the-loop collaboration. *Preprint*, arXiv:2606.03103.

Ziwei Wang, Leyang Yang, Xiaoxuan Tang, Sheng Zhou, Dajun Chen, Wei Jiang, and Yong Li. 2025b. History-aware reasoning for gui agents. *Preprint*, arXiv:2511.09127.

Yuyang Wanyan, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Jiabo Ye, Yutong Kou, Ming Yan, Fei Huang, Xiaoshan Yang, Weiming Dong, and Changsheng Xu. 2025. Look before you leap: A gui-critic-r1 model for pre-operative error diagnosis in gui automation. *Preprint*, arXiv:2506.04614.

Wenyi Wu, Kun Zhou, Ruoxin Yuan, Vivian Yu, Stephen Wang, Zhiting Hu, and Biwei Huang. 2025a. Auto-scaling continuous memory for gui agent. *Preprint*, arXiv:2510.09038.

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

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. 2024. Os-atlas: A foundation action model for generalist gui agents. *Preprint*, arXiv:2410.23218.

Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, Shuai Ren, Yafei Wen, Xiaoxin Chen, Aojun Zhou, and Hongsheng Li. 2025. Ui-genie: A self-improving approach for iteratively boosting mllm-based mobile gui agents. *Preprint*, arXiv:2505.21496.

Hai-Ming Xu, Qi Chen, Lei Wang, and Lingqiao Liu. 2024. Attention-driven gui grounding: Leveraging pretrained multimodal large language models without fine-tuning. *Preprint*, arXiv:2412.10840.

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

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, Jitong Liao, Qi Zheng, Fei Huang, Jingren Zhou, and Ming Yan. 2025. Mobile-agent-v3: Fundamental agents for gui automation. *Preprint*, arXiv:2508.15144.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. Appagent: Multimodal agents as smartphone users. *Preprint*, arXiv:2312.13771.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. 2025a. Instruction tuning for large language models: A survey. *Preprint*, arXiv:2308.10792.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025b. The lessons of developing process reward models in mathematical reasoning. *Preprint*, arXiv:2501.07301.

Congmin Zheng, Xiaoyun Mo, Xinbei Ma, Qiqiang Lin, Yin Zhao, Jiachen Zhu, Xingyu Lou, Jun Wang, Zhaoxiang Wang, Weiwen Liu, Zhuosheng Zhang, Yong Yu, and Weinan Zhang. 2026. Adaptive milestone reward for gui agents. *Preprint*, arXiv:2602.11524.

Kaizhi Zheng, Xuehai He, and Xin Eric Wang. 2024. Minigpt-5: Interleaved vision-and-language generation via generative vokens. *Preprint*, arXiv:2310.02239.

A Tool Details

This section provides the specific implementation details for the multi-granularity visual toolset $\mathcal{T}$ employed during the *Criterion-Guided Autoregressive Perception* phase. These tools enable the GUI-PRA to actively transition from global understanding to local inspection, resolving visual ambiguities in static screenshots. The input/output specifications are summarized in Table 6, with detailed functional descriptions provided below.

OmniParser: Global UI Perception. We utilize the OmniParser (Lu et al., 2024) to establish a structural understanding of the global GUI state. This tool operates in two stages to convert a raw screenshot into a semantic representation: (1) **Optical Character Recognition (OCR):** It first detects and recognizes all textual elements on the screen, extracting their semantic content and bounding box coordinates. (2) **Set-of-Mark (SoM) Annotation:** Leveraging the OCR results and icon detection, the module overlays numeric tags or bounding boxes on interacting elements. The output serves as the foundational "map" for the agent, providing a structured textual description (e.g., HTML/XML-like format) alongside the visually annotated image, enabling the LLM to ground its reasoning in the global layout.

ZoomInSubfigure: Local Detail Disambiguation. To address the *Visual Ambiguity* challenge where fine-grained details (e.g., checkbox states, dense tabular data, or small status icons) are lost in the global view, we introduce the `ZoomInSubfigure` tool. Specifically, we employ **Qwen3-VL-8B-Instruct** as the vision backbone to identify and interpret visual content within these specific regions. **Input:** A target bounding box $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$ derived from the OmniParser output or the agent's reasoning. **Operation:** The tool crops the original, high-resolution screenshot to the specified region to preserve native pixel density. This cropped view is then processed by the **Qwen3-VL** backbone to resolve local ambiguities. **Output:** A high-fidelity sub-image of the Region of Interest (RoI). This allows the model to inspect visual features (e.g., whether a toggle is "On" or "Off") with pixel-level clarity, effectively removing the noise from the surrounding GUI context.

Point: Local UI Element Grounding. The `Point` tool is engineered to validate element existence and determine precise interaction coordinates based on semantic intent. We employ Molmo-7B-D-0924 (Deitke et al., 2024), a state-of-the-art open-weights model, as the backbone.

- **Functionality:** It accepts a natural language description (e.g., "The search icon in the top right corner" or specific button text) and outputs the precise normalized coordinates (x, y).
- **Visual Feedback:** To make the result interpretable for the subsequent verification step, we post-process the output by overlaying a highly visible marker (e.g., a red pentagram) at the predicted coordinates on the screenshot. This explicitly visualizes *where* the agent believes the target element is located, allowing the Judge Agent to verify if the location aligns with the criteria (e.g., "Is the pointed element actually the 'Submit' button?").

Terminate: Investigation Convergence. This tool does not perform visual processing but serves as a control signal. It is triggered when the agent determines that the accumulated evidence from the history H^{inv} is sufficient to answer all verification criteria in C_t with high confidence, ending the autoregressive loop.

B Comprehensive Implementation Details

B.1 Efforts for Reproducibility

To ensure the reproducibility of our experimental results, we meticulously documented and controlled several key parameters and settings throughout the evaluation of GUI-PRA.

For the underlying GUI Agent, we deployed the model as a server using the vLLM framework, with the deployment random seed initialized to 42. To ensure deterministic behavior, the inference temperature was set to 0 for the base model and all other experimental modules. In contrast, for the online Best-of-N (BoN) sampling experiments, the temperature was set to 1.0 to encourage diversity. Specifically, we employed distinct random seeds (i.e., $[30, 42, 3407, 114514, 256, 64, 1024, 2]$) to generate the eight candidate trajectories during this phase. During the base model testing phase, a random seed of 42 was used.

In the configuration of GUI-PRA's components, we set the number of criterion sampling attempts to 5, ensuring the selection of the most relevant experiential references. The maximum number of routing attempts for the autoregressive UI tools component was capped at 4.

Tool	Input	Output	Description
`OmniParser`	Image	SoM + BBox	Text-driven object detection
`Point`	Text + Image	Coordinates	Object localization
`ZoomInSubfigure`	BBox + Image	Sub-image	Detail disambiguation
`Terminate`	–	–	Terminate loop

Table 6: Input and Output Interfaces of the multi-granularity visual toolset $\mathcal{T}$ for interface analysis.

B.2 Benchmark Details and Evaluation Protocol

AndroidWorld. We evaluate on the standard M3A setting, which uses AndroidWorld as a dynamic benchmark for long-horizon tasks in real applications.

Mobile-MiniWoB++. We use the Pure Vision setting for mobile-style single-page tasks with dense UI elements and fine-grained grounding requirements.

OS-Critic Bench. We evaluate only the mobile subset, since the offline critic experiment is designed to test whether a supervisor can distinguish valid transitions from execution errors.

Protocol. Online evaluation uses Best-of-N rejection sampling with temperature $\tau = 1.0$. The supervisor scores each candidate step and selects the highest-scoring action for execution. Offline evaluation ranks pre-collected trajectories directly.

B.3 Data Construction Statistics

To construct the principle repository (P), we employed a rigorous filtering process. For the online experiment repository, we sampled 50 positive and 50 negative trajectories from the validation set using the Qwen3-VL-8B-Instruct. Following synthesis by the GPT-5.1 Meta-Judge and subsequent deduplication, we obtained a compact set of **64** verification principles. For the offline experiments, we scaled the collection to include 150 positive and 150 negative samples. After iterative merging and refinement, this yielded **143** and **119** distinct principles for the Qwen3-VL and Qwen2.5-VL based experiments, respectively.

C Additional Analyses

This section reports additional analyses of actor coverage, latency, tool reliability, and reproducibility that complement the main experimental results.

C.1 Actor Coverage and the Hard-Task Boundary

GUI-PRA is a verifier and selector, not a replacement for long-horizon actor planning. We therefore ran eight independent Qwen3-VL-8B executions on the same 116 AndroidWorld tasks and computed a hindsight task-level pass@8 reference: a task is covered when any complete trajectory succeeds. This is an empirical coverage reference, not a step-level oracle. As shown in Table 7, the Standard-PRM-to-reference gap is concentrated on Easy tasks (+5.61 points), with only +0.86 on Medium and no gap on Hard. GUI-PRA closes 5.17 of the 6.46 overall points (about 80%) and matches the Easy reference. Hindsight covers only 5/116 Hard tasks (4.31%), so a verifier has little headroom when the actor has not generated a valid continuation; it can reject a plausible wrong action, but cannot create a missing branch.

Table 7: Task-level actor coverage on 116 AndroidWorld tasks. Hindsight pass@8 is an empirical complete-trajectory coverage reference, not a step-level oracle.

Reference	Easy	Med.	Hard	Overall
Standard PRM	30.60	14.66	4.31	49.57
GUI-PRA	**36.21**	14.66	3.88	**54.74**
Hindsight pass@8	**36.21**	**15.52**	**4.31**	**56.03**

C.2 Latency, Tool Reliability, and Reproducibility

The diagnostics in this section come from two separately logged AndroidWorld evaluation snapshots. The latency and tool-call audit uses a 1,475-step runtime log, whereas the criterion-frequency analysis uses a separate log containing 1,483 distinct evaluation states. These counts refer to different units and snapshots, so they are not expected to match one-to-one.

The 1,475-step runtime log adds 6.70 seconds per evaluation step and uses 1.17 active visual-tool calls per step, or 1.48 total tool calls when `Terminate` is included. In a matched seven-step

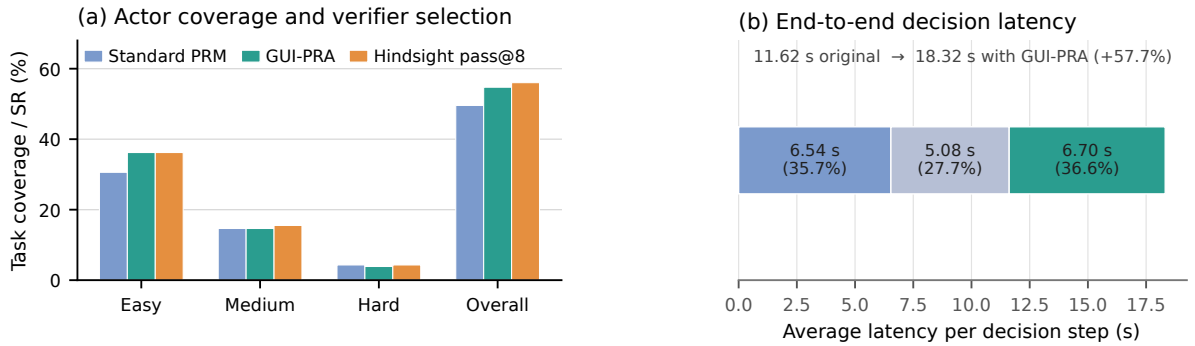


Figure 5: Additional diagnostics. (a) Task-level hindsight coverage measures the complete trajectories available to the verifier. (b) The eight-candidate decision-time breakdown separates actor generation, passive PRM selection, and GUI-PRA investigation.

Metric	Symbol	Value
Total evaluation steps	N_{step}	1,475
Total wall-clock overhead	T_{total}	9,880.0 s
Avg. overhead per step	$\bar{T}_{\text{step}}$	6.70 s
Active visual calls / step	$\bar{C}_{\text{active}}$	1.17
Total tool calls / step	$\bar{C}_{\text{total}}$	1.48

Table 8: Latency profile of GUI-PRA on AndroidWorld. Active visual calls exclude `Terminate`; total tool calls include it.

timing trace, actor candidate generation ($N = 8$), Standard PRM selection, and GUI-PRA investigation take 6.54, 5.08, and 6.70 seconds, respectively, for an estimated 18.32-second decision. Investigation therefore accounts for 36.6% of the full decision and increases the original 11.62-second pipeline by 57.7% (Figure 5). Candidate evaluations are independent and can be parallelized, but the underlying computation remains a substantial cost. A compute control gives PRM-SC3 48.28% SR, below Standard PRM CoT (49.57%) and GUI-PRA (54.74%), indicating that criterion-relevant evidence, rather than repeated unchanged scoring, drives the gain. The 6.70-second increment is measured investigation overhead, not a fixed per-step sequence: calls are made only when evidence is insufficient and routing is capped at four attempts.

Table 9: Timing breakdown under the eight-candidate AndroidWorld protocol.

Component	s/step	Share
Actor generation ($N = 8$)	6.54	35.7%
Standard PRM selection	5.08	27.7%
GUI-PRA investigation	6.70	36.6%
Full GUI-PRA decision	**18.32**	**100.0%**

We audited 164 logged decision steps (Table 10). When OmniParser or Point explicitly reports missing evidence, the router preserves that gap and can invoke a complementary tool or choose a state-exposing action; the Clock/Stopwatch, Wi-Fi/Bluetooth, and Settings traces all recover this way. Silent semantic misgrounding remains harder: Point repeatedly maps the Google Lens shortcut to Camera, producing a Home–Lens oscillation and failure. GUI-PRA therefore mitigates overt tool failure but does not guarantee robustness to syntactically valid, semantically wrong evidence.

The runtime components are modular and replaceable: OmniParser supplies global structure, Molmo supplies point grounding, and Qwen3-VL interprets local crops. Deterministic modules use temperature 0; Best-of-N uses temperature 1.0 with fixed seeds; criterion synthesis uses five attempts and routing is capped at four. The implementation, prompts, principle repository, and repository-construction scripts will be released with the camera-ready artifacts. Principles are distilled only from AndroidControl (Li et al., 2024b), while AndroidWorld and web-centric Mobile-MiniWoB++ provide cross-environment evaluation evidence; desktop and highly dynamic web interfaces remain future work.

D Failure-Mode Alignment Analysis

We also analyze 50 representative interaction trajectories sampled from AndroidWorld and manually annotate the main reason why the Standard PRM gives an unreliable score. As shown in Figure 6, 46% of the sampled trajectories involve an evaluative knowledge gap, and 34% involve visual ambiguity. Some trajectories involve both. The 16% overlap suggests that the two problems often appear together, which is why we couple criterion

Trace	Tool issue	GUI-PRA response and outcome
`ClockStopWatchRunning` *Step 0*	OmniParser returns no relevant element, and Point reports `Coordinates not found`.	The router preserves the evidence gap, invokes `ZoomInSubfigure`, and selects opening the app drawer. The task completes successfully.
`TurnOffWifiAnd TurnOnBluetooth` *Step 0*	No relevant connectivity control is parsed, and Point cannot locate the Wi-Fi icon.	The router zooms into the status-bar region and selects opening Quick Settings. The task completes successfully.
`TurnOnWifiAndOpenApp` *Step 6*	OmniParser detects only `Messages`, and Point cannot locate the Settings app.	The router invokes `ZoomInSubfigure` and selects scrolling rather than fabricating a coordinate. Settings becomes visible and is opened successfully.
`CameraTakePhoto` *Steps 0/2/4/6/8*	Point maps the Google Lens shortcut to Camera across repeated steps.	The apparently valid but incorrect coordinate is repeated, causing Home–Lens oscillation and task failure.

Table 10: Tool-reliability audit of 164 logged decision steps. Explicit failures can trigger complementary investigation, whereas silent semantic misgrounding can cascade.

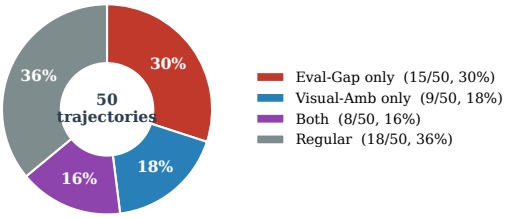


Figure 6: Failure-mode breakdown on 50 sampled AndroidWorld trajectories.

synthesis with active perception.

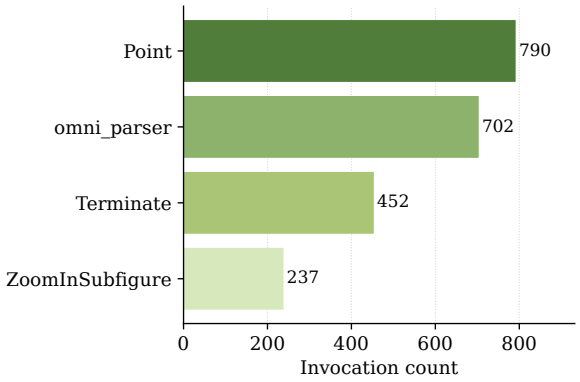
The examples below show what these categories mean in practice. On *SportsTrackerActivitiesCountForWeek*, the Standard PRM over-rewards a premature `terminate` action because the app's main title is already visible. GUI-PRA instead applies a criterion requiring the evaluator to confirm that the expected interaction pattern has been completed, and it selects a `swipe` action that exposes the next task-relevant state. On dense-UI tasks such as *MarkorDeleteNewestNote* and *ExpenseDeleteDuplicates2*, static global-view scoring cannot reliably distinguish visually similar candidates. In these cases, GUI-PRA invokes `ZoomInSubfigure` to isolate the relevant region and ground the verdict in local visual evidence. This analysis complements the aggregate results and helps explain why the gains concentrate on solvable Easy and Medium tasks, while Table 3 shows that both principles and tools are necessary.

E Investigation Profile and Criterion Diversity

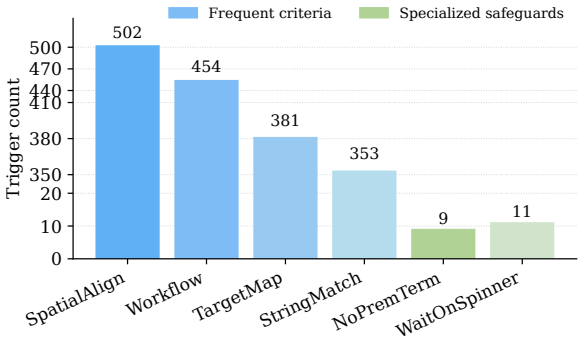
We inspect the investigation process by logging tool calls and synthesized criteria during the AndroidWorld run with Qwen3-VL-8B-Instruct (Figure 7). The tool-call panel uses the 1,475-step runtime log described above, while the criterion panel uses the separate 1,483-state criterion-analysis log.

Tool calls are selectively allocated. Across 1,475 runtime decision steps, GUI-PRA invokes `Point` 790 times, `omni_parser` 702 times, `ZoomInSubfigure` 237 times, and `Terminate` 452 times. The first three tools total 1,729 calls, or $\approx$1.17 active visual-tool calls per step; including `Terminate`, the total is 2,181 calls, or $\approx$1.48 calls per step. `Point` and `omni_parser` account for most active calls because they provide routine spatial grounding and global layout evidence. `ZoomInSubfigure` is used less frequently, suggesting that local inspection is triggered selectively when global evidence is insufficient rather than being applied uniformly as extra computation.

Criteria remain state-adaptive. Out of 1,483 distinct evaluation states, GUI-PRA instantiates 61 unique criteria. Nine head criteria are triggered more than 300 times each, and 22 are triggered more than 100 times, indicating that the model repeatedly uses several broad GUI-logic dimensions rather than collapsing to a single fallback rule. The four most frequent criteria correspond to dialog spatial alignment (502), workflow progression (454), target mapping for lists and icons (381), and exact string matching for text entry (353). Long-tail



(a) Tool invocation frequency across 1,475 runtime decision steps. GUI-PRA makes an average of 1.17 active visual-tool calls per step, or 1.48 total calls when `Terminate` is included.



(b) Criterion utilization with a piecewise scale. In a separate 1,483-state criterion-analysis log, GUI-PRA instantiates 61 unique criteria, with frequently reused checks and rarer specialized safeguards.

Figure 7: **Investigation profile of GUI-PRA on AndroidWorld (Qwen3-VL-8B-Instruct).** The two panels use separately logged snapshots and therefore have different denominators. (a) Tool calls are selectively allocated across routine grounding and local inspection. (b) Criteria combine frequently reused verification routines with specialized safeguards for rarer UI states.

criteria handle rarer but consequential states. For example, the rule "*do not terminate solely because a main title or header is visible*" triggers 9 times, while "*when a loading indicator is visible, prefer a wait action*" triggers 11 times. This distribution indicates that the Principle Repository provides both reusable verification routines and specialized safeguards for deceptive UI states.

F Detailed Case Study

Figure 8 illustrates a complete operational flow of our GUI-PRA framework, showcasing its interaction with a base GUI Agent to fulfill a user's request. The top row depicts the trajectory of the base GUI Agent. It correctly executes all the positive data entry sub-tasks: it navigates to the contacts application, initiates the creation of a new contact, and accurately inputs the name, phone number, and label.

In parallel, the bottom row shows the continuous monitoring and reasoning process of our GUI-PRA. At each step, GUI-PRA leverages its UI Tools to perceive the screen, grounding the agent's actions and the state of the UI elements, as evidenced by the 'Icon' and '<point>' outputs. The primary challenge here is not the data entry itself, but correctly interpreting the negative constraint: "Do NOT hit save." GUI-PRA excels by continuously validating the agent's progress against this complex goal. It correctly determines the precise moment the task is finished, instructing the agent to terminate rather than incorrectly proceeding to save.

This intervention prevents the GUI Agent from making an irreversible error that would have resulted in task failure. This case highlights GUI-PRA's ability to provide nuanced, process-level supervision that goes beyond simple action validation, ensuring strict adherence to complex user constraints.

As shown in Figure 9, this second example illustrates the same supervision pattern on a different task type. On `NotesTodoItemCount`, the goal requires the agent to enumerate todo items inside a notes view and report a final count. The base agent tends to terminate based on a partial view of the list, returning a count that omits items below the fold or hidden inside collapsed sections. GUI-PRA instead instantiates a counting-completeness criterion that explicitly requires verifying whether every relevant item has been observed, and uses active perception to inspect the list region before adjudicating termination. Together, the two cases show that process supervision is most useful when the verifier must enforce task-specific completion semantics—negative constraints in `ContactsNewContactDraft` and counting completeness in `NotesTodoItemCount`—rather than a single generic notion of progress.

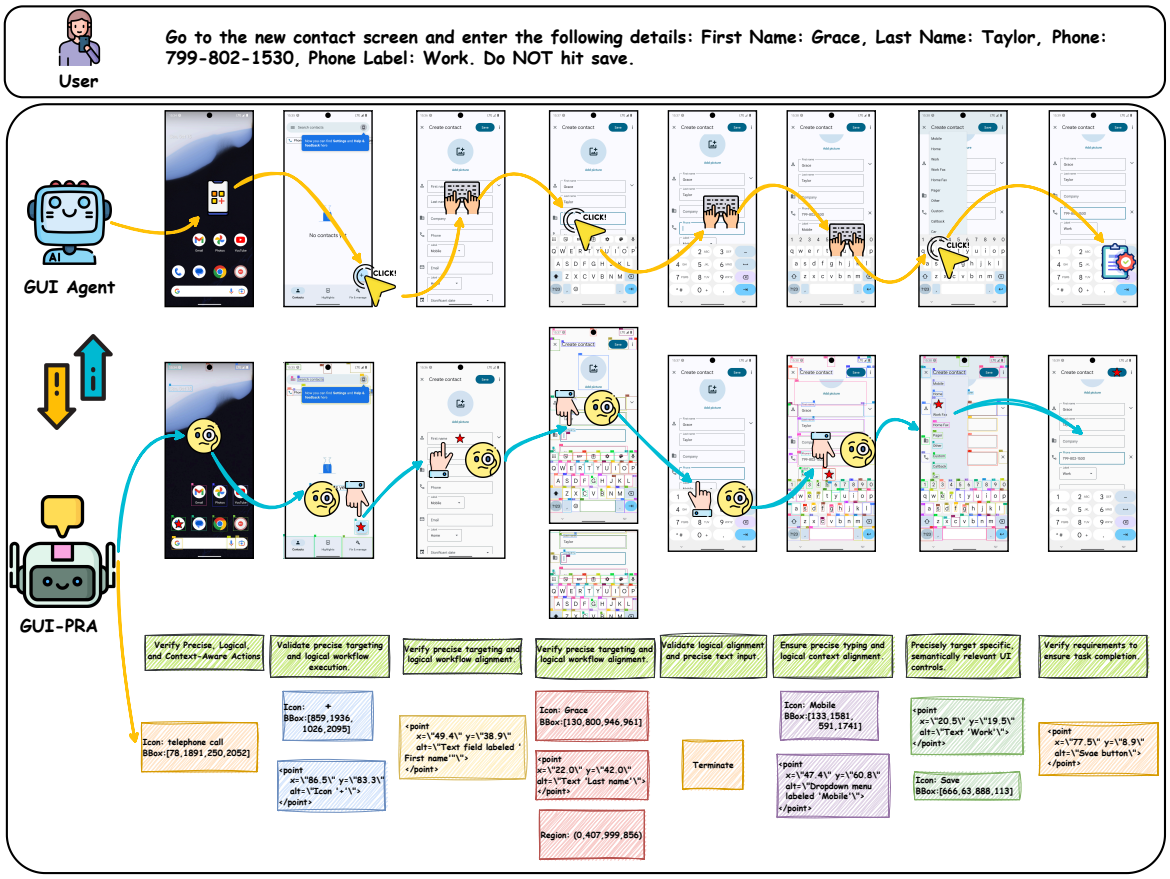


Figure 8: A complete case of GUI-PRA guiding a GUI Agent to complete the 'ContactsNewContactDraft' task. The figure illustrates the parallel process flows, showing the agent's action trajectory (top row) and the continuous supervision provided by GUI-PRA (bottom row) across multiple steps until task completion.

G PROMPTS

We provide the prompts in constructing GUI-PRA below.

G.1 GUI-PRA: Experience-Injected Criterion Synthesis

GUI-PRA: Criterion retrieval

SYSTEM:
You are an expert in Android UI automation. Select the 5 most relevant experiences/guidelines for the current situation.

USER:
Current Goal: {goal}
Available Experiences: {exp_list_str}
Select the 5 most relevant experiences that help with the current step. Output ONLY the indices of the selected experiences as a comma-separated list (e.g., 0, 5, 12).

G.2 GUI-PRA: Criterion-Guided Autoregressive Perception

GUI-PRA: UI Tool Routing

SYSTEM:
You are a visual assistant for GUI process verification. Your goal is to inspect the current UI state and collect only the visual evidence needed by a downstream process-reward judge. Do not answer the task yourself.

Here are the available tools:

- **`Point`**: Identifies a specific point in the image based on a description and returns coordinates. Use when the current evidence does not localize a required UI icon or text target. Example:

```
{"name": "Point", "arguments": {"image": "img_1", "param": "Icon 'Gmail'"}}
```

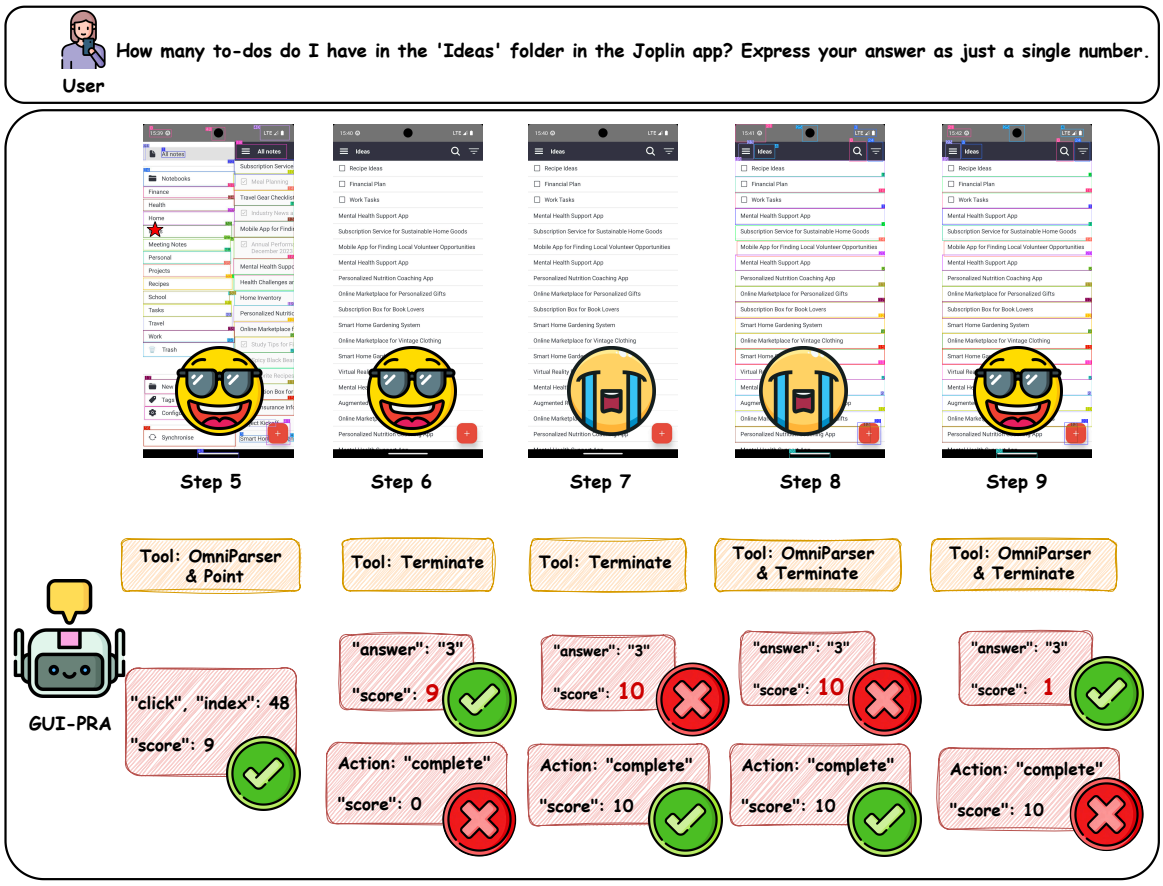


Figure 9: A second case on the `NotesTodoItemCount` task. The agent prematurely terminates before all todo items have actually been observed, while GUI-PRA enforces a counting-completeness criterion and only accepts termination after every relevant item has been verified through active perception.

- **`omni_parser`**: Parses a UI or general image to detect, label, and locate all interactive and non-interactive elements like icons, buttons, and text fields. Use it when a global inventory is needed and the current evidence is incomplete. Example:

  `{"name": "omni_parser", "arguments": {"image": "img_1"}}`

- **`ZoomInSubfigure`**: Crops the image to the specified subfigure. Use when a specific region remains visually ambiguous or illegible. Example:

  `{"name": "ZoomInSubfigure", "arguments": {"image": "img_1", "param": "File Creation Panel: Naming & Format"}}`

- **`Terminate`**: Ends the investigation and returns the result for downstream scoring. Use when all required evidence has been gathered. Example:

  `{"name": "Terminate", "arguments": {"ans": "1985"}}`

To gather relevant information:

1. Assess the type of question provided. Based on the problem type (e.g., locating specific data points, extracting text, analyzing regions, etc.), select the most suitable tool or combination of tools.
2. If segmentation or line drawing requires coordinates that are not already available, use the `Point` tool to identify them.
3. Use the selected tools logically and sequentially, with each tool building on the output of the previous one.
4. Use a visual tool only for an evidence gap; if no gap remains, call `Terminate`.

Example Output:
Example 1:

```
{
"thought": "The current evidence does not expose all interactive elements needed to evaluate the candidate action. I will use the omni_parser tool to fill this specific evidence gap and provide the downstream judge with a global UI inventory.",
"actions": [
{"name": "omni_parser", "arguments": {"image": "img_1"}}
]
}
```

Example 2:

```
{
"thought": "The current evidence does not localize the text target needed to evaluate the candidate action. I will use the Point tool to obtain the coordinates of 'Sun, Oct 15' and close this evidence gap.",
"actions": [
{"name": "Point", "arguments": {"image": "img_1", "param": "Text 'Sun,Oct 15'"}}
]
}
```

Example 3:

```
{
"thought": "The current view leaves the file-naming field ambiguous, so the evidence is insufficient for evaluation. I will use ZoomInSubfigure to isolate 'File Creation Panel: Naming & Format' and inspect that region at higher resolution.",
"actions": [
{"name": "ZoomInSubfigure", "arguments": {"image": "img_1", "param": "File Creation Panel: Naming & Format"}}
]
}
```

If further action is required, continue building on the previous step with the correct tool.

GUI-PRA: UI Tool Routing

USER:
User Question: <*initial_prompt*>

Relevant Judge Experiences/Guidelines: <*experiences_text*>
You have already taken some steps. Here is the history of your actions and their observations:
Current tool calling history: <*history_str*>

Your Task (OI - Observation & Introspection):
Summarize: Briefly summarize what you have learned from the history.
Decide: Based on your summary and the initial goal, decide on the next step. Is the evidence sufficient for the downstream process-reward judge to evaluate the current candidate action?

- **If YES**, call the tool: `{"name": "Terminate", "arguments": {"ans": "<your final answer>"}}`
- **If NO**, call the tool that directly addresses the identified evidence gap.

Avoid repeats unless the UI state changes.

G.3 GUI-PRA: Best of N Selection

GUI-PRA: BoN Selection

SYSTEM:
You are a Process Reward Model (PRM). Evaluate the candidate action based on the user's instruction, screen image, and history.

GUI Agent Action Space and Execution Rules. A candidate action must follow the declared GUI-agent JSON schema and be executable from the current screen state. Evaluate one action at a time; treat `Terminate` as valid only when the task is complete and all explicit constraints are satisfied.

CRITICAL: Task Constraint Compliance (MUST CHECK FIRST)
Before scoring, identify ALL explicit constraints in the user's instruction (e.g., "in order", "one by one", "first... then...", "all of", specific sequences).

- If the action **violates any explicit constraint**, the score MUST be ≤ 3, regardless of how "reasonable" the action seems otherwise.
- Example: If instruction says "add songs in order: A, B, C" and the action adds B first, this is a FATAL violation (score 0-2).

Evaluation Criteria (in priority order):

1. **Constraint Compliance**: Does the action respect ALL explicit requirements in the instruction? (e.g., order, sequence, completeness). Violations = automatic low score.
2. **Reasoning Accuracy**: Does the reasoning correctly understand the current state AND the task requirements? Penalize reasoning that ignores or misinterprets constraints.
3. **Progress**: Does the action move toward the goal in the CORRECT way (respecting constraints)?
4. **Efficiency**: Is it optimal? Avoids loops and redundancy.
5. **Format Compliance**: Action JSON must follow the defined schema strictly.

Scoring (0-10) - Be Discriminative:

- **9-10: Perfect** - respects ALL constraints, correct reasoning, optimal action.
- **7-8: Good** - respects constraints, minor inefficiencies or slightly suboptimal path.
- **4-6: Acceptable** - respects constraints but has weak reasoning or unnecessary steps.
- **2-3: Poor** - violates a constraint OR has significant reasoning errors.
- **0-1: Critical failure** - violates core task constraints, completely wrong action, or false completion.

Output Format:
First, provide a brief explanation of your evaluation, detailing why you assigned the score. Then, output the score enclosed within `<score>` and `</score>` tags.

GUI-PRA: BoN Selection

USER:
Please evaluate the following candidate action.

User's Instruction:

- **User Goal:** {goal}
- **Action History:**
  {history}
- **Relevant Experiences:**
  {experiences}

Evaluate the candidate action based on the background information provided above. Strictly follow the content and requirements mentioned in the Agent Capabilities and Guidelines. Ensure the action is valid within the current UI state and history.

Candidate Action to Evaluate:
{action}

Tool Information:

The information below is tool-using information which will help you better score the candidate action in the PRM stage.
Relevant tool introductions:

- **`Point`**: Identifies a specific point in the image based on a description and returns coordinates. Use when the current evidence does not localize a required UI icon or text target. Then use a red five-pointed star to highlight the coordinates in the image. Example: `{"name": "Point", "arguments": {"image": "img_1", "param": "Icon 'Gmail'"}}`
- **`omni_parser`**: Parses a UI or general image to detect, label, and locate all interactive and non-interactive elements like icons, buttons, and text fields. Use it when a global inventory is needed and the current evidence is incomplete. Example: `{"name": "omni_parser", "arguments": {"image": "img_1"}}`
- **`ZoomInSubfigure`**: Crops the image to the specified subfigure. Use when a specific region remains visually ambiguous or illegible. Example: `{"name": "ZoomInSubfigure", "arguments": {"image": "img_1", "param": "File Creation Panel: Naming & Format"}}`

TOOL_INFORMATION: {tool_chain_filter}

[Attached Screen Images & Tool Result Images]

Based on the tool information, carefully analyze the candidate action's correctness and format, and assign a granular score based on how helpful the action is for completing the task.

G.4 PRM: Best of N Selection

PRM: BoN Selection

`SYSTEM:`
You are a **Process Reward Model**. Your task is to evaluate a single **candidate action step** based on a **user's instruction** and **provided screen image** (screenshots of the most recent steps, with a maximum of 5).

{task_template}

Evaluation Process:

1. **Understand the Goal:** Carefully review the user's instruction and the current screen image.
2. **Determine Your Optimal Action:** Based on the instruction and image, decide what you believe is the best possible action step.
3. **Evaluate the Candidate Action:** Compare the provided candidate action step against your optimal action.
4. **Assign a Score:** Assign a numerical score to the candidate action from 0 to 100. If the candidate action is correct and has a correct reasoning process, a higher score should be given.

Output Format:
First, provide a brief explanation of your evaluation, detailing why you assigned the score. Then, output the score enclosed within `<score>` and `</score>` tags.

PRM: BoN Selection

`USER:`
Please evaluate the following candidate action based on the user's instruction and the provided screen image (screenshots of the most recent steps, with a maximum of 5), following all guidelines from the system prompt.

User's Instruction:
{goal}

History:
{history}

Candidate Action to Evaluate:
{candidate_response}

Your evaluation should include a reasoning process and a score wrapped in `<score></score>` tags.

H Statement on the Usage of Large Language Models

During the preparation of this manuscript, a Large Language Model (LLM) was utilized as an auxiliary tool. Its application was strictly limited to

improving the language and readability of the text, as well as assisting with the formatting of figures. The authors have meticulously reviewed and edited all machine-generated suggestions to ensure the scientific accuracy and integrity of the final content, for which they take full responsibility.