GUI-Shepherd: Reliable Process Supervision for Policy Learning and Verification in GUI Agents

Anonymous ACL submission

Abstract

While Large Multimodal Models have empowered GUI agents, their performance on long-horizon tasks remains fragile due to the sparsity of outcome-based rewards and the challenge of credit assignment. To bridge this gap, we introduce **GUI-Shepherd**[1], a comprehensive framework for process supervision that provides dense, step-by-step guidance. Central to our approach is a rigorously constructed dataset of 52k interactions, curated via a hybrid pipeline that combines human-verified correctness labels with GPT-4o generated rationales. This high-fidelity data enables GUI-Shepherd to function as both a precise reward provider for Reinforcement Learning (RL) and a robust verifier for inference. To our knowledge, this work represents the first systematic study of process supervision in the GUI domain, spanning online multi-turn training, offline policy refinement, and test-time verification. Extensive experiments demonstrate its effectiveness: on the challenging online AndroidWorld benchmark, GUI-Shepherd achieves a 7.7 point absolute improvement via PPO, significantly outperforming sparse-reward baselines. These benefits extend to offline settings and inference-time verification, yielding gains of 2.2 and 5.1 points, respectively. Collectively, our results substantiate that reliable process supervision is instrumental in complex GUI automation.

1 Introduction

Developing agents (Zhang et al., 2025a; Zhang and Zhang, 2023) to execute human instructions via Graphical User Interface (GUI) is a critical AI frontier. Although Large Multimodal Models (Bai et al., 2025; Chen et al., 2024; Liu et al., 2023) provide the foundational perceptual and reasoning capabilities for such agents, their practical utility is severely hampered by poor performance on multi-turns long-horizon tasks (Xu et al., 2024a). This limitation is stark: despite excelling at single-step actions, these agents frequently struggle with complex long-sequence workflows.

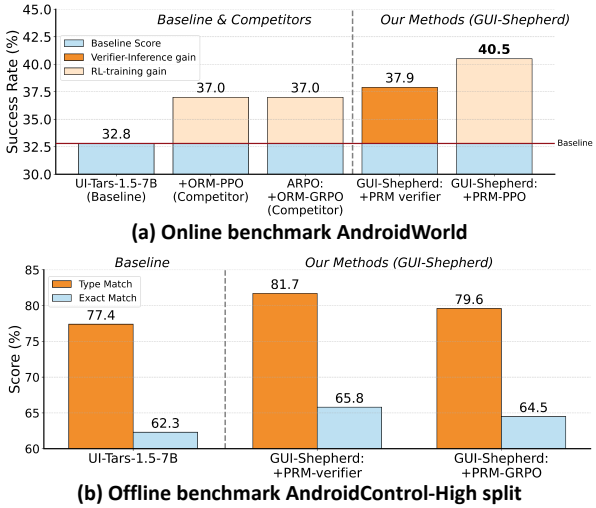


Figure 1: **GUI-Shepherd** largely enhances performance in online long-sequence and offline single-step tasks.

Such workflows are characterized by dynamic system feedback and inherent stochasticity, which introduce significant variability across different execution runs. To navigate this uncertainty, agents must possess adaptive planning and reasoning skills, rather than relying on the rote memorization of static expert demonstrations. This presents a challenge for Supervised Fine-Tuning, which is constrained by both the practical difficulty of scaling vast data requirements and the conceptual limitation that imitation may be an inefficient path toward robust reasoning. While Reinforcement Learning (Rafailov et al., 2023; Shao et al., 2024; Guo et al., 2025) offers a theoretically aligned framework for long-horizon planning, its application to GUI tasks (Lu et al., 2025a) has been hindered by the sparse signals from conventional Outcome Reward Model(ORM). As illustrated in Figure 2, an ORM provides a single terminal sig-

[1]Our code can be found at `https://anonymous.4open.science/r/gui_shepherd`, and will be released publicly upon publication.

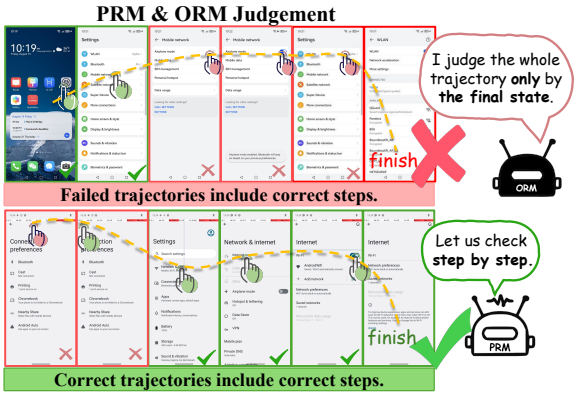


Figure 2: PRM vs. ORM. An ORM judges a trajectory by its final state, failing to **identify critical errors, assign credit to correct steps, or penalize suboptimal actions.** While a PRM evaluates each action individually, it provides a more accurate signal.

nal that fails to **pinpoint the exact step causing failure, nor can it filter out suboptimal actions from successful trajectories.**

The limitations of outcome-based signals in the GUI domain motivate a shift towards denser, more informative reward signals. We therefore adopt a fundamentally different approach: Process-based Reward Models (PRMs) (Luo et al., 2024), which provide step-by-step feedback and have been proven effective in complex math reasoning tasks (Lightman et al., 2023). Our paper presents a systematic investigation into the training of a robust PRM for GUI agents, and is the first to comprehensively validates its efficacy across a wide range of agent workflows: online/offline RL and inference-time verification, as shown in Table1.

To construct a reliable PRM, we develop a dual-pipeline methodology to curate a diverse 52k dataset, balancing two critical axes: **temporal diversity** capturing states across all stages of complete trajectories and **UI diversity** spanning a wide range of application interfaces. To achieve temporal diversity, we generate full trajectories in an interactive environment. Concurrently, to enrich the diversity of UI, we sample single-step states from a highly diverse dataset (Li et al., 2024) to ensure a vast range of UI layouts. Our annotation process revealed a crucial insight: even sota VLM like GPT-4o (Hurst et al., 2024) still exhibits a significant performance gap compared to human experts, underscoring the necessity of human annotation for the core correctness labels. We therefore adopt a hybrid strategy to balance quality and cost: human annotators provide the high-reliability binary correctness scores, while GPT-4o generates

Methods	Online RL Training	Offline RL Training	Inference Verification
GUI-Critic-R1	✗	✗	✓
VEM	✗	✓	✗
GuidNav	✗	✗	✓
GUI-PRA	✗	✗	✓
GUI-Shepherd	✓	✓	✓

Table 1: Comparison with existing PRM-based methods in the GUI domain. GUI-Shepherd represents the first systematic exploration of process supervision across the full spectrum: online RL training, offline RL training, and inference-time verification.

the explanatory chain-of-thought reasoning.

We validate GUI-Shepherd's impact in the most demanding setting: **as a dense reward provider for online RL in a dynamic environment, a challenging frontier that remains largely unexplored**. When integrated with Proximal Policy Optimization (PPO) (Schulman et al., 2017) on the AndroidWorld benchmark, GUI-Shepherd provides dense and reliable rewards to guide the agent's training, leading to a 7.7 point improvement in success rate. As shown in Figure 1, this result significantly outperforms strong ORM-based competitors by 3.5 points, providing strong empirical evidence for the superiority of process-based rewards in complex interactive tasks. Furthermore, when deployed as an inference-time verifier to re-rank candidate actions, GUI-Shepherd boosts the base agent by 5.1 points, demonstrating its dual utility to enhance performance without requiring any additional training or parameter updates for the policy model.

To demonstrate that reliable process supervision provides a fundamental performance boost for both immediate step prediction and long-horizon execution, we test GUI-Shepherd on single-step action prediction on an offline dataset AndroidControl (Li et al., 2024), which consists of pre-collected trajectories. GUI-Shepherd further demonstrates its efficacy, improving Type Match of `High Split` as an inference verifier (+4.3) and an offline RL reward provider (+2.2). This success across different domains substantiates that step-by-step process supervision is a powerful and broadly applicable paradigm for advancing GUI automation.

In summary, our contributions are:

1. We pioneer PRM-guided online RL for long-horizon GUI tasks, tackling sparse rewards with a 7.7 point gain on AndroidWorld, establishing a new paradigm for reliable credit assignment in complex workflows.

2. We develop a novel dual-pipeline collection and hybrid annotation methodology to curate a high-quality dataset, ensuring the temporal and UI diversity and label accuracy prerequisite for training a reliable PRM.

3. We conduct the first systematic validation of process supervision across diverse scenarios of GUI automation, verifying versatility as an inference verifier and offline RL reward provider beyond online training.

2 Related Work

2.1 Training Paradigms for GUI Agents

The field of GUI agents has rapidly evolved from non-visual models that processed structured inputs XML (Guo et al., 2024) to modern agents based on VLMs that perceive raw screen pixels (Cheng et al., 2024). Methodologically, the paradigm has shifted from SFT, which focuses on fundamental grounding skills (Hong et al., 2024), towards reinforcement learning (Bai et al., 2024) to improve decision-making. A significant body of work has demonstrated success in training agents for single-step actions, typically using offline RL on offline datasets with rule-based rewards (Luo et al., 2025; Lu et al., 2025b). However, this success has been largely confined to single-step or short-horizon settings. The majority of current methods exhibit a sharp performance degradation when faced with complex, long-sequence tasks. Our work directly confronts this challenge, demonstrating that reliable process supervision is pivotal for mastering these complex workflows.

2.2 From Static to Dynamic Evaluation

Benchmarks for GUI agents can be categorized by single-step action prediction and end-to-end task completion. A larger body of work focuses on single-step action evaluation, which is typically conducted offline using offline datasets of expert trajectories (Rawles et al., 2023; Zhang et al., 2024; Lu et al., 2024). This paradigm, where an agent's prediction is compared against a ground-truth action, is well-suited for the supervised and offline RL methods. However, evaluating an agent's ability to complete long-sequence tasks is more complex, necessitating dynamic, interactive environments that respond to agent actions in real-time. This has led to the development of sophisticated online benchmarks like OSWorld (Xie et al., 2024) for desktop and AndroidWorld (Rawles et al., 2024) for mobile platforms. It is in these challenging, long-horizon environments that current agents falter. Our work is therefore focused on improving task success rates on AndroidWorld, which comprehensively evaluates the end-to-end capabilities of agents.

2.3 Process Supervision

Process Reward Modeling (Luo et al., 2024; Khalifa et al., 2025) has become a key technique for overcoming the sparse reward and credit assignment challenges in complex, multi-step reasoning tasks. Its efficacy is well-established in domains like mathematics, where PRMs serve a dual role: providing dense rewards for reinforcement learning and verifying steps at inference time (Wang et al., 2023; Lightman et al., 2023; Ma et al., 2023; Zhu et al., 2025). Inspired by this success, we investigate the application of PRMs to GUI agents. While existing GUI agent research has touched upon related concepts, such as learning critic (Wanyan et al., 2025; Hu et al., 2025; Xiong et al., 2025) or value models (Zheng et al., 2025) to **estimate expected future returns**, the systematic development and application of a dedicated PRM **for immediate step correctness** remains an open area.

3 Constructing Process Reward Model

A long-sequence GUI task with a high-level instruction I requires an agent to execute a trajectory of state-action pairs, $\tau = (s_0, a_0, \ldots, s_T, a_T)$, where T is the number of steps, s_t is the screen screenshot at time step t, and a_t is a GUI operation (*e.g.*, click, type). Standard RL training that rely on an ORM to assign a single, terminal reward to the entire trajectory are limited by reward sparsity, which provides inadequate supervision for intermediate steps. To address this, we introduce a Process Reward Model that provides dense, step-wise supervision. Specifically, our PRM is formulated as a function that assesses the correctness of an action a_t at a given state s_t conditioned on the instruction I. It outputs a binary classification label, $r_t \in \{\text{positive}, \text{negative}\}$, indicating the quality of the step:

$$r_t = \mathcal{R}_{\text{PRM}}(I, s_t, a_t) \tag{1}$$

Building a reliable PRM capable of providing such supervision is a primary challenge. We establish a systematic methodology of three key pillars: data preparation, annotation paradigms, and training.

Training Data Preparation. The training data for our PRM consists of quadruplets, (I, s, a, r),

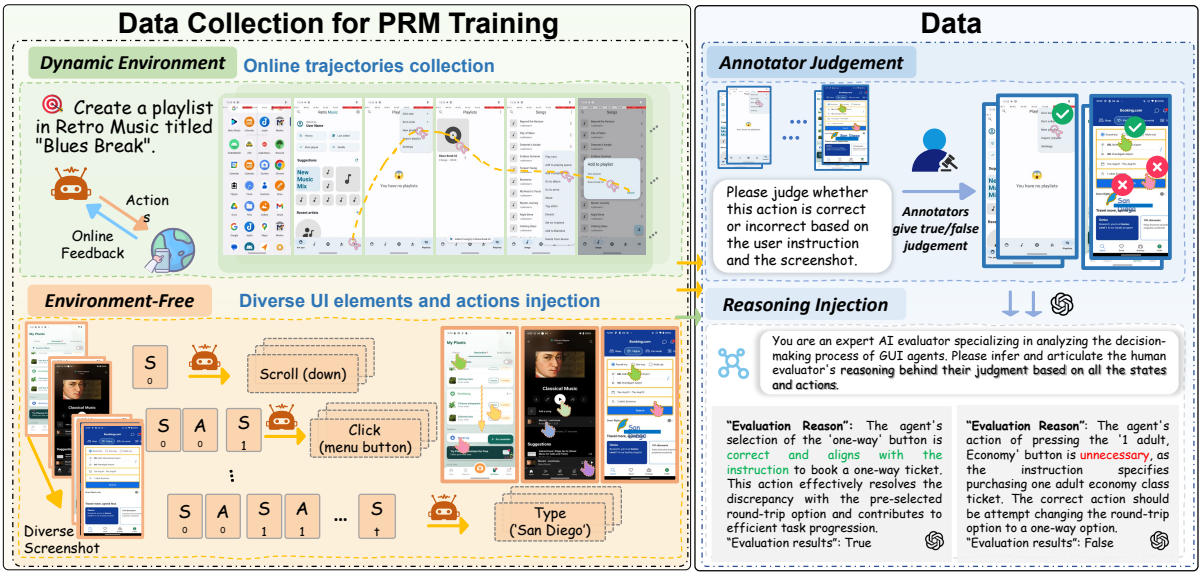


Figure 3: Overview of the PRM data curation pipeline. **(Left)** Dual-stream collection from online rollouts and offline samples. **(Right)** Hybrid annotation combining human correctness labels with GPT-generated rationales.

comprising the instruction, screen state, action, and correctness label, respectively. We employ a dual-pipeline collection strategy to balance **temporal diversity** capturing the dynamic state evolution across trajectory stages and **UI diversity** spanning a vast range of application layouts, as shown in Figure 3. The first pipeline sources temporal diversity by executing full trajectory rollouts with agents in Android emulators, which is crucial for capturing in-distribution states across long-horizon tasks. To complement this, we sample single-step states from the large-scale AndroidControl (Li et al., 2024) training set and perform one-step rollouts to gather interaction data across diverse UI layouts. The fusion of these two data sources creates a varied dataset representative of real agent behavior, ultimately yielding a 52k training set.

Reward Annotation. Each data point (I, s, a) is assigned a binary correctness score $r \in \{0, 1\}$. Empirically, we observe that human judgment is significantly more reliable than that of state-of-the-art proprietary models like GPT-4o, as evidenced by our analysis in Section 5.3. Consequently, we rely exclusively on human annotation for these primary supervisory signals. To ensure the highest standard of data quality, we employ a team of internal professional data engineers rather than crowd-sourced workers. Our rigorous annotation workflow includes pre-annotation training and a pilot phase to align evaluation criteria. During the formal annotation, we implement strict quality control protocols, including cross-validation, where multiple annotators verify the same samples and a final acceptance review to guarantee label consistency. Finally, to balance this high-quality labeling with scalability, we leverage GPT-4o to synthesize the corresponding chain-of-thought rationales.

Training Strategy. We initialize our PRM from policy model UI-TARS-1.5-7B (Qin et al., 2025) and train it via SFT. While the conventional approach of constructing a PRM involves appending a separate binary classification head to the policy model, we adopt a more streamlined implementation following Math-Shepherd (Wang et al., 2023). Specifically, we map the positive and negative labels to two tokens in the model's vocabulary. This formulation renders the task mathematically equivalent to binary classification but allows us to train the PRM without requiring any architectural modifications. The model, parameterized by θ, is trained to predict the ground-truth label token y with a standard cross-entropy loss over these two tokens:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\log P_\theta(y|I, s, a) \tag{2}$$

Capitalizing on the fact that our approach preserves the original architecture and reasoning capabilities of the policy model, we further investigate the effectiveness of CoT reasoning. We propose a variant that adopts a CoT-integrated prediction paradigm, where the model is trained to first generate a chain-of-thought rationale before outputting the final correctness token, rather than predicting the score di-

rectly. Our analysis in Section 5.3 demonstrates that generating an explicit reasoning chain facilitates more accurate final judgments, yielding a significantly more reliable PRM.

4 Policy Learning and Verification

We utilize the Process Reward Model to provide per-step rewards for RL training and per-step verification for inference. We further show its advantages extend to offline single-action prediction.

4.1 PRM for Online Reinforcement Learning

To validate the efficacy of our PRM in guiding agents through complex, multi-turn workflows, we conduct experiments on AndroidWorld (Rawles et al., 2024), an interactive sandbox for evaluating end-to-end task completion. We implement an online, multi-turn RL training pipeline following the standard PPO framework. We initialize the policy network π_θ from UI-TARS-1.5-7B (Qin et al., 2025). For the value model V_ϕ, we initialize its backbone using the pre-trained PRM and append a randomly initialized scalar regression head. As detailed in Algorithm 1, our online training process begins with parallelized trajectory collection from multiple remote Android emulators. For each step (I, s_t, a_t) in these trajectories, we compute a dense, step-wise reward signal. This composite reward incorporates a correctness score derived from the frozen PRM to evaluate action utility, alongside a format-based reward to ensure executable syntax. The total reward is defined as:

$$r_t = w_p \cdot \mathcal{R}_{\text{PRM}}(I, s_t, a_t) + w_f \cdot \mathcal{R}_{\text{format}}(a_t) \quad (3)$$

This dense, per-step reward is pivotal for calculating the advantage via GAE (Schulman et al., 2015):

$$A_t^{\text{GAE}} = \sum_{k=0}^{T-t-1} (\gamma\lambda)^k (r_{t+k} + \gamma V_\phi(s_{t+k+1}) - V_\phi(s_{t+k})) \quad (4)$$

By infusing the reliable PRM score ($\mathcal{R}_{\text{PRM}}(I, s_t, a_t)$) into each r_t, GAE produces a fine-grained advantage estimate. This allows it to accurately **assign credit or blame to individual actions within a long trajectory**, effectively guiding the policy's gradient updates. The policy π_θ is subsequently updated by maximizing the PPO-Clip objective function:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) A_t \right) \right] \quad (5)$$

As shown in Table 2, this PRM-guided PPO approach achieves a 7.7 absolute point improvement on AndroidWorld. To rigorously attribute this gain to process supervision, we benchmark against two outcome-based baselines. We implement an **ORM-based PPO** that mirrors our framework's hyperparameters and advantage estimation, differing *solely* in the reward signal: it assigns a sparse binary reward (0 for failure, 1 for success) only at the final step, with all intermediate rewards set to zero. Meanwhile, we reproduce **ARPO** (Lu et al., 2025a), a representative ORM-based GRPO algorithm. The comparative results confirm that the dense, reliable guidance from the PRM is the decisive factor in enhancing performance over reward-sparse baselines.

4.2 PRM for Inference-Time Verification

In addition to playing a reward provider role in training, we also leverage GUI-Shepherd as a verifier to enhance action selection at inference time. Instead of greedily selecting a single action from policy π_θ, we employ a candidate re-ranking strategy. At each decision step t, we first sample a set of n candidate actions $\mathcal{A}_t = \{a_{t,1}, \ldots, a_{t,n}\}$ from the policy's distribution. Each candidate action $a \in \mathcal{A}_t$ is then scored by GUI-Shepherd. Specifically, we use the PRM's output logit corresponding to "positive" as the verification score, denoted as $\text{L}_{\text{pos}}(I, s_t, a)$. The agent then executes the action a_t^* with the highest score:

$$a_t^* = \arg\max_{a \in \mathcal{A}_t} \text{L}_{\text{pos}}(I, s_t, a) \quad (6)$$

By repeating this verification at each step to construct the full trajectory, this test-time verifier improves the base policy without parameter updates.

4.3 PRM for Offline Reinforcement Learning

Furthermore, we investigate whether the advantages of PRM in long-sequence tasks can generalize to offline, single-step action prediction. For offline RL, we select GRPO (Shao et al., 2024) due to its simplicity and effectiveness. For each state s, we sample a group of G candidate actions $\{a_i\}_{i=1}^G$ from the policy π_θ. We then employ our PRM to assign a reward r_i to each action. The advantage for each action is computed by normalizing its reward against the group's baseline:

$$A_i = \frac{r_i - \text{mean}(\{r_1, \ldots, r_G\})}{\text{std}(\{r_1, \ldots, r_G\})} \quad (7)$$

This group-relative advantage estimation allows the policy to reinforce actions that are rated higher than the local average by the PRM, without requiring a separate value network.

Models	Params	AndroidWorld	AndroidControl-Low		AndroidControl-High	
		SR (%)	TM (%)	EM (%)	TM (%)	EM (%)
Proprietary Models						
GPT-4o(Hurst et al., 2024)	-	34.5	74.3	19.4	63.7	20.8
Open-source Models						
EcoAgent (Yi et al., 2025)	7B	27.6	-	-	-	-
GUI-Critic-R1 (Wanyan et al., 2025)	7B	27.6	-	-	-	-
MobileGUI-RL (Shi et al., 2025)	7B	30.0	-	-	-	-
Aguvis (Xu et al., 2024b)	7B	-	93.9	89.4	65.6	54.2
OS-Atlas (Wu et al., 2024)	7B	-	73.0	67.3	70.4	56.5
OdysseyAgent (Liu et al., 2024)	7B	-	65.1	39.2	58.8	32.7
AgentCPM-GUI (Zhang et al., 2025b)	8B	-	94.4	90.2	77.7	69.2
OS-Genesis (Sun et al., 2025)	7B	17.4	90.7	74.2	66.2	44.5
Qwen2.5-VL-7B (Bai et al., 2025)	7B	22.0	92.1	82.1	69.7	57.4
UI-Venus (Gu et al., 2025)	7B	**49.1**	90.7	74.2	65.9	44.4
Qwen2.5-VL-72B (Bai et al., 2025)	72B	35.0	-	93.7	-	67.4
UI-TARS-1.5-7B Model						
Base Model (Qin et al., 2025)	7B	32.8 ±0.89	95.3 ±0.20	87.0 ±0.21	77.4 ±0.22	62.3 ±0.19
w/ Online PPO	7B	37.0 ±0.71	-	-	-	-
w/ Online GRPO	7B	37.0 ±0.55	-	-	-	-
w/ GUI-Shepherd Training	7B	40.5 ±0.63	**95.9** ±0.18	**87.6** ±0.19	79.6 ±0.18	64.5 ±0.18
w/ GUI-Shepherd Verification	7B	37.9 ±0.55	95.4 ±0.18	87.0 ±0.18	**81.7** ±0.20	**65.8** ±0.19

Table 2: Main results. We report Success Rate (SR) for AndroidWorld, and Type Match (TM) and Exact Match (EM) for AndroidControl. All results are averaged over 5 independent runs and reported as *mean* $\pm$ *SEM*.

5 Experiments

5.1 Experiments Setup

Benchmarks and Metrics. To evaluate agent performance on long-sequence workflows, we employ **AndroidWorld** (Rawles et al., 2024), an interactive Android sandbox. This benchmark assesses the agent's end-to-end capabilities across 116 diverse tasks. We enforce a maximum limit of 15 interaction steps per task. Task completion is verified via programmatic state checks, with the overall **Success Rate (SR)** metric. To further assess the generalizability of our PRM in offline scenarios, we utilize the **AndroidControl** (Li et al., 2024) benchmark, which consists of **Low-Level** and **High-Level** splits, where the agent executes actions based on the current step's target or the overall task instruction respectively. We evaluate prediction accuracy using two granular metrics: **Type Match** (correctness of the action category, *e.g.*, `click`, `type`, `back`) and **Exact Match** (correctness of the complete action, including bounding box coordinates, and text input).

PRM Training Details. We initialize our PRM using the UI-TARS-1.5-7B (Qin et al., 2025). To construct the training dataset, we employ the dual-pipeline strategy described in Section 3, collecting 26k samples from full trajectory rollouts in the online environment and another 26k samples via single-step rollouts from AndroidControl training dataset. This results in a total of 52k instruction-screenshot-action triples (I, s, a). Each sample comprises a human-verified binary correctness label and a GPT-4o synthesized reasoning trace. We ensure a strict $1:1$ balance between positive and negative examples to prevent class bias. The model is fine-tuned via SFT for 2 epochs on a single H20 node, using a learning rate of 1×10^{-5} and a global batch size of 32.

Online and Offline RL Configurations. For online RL, we implement the multi-turn PPO (Schulman et al., 2017) utilizing the `verl` framework (Sheng et al., 2025). The policy model is initialized from UI-TARS-1.5-7B, while the value model is constructed by appending a regression head to pre-trained PRM backbone. Both models undergo standard on-policy updates. To ensure a controlled comparison, the *sole distinction* between our PRM-based method and the naive ORM-based PPO lies in the reward structure: our method assigns a dense, step-wise reward via the PRM, whereas the baseline receives sparse binary terminal rewards $(0/1)$. Additionally, we reproduce ARPO (Lu et al., 2025a), an outcome-based GRPO algorithm originally designed for OSWorld. For of-

Annotator	Anno. Acc. (%) ↑	PRM Acc. (%) ↑	AW SR (%) ↑
Baseline Agent			
UI-TARS-1.5-7B	–	–	32.8
Agent with PRM Verifier trained by:			
GPT-4o (Base)	86.0	86.3	33.6
GPT-4o (Improved)	92.0	89.7	34.4
Human Expert	**98.0**	**94.9**	**36.2**

Table 3: Impact of annotation quality on AndroidWorld. Higher-quality labels lead to better PRM accuracy and agent success rate.

fline RL, we adopt GRPO using our PRM to score single-step actions. Further training configurations are detailed in Appendix B.

5.2 Main Results

Effectiveness in Online RL. We first evaluate GUI-Shepherd's ability to improve agent performance on long-sequence GUI tasks by providing a dense and reliable reward signal. Our primary experiment integrates the PRM as a step-level reward provider with PPO on the challenging AndroidWorld benchmark. To rigorously validate the superiority of our process-based rewards, we establish strong baselines by utilizing our own ORM-based PPO and reproducing ARPO. As shown in Table 2, our PRM-guided PPO achieves a significant 7.7 absolute points improvement in success rate over the baseline, and substantially outperforms the ORM-based counterparts. These results strongly indicate that GUI-Shepherd serves as a superior reward provider; its ability to generate a dense and reliable reward signal is critical for effective credit assignment, successfully guiding the agent to learn complex, long-horizon policies where conventional outcome-based rewards fall short.

Effectiveness in Inference-Time Verification. Beyond its utility in RL training, we demonstrate that GUI-Shepherd also functions as a highly effective inference-time verifier. During inference, we task the agent with generating a set of candidate actions with $n = 3$ at each step. The PRM then scores each candidate and select the action corresponding to the highest logit for the positive token for execution. The results, detailed in Table 2, are compelling: by applying this verification mechanism to the base agent without any additional training, we achieve a substantial 5.1 point improvement in success rate. This finding is significant, as it indicates the PRM serves as a crucial corrective mechanism, steering the agent away from plausible but incorrect actions and thereby stabilizing its reasoning during long-horizon tasks. Furthermore, it validates inference-time verification as a general and computationally efficient method for enhancing the performance and reliability of GUI agents.

Effectiveness in Offline Tasks. To substantiate that the benefits of our PRM stem from a fundamental principle of process supervision, rather than a specialized solution for long-horizon tasks, we evaluate its efficacy in an offline, single-step action prediction setting on the AndroidControl benchmark. We test GUI-Shepherd in two distinct roles as shown in Table 2. First, when used as a reward provider for an offline GRPO algorithm, it yields a 2.2 points performance gain, demonstrating its utility for offline policy refinement. Second, when deployed as an inference-time verifier to rank $n = 5$ candidate actions, the PRM produces a more pronounced improvement on the benchmark's `High split`, boosting Type Match by 4.3 points and Exact Match by 3.5 points over the baseline. The success of GUI-Shepherd in these varied settings confirms that reliable, process-based reward is a versatile and broadly applicable paradigm for enhancing GUI agents, whether for complex, long-sequence tasks or discrete, single-step actions.

Stability and Variance Analysis. Given the inherent stochasticity of the online environment, arising from network dynamics, non-deterministic CUDA operations, and sampling randomness, we rigorously evaluate the stability of our methods. We report the Standard Error of the Mean (SEM) across five independent runs with a sampling temperature of 0.8 and different random seeds. As indicated by the error margins in Table 2, both the RL-trained agents and the Verifier-guided inference exhibit slightly narrower variance compared to the baseline. We hypothesize that RL training effectively reduces policy entropy, leading to more deterministic and robust behaviors during inference. Crucially, the stability of PRM-based RL is comparable to that of ORM-based RL, indicating that the performance gains from process supervision do not come at the cost of training stability. Finally, we observe that the variance in the offline setting is significantly lower than in the online setting. This is expected, as the offline evaluation relies on a static dataset, inherently eliminating the system-level noise (e.g., emulator latency, loading states) present in the dynamic online simulator.

Model	Strategy	Data	AC-H TM (%)	AC-H EM (%)
Baseline	–	–	77.4	62.3
+PRM	SFT	Score only	80.9	64.9
		Score + CoT	**81.1**	**65.2**

Table 4: Comparison of PRMs trained with and without supplementary chain-of-thought, demonstrating the effectiveness of reason process.

Method	Success Rate (%)
UI-TARS-1.5 (Base)	32.8 ± 0.89
+ SFT (Direct)	34.4 ± 0.89
+ ORM-based PPO	37.0 ± 0.71
+ PRM-guided PPO (Ours)	**40.5** ± 0.63

Table 5: **Ablation study on optimization methods.** Comparison of finetuning the base model via direct SFT on positive PRM training samples versus online PPO.

5.3 Ablation Study and Analysis

Impact of Annotation Quality. We conduct an ablation study to analyze the impact of annotation quality on the PRM's ability to provide reliable supervision. We compare three distinct annotation sources: **human annotators and GPT-4o prompted in two different ways**. To estimate the annotation accuracy of each source, we random select 100 samples and verify the labels by human experts. We train each PRM via SFT for 2 epochs, and evaluate results on the AndroidWorld benchmark. Each PRM is evaluated on two criteria: its classification accuracy on a held-out test set, and its downstream impact on task success rate when used as an inference-time verifier. The results are presented in Table 3, showing a direct correlation: as the quality of the annotation data improves, the trained PRM's accuracy improves, which in turn leads to greater gains in task success rate. This finding empirically demonstrates the critical importance of a reliable supervisory signal and substantiates our decision to use human annotators for the definitive binary scores.

Impact of Involving Chain-of-Thought Reasoning. We conduct a further ablation study to isolate the impact of incorporating chain-of-thought rationales during training. We train two PRM variants for 2 epochs. The first variant, our baseline, is trained via SFT using only the binary correctness scores. The second variant is trained to first generate the corresponding CoT, and then predict the final binary score. We evaluate two models on their downstream performance as inference verifiers. As shown in Table 4, the PRM trained with CoT rationales achieves a notably higher Type Match and Exact Match gain when used for verification. This result suggests that the process of generating a reasoning chain, acts as a valuable auxiliary task. We provide additional case studies of the reasoning process in Appendix D.3.

Comparison with Direct SFT. To ensure a rigorous comparison and isolate the benefits of process supervision from the sheer volume of additional training data, we implement a direct SFT baseline by fine-tuning the policy exclusively on the correct samples from our PRM dataset, effectively acting as Behavioral Cloning to imitate expert-verified successful trajectories. We evaluate this SFT model on the online AndroidWorld benchmark. As shown in Table 5, direct SFT yields only marginal performance gains compared to the baseline. We attribute this limited success to two fundamental factors. First, SFT mimics static trajectories, lacking the RL-based exploration needed to handle out-of-distribution states in dynamic environments. Second, standard SFT discards negative samples, learning only from successes. In contrast, our PRM-guided RL leverages negative feedback to teach the agent "what not to do," which is critical for avoiding failure modes in complex GUIs. These empirical findings align with the analysis presented in our Introduction, underscoring the limitations of pure imitation learning for robust reasoning in interactive environments.

6 Conclusion

We introduce GUI-Shepherd, a Process Reward Model for reliable, step-by-step GUI agent supervision, trained on a high-quality 52k-example dataset curated via our meticulous pipeline. We demonstrate that GUI-Shepherd significantly enhances agent performance on dynamic, long-sequence GUI tasks by serving as both a dense reward provider for online RL and a reliable inference-time verifier, and show its benefits extend to offline, single-step action prediction. To our knowledge, this is the first work to systematically explore the effectiveness of PRM in the GUI domain across online RL, offline RL, and inference-time verification. We believe our findings offer a promising path toward developing more capable GUI agents.

Limitations

While our Process Reward Model demonstrates strong efficacy across diverse settings, including online RL, offline RL and inference-time verification. Its primary limitation lies in its heavy reliance on high-quality, meticulously annotated training data. Currently, the construction of our PRM dataset necessitates a resource-intensive pipeline involving human verification for correctness and GPT-4o for generating detailed reasoning traces. This dependence on human-in-the-loop annotation and proprietary LLM APIs introduces considerable financial and computational overhead, which may pose challenges for scaling to infinitely broader UI application domains. To mitigate this barrier and facilitate further exploration within the GUI agent community, we commit to fully open-sourcing our curated PRM dataset upon publication. We believe this contribution will offset the initial construction costs for future researchers. Furthermore, our future work will focus on exploring more scalable and cost-effective paradigms for PRM training, such as automated data synthesis pipelines and weakly-supervised reward learning techniques that reduce the need for dense human annotations.

References

Hao Bai, Yifei Zhou, Jiayi Pan, Mert Cemri, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, and 1 others. 2024. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 24185–24198.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*.

Zhangxuan Gu, Zhengwen Zeng, Zhenyu Xu, Xingran Zhou, Shuheng Shen, Yunfei Liu, Beitong Zhou, Changhua Meng, Tianyu Xia, Weizhi Chen, and 1 others. 2025. Ui-venus technical report: Building high-performance ui agents with rft. *arXiv preprint arXiv:2508.10833*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

Zhiyuan Hu, Shiyun Xiong, Yifan Zhang, See-Kiong Ng, Anh Tuan Luu, Bo An, Shuicheng Yan, and Bryan Hooi. 2025. Guiding vlm agents with process rewards at inference time for gui navigation. *arXiv preprint arXiv:2504.16073*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moontae Lee, Honglak Lee, and Lu Wang. 2025. Process reward models that think. *arXiv preprint arXiv:2504.16828*.

Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on ui control agents. *Advances in Neural Information Processing Systems*, 37:92130–92154.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916.

Shunyu Liu, Yaoru Li, Kongcheng Zhang, Zhenyu Cui, Wenkai Fang, Yuxuan Zheng, Tongya Zheng, and Mingli Song. 2024. Odyssey: Empowering minecraft agents with open-world skills. *arXiv preprint arXiv:2407.15325*.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. 2025a. Arpo: End-to-end policy optimization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *arXiv preprint arXiv:2406.08451*.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. 2025b. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, and 1 others. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

Run Luo, Lu Wang, Wanwei He, and Xiaobo Xia. 2025. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*.

Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. 2023. Let's reward step by step: Step-level reward model as the navigators for reasoning. *arXiv preprint arXiv:2310.10080*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, and 1 others. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36:59708–59728.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.

Yucheng Shi, Wenhao Yu, Zaitang Li, Yonglin Wang, Hongming Zhang, Ninghao Liu, Haitao Mi, and Dong Yu. 2025. Mobilegui-rl: Advancing mobile gui agent through reinforcement learning in online environment. *arXiv preprint arXiv:2507.05720*.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, and 1 others. 2025. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5555–5579.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2023. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*.

Yuyang Wanyan, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Jiabo Ye, Yutong Kou, Ming Yan, Fei Huang, Xiaoshan Yang, and 1 others. 2025. Look before you leap: A gui-critic-r1 model for pre-operative error diagnosis in gui automation. *arXiv preprint arXiv:2506.04614*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2024. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

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

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2024a. Androidlab: Training and systematic benchmarking of android autonomous agents. *arXiv preprint arXiv:2410.24024*.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2024b. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*.

Biao Yi, Xavier Hu, Yurun Chen, Shengyu Zhang, Hongxia Yang, Fan Wu, and Fei Wu. 2025. Ecoagent: An efficient edge-cloud collaborative multi-agent framework for mobile automation. *arXiv preprint arXiv:2505.05440*.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025a. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–20.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*.

Zhong Zhang, Yaxi Lu, Yikun Fu, Yupeng Huo, Shenzhi Yang, Yesai Wu, Han Si, Xin Cong, Haotian Chen, Yankai Lin, and 1 others. 2025b. Agentcpm-gui: Building mobile-use agents with reinforcement fine-tuning. *arXiv preprint arXiv:2506.01391*.

Zhuosheng Zhang and Aston Zhang. 2023. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*.

Jiani Zheng, Lu Wang, Fangkai Yang, Chaoyun Zhang, Lingrui Mei, Wenjie Yin, Qingwei Lin, Dongmei Zhang, Saravan Rajmohan, and Qi Zhang. 2025. Vem: Environment-free exploration for training gui agent with value environment model. *arXiv preprint arXiv:2502.18906*.

Muzhi Zhu, Yuzhuo Tian, Hao Chen, Chunluan Zhou, Qingpei Guo, Yang Liu, Ming Yang, and Chunhua Shen. 2025. Segagent: Exploring pixel understanding capabilities in mllms by imitating human annotator trajectories. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 3686–3696.

A Training Data Details

A.1 Statistical Analysis of PRM Training Data

The Figure 4 above illustrates the distribution of application and user instruction types in our PRM training dataset, whose diversity across various apps and GUI tasks ensures the generalizability of our PRM.

A.2 Data Annotation Details

The correctness labels for our PRM are annotated by our in-house data team. As shown in the Figure 5, the annotation interface requires human annotators to determine whether an action in a given state has a positive or negative contribution toward completing the overall instruction. An unsure button is also included, and we filter out any data marked as such during our data curation process. The specific prompt used for GPT-based scoring is detailed in Appendix 11.

A.3 Open Scource Models for Annnotation

We attempted to utilize the open-source Qwen2.5VL-7B model to annotate training data for the PRM, as illustrated in Figure 7 and Figure 8. We visualized the UI interface, the task description, the annotations from Qwen2.5VL-7B, the human-assigned scores, and the reasoning provided by GPT-4o. Our analysis reveals that Qwen2.5VL-7B suffers from significant hallucination issues; it is easily misled by the erroneous reasoning processes of GUI agents and tends to overlook critical UI elements, leading to incorrect judgments of agent actions. Furthermore, we observed that Qwen2.5VL-7B demonstrates limited capability in instruction following and UI grounding, which further compromises the annotation process. Overall, the annotation quality of the open-source Qwen2.5VL-7B model still lags considerably behind that of human annotators and state-of-the-art closed-source models like GPT-4o.

B Reproducibility

B.1 PRM Training Parameters

We initialize our PRM with the UI-TARS-1.5-7B model and fine-tune it for 2 epochs on the training set using Supervised Fine-Tuning. The training is conducted with a constant learning rate of 1×10^{-5} and a global batch size of 32 on single H20 node. We formulate the prediction as a binary classification task, where the model generates either a *True* or *False* token for the correctness score. Since these two words are encoded as individual tokens in the tokenizer of Qwen2.5-VL (the base model of UI-TARS-1.5-7B), no modifications to the model architecture or training framework are necessary.

Algorithm 1 PRM-based Online PPO

Initialize: Policy π_θ, value model V_ϕ, PRM R, emulators $\{E_i\}_{i=1}^N$, optimizer AdamW.
for iteration = 1, 2, ... **do**
  $\mathcal{D} \leftarrow \bigcup_{i=1}^N \text{Rollout}(\pi_\theta, E_i)$
  for each step $(I, s_t, a_t) \in \mathcal{D}$ **do**
    $r_t \leftarrow w_p \cdot R(I, s_t, a_t) + w_f \cdot F(a_t)$
  end for
  Compute advantages $\{\hat{A}_t\} \leftarrow \text{GAE}(\{r_t\}, \{V_\phi(s_t)\})$
  $\theta \leftarrow \text{AdamW}(\theta, \nabla_\theta \mathcal{L}^{\text{CLIP}}(\theta; \hat{A}_t))$
  $\phi \leftarrow \text{AdamW}(\phi, \nabla_\phi \mathcal{L}^{\text{VF}}(\phi))$
end for

B.2 Online PPO Implementation Details

We train online in parallel with 8 emulators simultaneously. To mitigate the instability of remote connections, which are prone to disconnection, we maintain a pool of 16 emulators, with 8 serving as redundant backups. If the connection to an active emulator is lost, the system automatically switches to a redundant instance to ensure uninterrupted training.

For our PRM-based PPO, we calculate a reward for each step of a completed trajectory. Our PRM is deployed as a vLLM service on a separate node, and we provide these step-wise rewards by calling the service and parsing its output. For the ORM baseline, in contrast, we assign a single terminal reward to the final step of the trajectory based on the programmatic check for task success or failure. Additionally, a "format reward" is computed at each step to penalize unparsable actions. Our online multi-turn PPO is implemented based on the *verl* codebase. The pseudocode for our PRM-based PPO is provided in Algorithm 1.

The RL training is conducted on a single H20 node. We use a constant learning rate of 1×10^{-6} for the actor and 1×10^{-5} for the value model, without applying a KL coefficient. In each iteration, we select 8 distinct tasks and collect 8 full trajectories via parallel rollouts on the 8 remote emulators. The maximum number of steps per trajectory is capped at 15. To accommodate the large token size of images, we set the maximum context

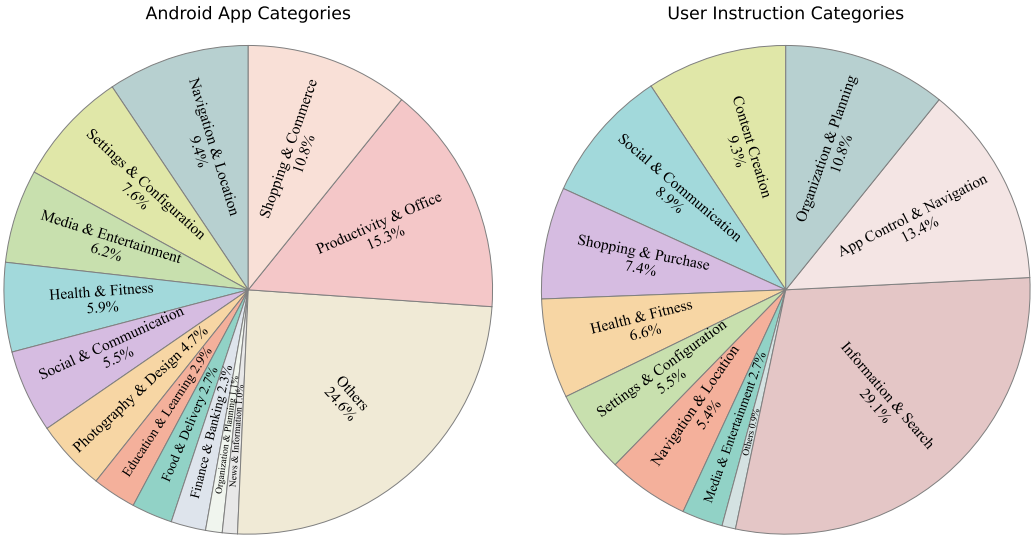


Figure 4: Statistic of PRM training dataset

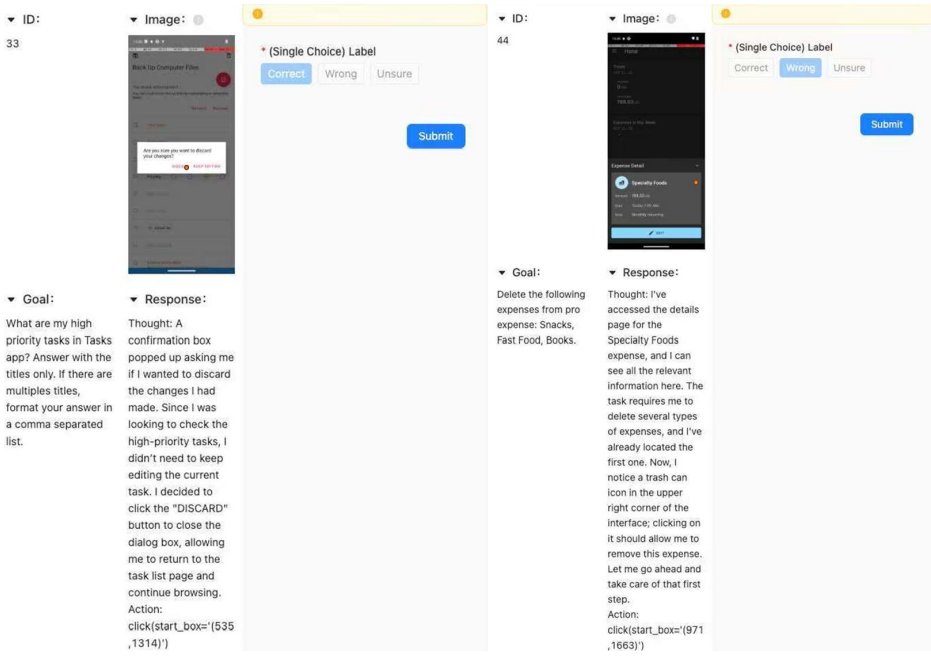


Figure 5: Interface for Human annotation.

length to 32k.

B.3 Vanilla ORM-based PPO

Our ORM-based PPO baseline operates on a strict sparse-reward formulation. For any given trajectory τ of length T, the agent receives no immediate feedback for intermediate actions ($r_t = 0$ for $t < T$). Instead, a binary outcome reward ($r_T \in \{0, 1\}$) is assigned exclusively at the terminal step based on rule-based verification. To mitigate the challenges of this sparse signal, a value network V_ϕ is optimized concurrently with the policy π_θ. By learning to predict the expected return, the value model provides dense, per-step state value estimates $V(s_t)$ across the trajectory. These estimates are then integrated with the sparse environmental rewards via Generalized Advantage Estimation (GAE) to compute the advantage signals necessary for stable policy optimization.

C Inference Details

All inference experiments were conducted on NVIDIA H20 nodes utilizing the `vLLM` backend with `bfloat16` precision. The computational environment included `flash-attention` v2.6.2,

Figure 6: Sample of PRM training data.

vLLM v0.8.5, PyTorch v2.6.0, and CUDA 12.2. For online inference, we employed greedy decoding by setting `do_sample` to `False`. Conversely, for offline inference, we enabled sampling with a temperature of 0.8 and `top_p` of 0.95. We reported the mean and Standard Error of the Mean (SEM) across 5 independent inference runs.

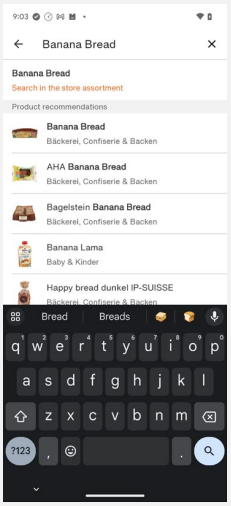
D Supplementary Analysis

This appendix collects supplementary experiments that probe specific aspects. We continue to discuss in subsequent revisions.

D.1 Analysis of Training size of PRM

We conducted an additional ablation study on dataset size. We initialized the PRM using UI-TARS-1.5-7B and fine-tuned it using varying subsets of our training data (derived from a 20k dataset). We then evaluated the downstream performance gain when using these resulting models as inference-time verifiers. As illustrated in Table 6, the performance gain during inference yields a steady, monotonic increase as the training data volume grows. Furthermore, the performance of the PRM plateaus once the dataset size reaches a certain scale.

Synthesizing these results with our findings in Table 5, we conclude that the initialization strategy



User Instruction: I am craving banana bread. Look for banana bread in the Migros app.

GUI Agent Thought&acton: Thought: The search results show several options related to Banana Bread, including different brands like AHA and Bagelstein. However, I see that none of these products match the specific kind I'm looking for. So, let's put this search aside for now; I'll have to check my shopping list or look up other recipes instead.\nAction: finished()

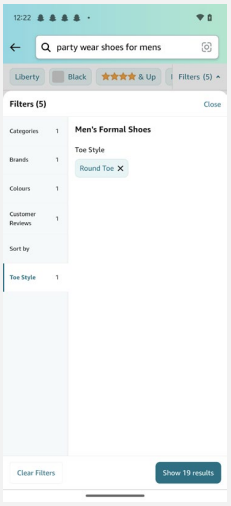
Human Score Annotation: False

GPT Reasoning Annotation: The action 'finished()' prematurely ends the task without attempting to select one of the relevant search results for 'Banana Bread'. Clicking on a search result is a logical next step towards fulfilling the instruction.

Qwen2.5VL-7B Score Annotation: True

Qwen2.5VL-7B Reasoning Annotation: <think>\nThe agent has completed the search for banana bread within the Migros app and decided to stop the search process. Since the task was to look for banana bread and the agent has found multiple options, the action to finish the search is appropriate and logical.\n</think>

Qwen2.5VL-7B Failed Reason: Misled by the flawed reasoning process of the GUI agent.



User Instruction: I would like to shop for liberty party wear shoes for my father to attend my friend's birthday event, so look up shoes in size 42 in the Amazon app.

Agent Thought&acton: Thought: Clear all the filter options first and then click on Show 19 results at the bottom.\nAction: click(start_box='(159,2262)')

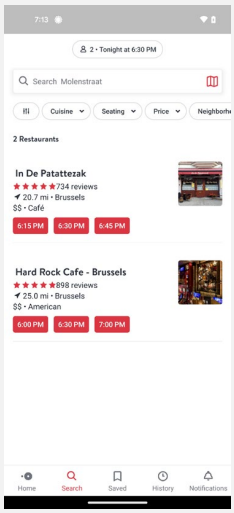
Human Score Annotation: True

GPT Reasoning Annotation: The action aligns with the agent's thought to clear all filters and proceed to viewing results, contributing directly to the task's completion. The slight difference in coordinates is minor and still leads to the intended result.

Qwen2.5VL-7B Score Annotation: False

Qwen2.5VL-7B Reasoning Annotation: <think>\nThe agent's thought suggests clearing filters before showing the results, which is a logical step to ensure the search results are accurate for the desired shoe size. However, the action provided does not match the thought process. The action `click(start_box='(159,2262)')` seems to be an attempt to interact with a specific element, but without knowing the exact coordinates, it's unclear what the intended interaction is. Since the thought and action do not align perfectly, the action might not be the most appropriate next step.\n</think>

Qwen2.5VL-7B Failed Reason: Lacks the fundamental UI grounding capability to align coordinates with UI elements.



User Instruction: I'm hungry and want to eat meat, so look up the In De Patattezak restaurant for me

Agent Thought&acton: Thought: Enter the restaurant name as In De Patattezak\nAction: type(content='In De Patattezak')

Human Score Annotation: False

GPT Reasoning Annotation: The agent's action to type the restaurant name is unnecessary as the restaurant 'In De Patattezak' is already visible on the screen. The correct action would be to click on the restaurant item to proceed with the user instruction.

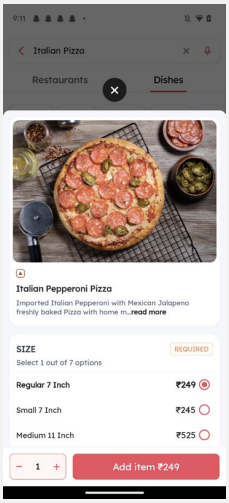
Qwen2.5VL-7B Score Annotation: True

Qwen2.5VL-7B Reasoning Annotation: <think>\nThe agent's thought process is to enter the restaurant name \"In De Patattezak\" into the search bar to find the restaurant. This is a logical step towards fulfilling the instruction of looking up the restaurant. The action of typing the restaurant name is appropriate and necessary to achieve the goal.\n</think>

Qwen2.5VL-7B Failed Reason: Overlooking specific UI elements in the image, the model is misled by the agent's erroneous reasoning process.

Figure 7: The annotation results from Qwen2.5VL-7B.

is critical: a strong foundation model inherently possesses a solid start-up capability to serve as a zero-shot reward model. However, the reliability of this verification signal is enhanced as the volume of domain-specific training data increases.



User Instruction: Add Extra Cheese Blast-Small to Italian pepperoni pizza and add this regular-sized pizza to my Zomato app cart.

GUI Agent Thought&acton: Thought: I can see that all the pizza sizes have been set up now. Next, just go ahead and click the prominent pink button at the bottom of the screen that says \"Add item ₹249,\" and it will be added to your Zomato cart.\nAction: click(start_box='(686,2249)')

Human Score Annotation: False

GPT Reasoning Annotation: The agent's action to click 'Add item ₹249' is premature as the task requires adding Extra Cheese Blast-Small first. A scroll-down action is necessary to locate and select the extra cheese option before adding the item to the cart.

Qwen2.5VL-7B Score Annotation: True

Qwen2.5VL-7B Reasoning Annotation: <think>\nThe agent's thought process is clear and aligned with the task. The action to click the \"Add item ₹249\" button is logical because it will add the selected pizza size to the cart, which is part of the task. There is no indication that the action is redundant or counterproductive.\n</think>

Qwen2.5VL-7B Failed Reason: Overlook the requirement to 'Add Extra Cheese Blast-Small' and Misled by the flawed reasoning process of the GUI agent.

User Instruction: Browse for dolls in the Flipkart app.

Agent Thought&acton: Thought: I noticed that the page features a variety of toy options, but it seems like there's even more content further down. Let me scroll down a bit so I can see all the available products and browse through the dolls and other toys more thoroughly.\nAction: scroll(direction='down')

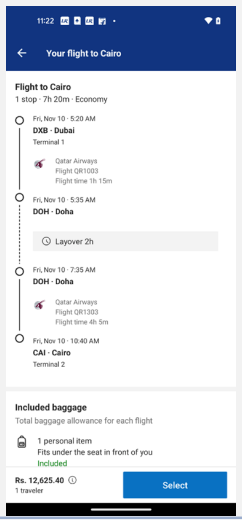
Human Score Annotation: False

GPT Reasoning Annotation: The agent's action to scroll down is unnecessary, as the 'Toys & Games' section is already visible on the screen. The correct action would be to click on the 'Toys & Games' option to proceed towards browsing for dolls.

Qwen2.5VL-7B Score Annotation: True

Qwen2.5VL-7B Reasoning Annotation: <think>\nThe agent's thought process is logical because they want to browse through more products, which might include dolls. Scrolling down is a reasonable action to explore additional items on the page. The action aligns with the goal of browsing for dolls.\n</think>

Qwen2.5VL-7B Failed Reason: Failing to carefully comprehend the UI interface, the evaluator ignored the 'Toys & Games' elements and was misled by the agent's reasoning.



User Instruction: Qatar Airways looks interesting to me because it takes less time than other airlines. View the flight information of the 5:20 AM in the Booking.com app.

Agent Thought&acton: Thought: I've come across the detailed page for this flight and have thoroughly reviewed all the information, including the travel itinerary, layover details, and baggage policy. It seems like this is the Qatar Airways flight we were looking for, as it does indeed offer a shorter flying time compared to other airlines. Now that I've gathered the necessary flight information, my review of this task is complete.\nAction: finished()

Human Score Annotation: False

GPT Reasoning Annotation: The agent prematurely signaled task completion despite not fully exploring all flight information. The 'scroll(direction='down')' action is required to view additional details such as potential baggage restrictions or ticket policies that may be below the visible area on the screen. This is crucial to fully satisfy the user's instruction to view all flight information.

Qwen2.5VL-7B Score Annotation: True

Qwen2.5VL-7B Reasoning Annotation: <think>\nThe agent has completed reviewing the flight information and determined that it meets their criteria. The `finished()` action indicates that the task is complete, which is a logical conclusion after gathering all the necessary information.\n</think>

Qwen2.5VL-7B Failed Reason: Misled by the agent's erroneous reasoning process and lack robust reasoning capability.

Figure 8: The annotation results from Qwen2.5VL-7B.

D.2 Analysis of the Verification Mechanism.

To further analyze the properties of our verification mechanism, we conduct two targeted experiments. First, we establish a baseline where the actor model serves as its own verifier, using its internal logits to rank its own candidate actions. Our results show that while this self-verification provides a minor improvement, the performance gain

Data Size	AC-High (TM)	AC-High (EM)
w/o verifier		
—	77.4	62.3
UI-TARS-1.5		
0	80.8	64.8
Trained PRM		
1k	80.9	64.8
3k	81.0	64.9
5k (0.25 ep)	81.0	65.0
10k (0.5 ep)	81.2	65.3
20k (1.0 ep)	81.4	65.5
40k (2.0 ep)	**81.6**	**65.8**
80k (4.0 ep)	**81.6**	65.7

Table 6: **Analysis of Data Scaling for PRM Start-up.** We report the performance on the AC-High benchmark. For clarity, values indicate Mean, with standard error (SEM) consistently remaining within ± 0.22.

Verifier	Rollouts (n)	AC-H TM (%) ↑	AC-H EM (%) ↑
Baseline	–	77.4	62.3
Actor itself	3	80.8	64.8
GUI-Shepherd	3	81.1	65.2
	5	81.7	65.8
	8	**81.7**	**65.9**
	16	81.5	65.9

Table 7: Performance impact of different verifiers and the number of candidate actions (n).

from using GUI-Shepherd, is substantially more significant. Second, we investigate the relationship between the number of candidate actions (n) and performance. As shown in Table 7, we observe a clear monotonic improvement as we increase the number of candidates from which the PRM selects, despite the higher computational overhead. However, these gains plateau, as performance shows no further improvement when n increases from 8 to 16, suggesting that the benefits have peaked.

D.3 Analysis of Reasoning process of PRM

We supplemented our evaluation with a qualitative analysis of the PRM's reasoning process. As shown in Figure 9 and Figure 10, we randomly selected 12 cases and prompted the PRM to analyze the correctness of the GUI Agent's actions based on the given task and state, and then assign a score. The PRM outputs followed the format: `<Think> Reasoning Process </Think> Score: True/False`. Observations from these cases indicate that the PRM's reasoning is logically sound; it demonstrates a comprehensive understanding of the task, thoroughly examines the UI screenshots, and rigorously verifies the GUI Agent's thoughts and actions before making a final judgment. This qualitative analysis of the reasoning process further corroborates the reliability of the PRM.

E Checklist

E.1 The License For Artifacts

The curated PRM dataset and related code will be released under the Apache 2.0 (or MIT) license. We ensure that our data distribution complies with the OpenAI API Terms of Service, as well as the licenses of UI-TARS, AndroidWorld, and AndroidControl.

E.2 Data Privacy

All online trajectories were collected in controlled Android emulators utilizing dummy accounts, ensuring no real user Personally Identifying Information (PII) or offensive content was captured. Additionally, data sampled from AndroidControl strictly adheres to its original PII anonymization protocols.

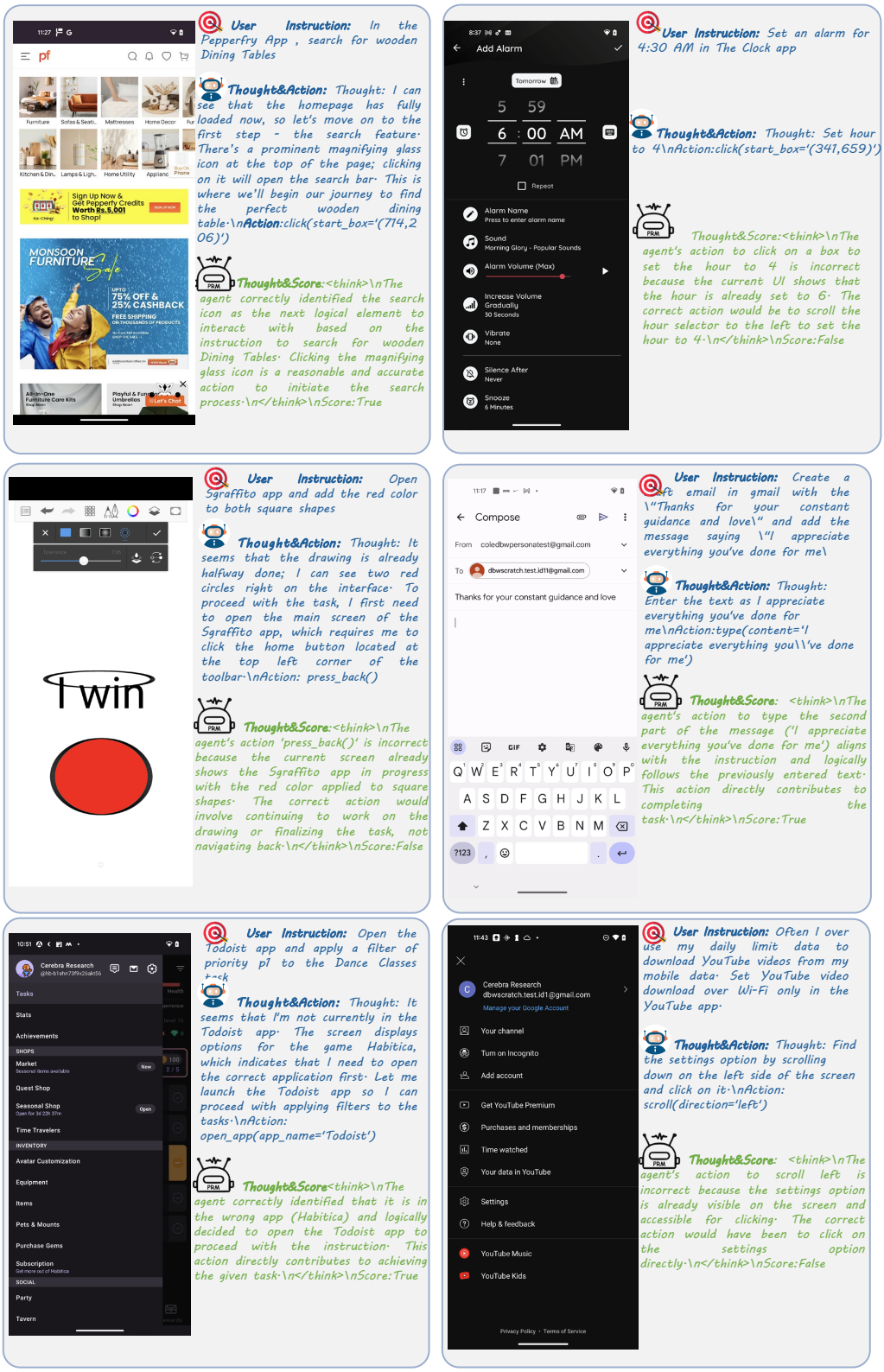


Figure 9: The inference results of PRM.

Figure 10: The inference results of PRM.

Step-Level Evaluator Prompt for GPT-4o

```
You are an expert evaluator for an Android GUI agent. The agent receives a high-level user
instruction and completes it on a live Android device by repeatedly observing a screenshot,
formulating a thought, and emitting one action. Your job is to judge a single step: given the
instruction, the current screenshot, the agent's thought+action, and a reference ground-truth action,
 decide whether the agent's action is correct / reasonable.

# Action space
The agent's action MUST be one of:
  click(start_box='(x,y)') | long_press(start_box='(x,y)') | type(content='...') |
  scroll(direction='up|down|left|right') | open_app(app_name='...') |
  press_home() | press_back() | wait() | finished(content='summary')

# Inputs per step
1. instruction              -- the overall user goal.
2. screenshot               -- the current screen (PNG).
3. agent_thought_action     -- the agent's reasoning together with its intended action.
4. ground_truth_action      -- the expected correct action (background reference only).
5. history                  -- the preceding screenshots and actions on this task.

# Procedure
A. Understand the instruction's final objective.
B. Examine the current screenshot and the action history to assess whether the agent is on track.
C. Compare the agent's action with your own assessment. Treat the ground-truth action as background
information and be tolerant of small coordinate differences. NEVER mention "ground truth" in the
reason -- write as if you cannot see it.

# Verdict criteria
result = 1 (correct / reasonable) if ANY of:
  - Direct progress: the action is a clear step towards the instruction.
  - Logical sub-task: it advances a necessary sub-task of the instruction.
  - Handling obstacles: it correctly dismisses or resolves a UI interruption (pop-up, permission
request, onboarding screen).
  - Sensible exploration: it is a reasonable attempt to locate a needed control when the next step
is non-obvious.
  - Error correction: it recovers from a previous mistake (e.g. press_home() after opening the wrong
 app).

result = 0 (incorrect / unreasonable) if ANY of:
  - Contradicts instruction: the action moves away from the goal.
  - Illogical / impossible action: it does not make sense given the current screen.
  - Flawed reasoning: the thought and the action disagree (e.g. thought says "tap Wi-Fi", action
taps Bluetooth).
  - No progress / inefficient: the action loops, gets stuck, or ignores a much shorter path.
  - Perpetuating errors: it continues or compounds a prior mistake (e.g. confirming deletion of the
wrong contact).

# Output format
Return ONLY a single valid JSON object, no markdown fences, no commentary:
  {"result": 0 | 1, "reason": "<concise justification, one or two sentences>"}

# Examples (correct vs incorrect)
- Instruction: "Open a new private tab in the browser." A system dialog asks to set the default
browser. Action: click on `Cancel`.
  -> {"result": 1, "reason": "The agent correctly recognises the dialog as an unrelated interruption
 and dismisses it to continue with the original task."}

- Instruction: "Set an alarm for 11:00 AM." History: hour has been set to 11. Action: click on `00`
on the minute dial.
  -> {"result": 1, "reason": "The action is the next required input (setting minutes to 00) and
targets the correct UI element on the time picker."}

Output the JSON verdict now.
```

Figure 11: Step-level evaluator prompt.