Don't Act Blindly: Robust GUI Automation via Action-Effect Verification and Self-Correction

Yuzhe Zhang[1,2*] **Xianwei Xue**[2] **Xingyong Wu**[2] **Mengke Chen**[2]
Chen Liu[2] **Xinran He**[2] **Run Shao**[2] **Feiran Liu**[1]
Huanmin Xu[2] **Qiutong Pan**[2] **Haiwei Wang**[2†]

[1]Beijing University of Technology, Beijing, China [2]Baidu Inc., Beijing, China
{zhangyuzhe02, xuexianwei, wuxingyong, chenmengke, liuchen26}@baidu.com
{hexinran, shaorun, xuhuanmin, panqiutong, wanghaiwei}@baidu.com

Abstract

Autonomous GUI Agents based on vision-language models (VLMs) often assume deterministic environment responses, generating actions without verifying whether previous operations succeeded. In real-world settings with network latency, rendering delays, and system interruptions, this assumption leads to undetected action failures, repetitive ineffective behaviors, and catastrophic error accumulation. Moreover, learning robust recovery strategies is challenging due to the high cost of online interaction and the lack of real-time feedback in offline datasets. We propose VeriGUI (**Veri**fication-driven **GUI** Agent), which explicitly models action outcomes and recovery under noisy environments. VeriGUI introduces a **T**hinking–**V**erification–**A**ction–**E**xpectation (**TVAE**) framework to detect failures and guide corrective reasoning, and a two-stage training pipeline that combines **Robust SFT** with synthetic failure trajectories and **GRPO** with asymmetric verification rewards. We train VeriGUI in both 3B and 7B configurations. We further construct a **Robustness Benchmark** based on AndroidControl-High to evaluate failure recognition and correction. Experiments demonstrate that VeriGUI-3B and -7B achieve strong results across offline and online benchmarks, with the verification-and-recovery mechanism transferring effectively to dynamic real-world environments.

1 Introduction

GUI automation has emerged as a promising application of vision-language models (VLMs) (Hong et al., 2024; Cheng et al., 2024; Lin et al., 2024), enabling autonomous agents to interpret screenshots, understand natural language instructions, and execute complex multi-step tasks on mobile devices (Zhang et al., 2025a; Rawles et al., 2023).

Figure 1: Overview of VeriGUI's closed-loop framework. Unlike blind action agents (above) that repeat failed actions indefinitely, VeriGUI (below) verifies each action's outcome against predicted effects, enabling failure detection and reasoned recovery.

These capabilities have driven rapid progress in benchmark performance (Zhou et al., 2024; Xie et al., 2024; Rawles et al., 2025) and broadened the scope of automated interaction. However, deploying such agents in real-world environments exposes a fundamental weakness shared by most existing approaches: they implicitly assume that every issued action executes as intended.

This assumption rarely holds in practice. Network latency may prevent a page from loading; rendering delays can cause click targets to shift; system interruptions may block expected transitions

*Work done during the internship at Baidu.
†Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 28924–28941

or interrupt animations. When such failures occur, current agents continue operating as if nothing went wrong—they observe an unchanged screen yet generate another action based on flawed assumptions about their progress (Huang et al., 2024). Worse still, because these agents have rarely encountered failure scenarios during training, they tend to repeat the exact same ineffective action, creating infinite execution loops that waste computational resources without advancing the task. Importantly, this class of idempotent failures—where the erroneous action leaves the screen unchanged—is not a corner case. Empirical evidence confirms their prevalence: execution timeouts caused by repeated ineffective actions account for 72.3% of all failures across 1,265 task executions (Liu et al., 2026); the Reasonable Operation Ratio metric was introduced in AndroidLab precisely to capture such non-progressive operations (Xu et al., 2025); and failed grounding and stuck repetitive behaviors have been identified as primary failure modes in real mobile deployments (Li et al., 2025).

Human users naturally avoid this trap. After each interaction, we implicitly verify whether the expected change occurred: did the button highlight, did the page navigate, or did new content appear? If something seems wrong, we pause, diagnose the issue, and adjust our approach accordingly. This verification–reasoning–correction loop forms the basis of robust human-computer interaction (Wei et al., 2022; Yao et al., 2023), yet it is conspicuously absent from current GUI Agent designs.

Training agents to exhibit such behavior presents its own challenges. Online reinforcement learning in live GUI environments suffers from prohibitive interaction latency, system instability, and poor scalability, making large-scale training impractical—requiring up to 64 parallel Android emulators (Bai et al., 2024) or complex distributed infrastructure (Wang et al., 2025a). Offline datasets enable faster iteration but lack explicit feedback signals indicating whether an action failed and how recovery should proceed (Li et al., 2024; Rawles et al., 2023). This gap between training conditions and deployment realities fundamentally limits the robustness of existing approaches. Prior work on self-correction in LLMs (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024) has shown promise in iterative refinement through verbal feedback, but these methods typically assume access to external feedback signals or oracle information, and have not been effectively adapted to the visual grounding requirements of GUI automation (Kumar et al., 2024).

We propose VeriGUI (**Veri**fication-driven **GUI** Agent) to address these limitations. Our approach makes three contributions:

- We propose a T-V-A-E framework that forms a closed reasoning loop, enabling outcome verification, failure diagnosis, and expected effect prediction to prevent blind error accumulation.
- We design a **two-stage training pipeline** combining Robust SFT on synthetic failure trajectories with GRPO that leverages GUI failure idempotency to simulate online feedback from offline data, using asymmetric penalties to promote honest self-assessment. This approach avoids the prohibitive infrastructure demands of online RL while still producing effective self-correcting behavior.
- We construct a **Robustness Benchmark** with controlled failure-injection cases and two novel metrics (Loop Rate and Recovery Success Rate) to reveal failure modes hidden in conventional success-only evaluations. On this benchmark, VeriGUI-3B achieves a Recovery Success Rate of 51.1%, and VeriGUI-7B achieves 52.5%, both surpassing all tested open-source baselines.

2 Related Work

2.1 Vision-Language Models for GUI Agents

Early GUI Agents relied on structured interface representations such as HTML trees or accessibility metadata, limiting robustness across heterogeneous platforms (Deng et al., 2023; Zhang et al., 2025a; Zheng et al., 2024). Recent VLMs enable direct visual grounding from raw screenshots, substantially improving cross-platform generalization: CogAgent (Hong et al., 2024) and SeeClick (Cheng et al., 2024) demonstrate accurate UI element localization and action prediction without structured inputs , while ShowUI (Lin et al., 2024), Ferret-UI (You et al., 2024), and UI-TARS (Qin et al., 2025) further enhance efficiency and generalization through UI-specific pretraining and unified vision–language–action modeling. Subsequent work extends to multi-step reasoning and planning via search, reflection, and execution-feedback-driven replanning (Yu et al., 2025; Antoniades et al., 2024; Zhang et al., 2025b; Wu et al., 2025b; Putta et al.,

2024; Wang et al., 2025b; Wu et al., 2025a), but typically assumes that selected actions execute as intended at the step level and handles failures implicitly through replanning. In contrast, VeriGUI explicitly models action–effect consistency by verifying whether each action induces its intended visual outcome, enabling principled step-level failure detection and recovery.

2.2 Reinforcement Fine-Tuning and Learning for GUI Agents

Reinforcement learning (RL) has become central to improving GUI Agent robustness and generalization, evolving from trajectory-level optimization in DigiRL (Bai et al., 2024) and DistRL (Wang et al., 2025a), and Android Coach (Gan et al., 2026) to preference-based learning with Direct Preference Optimization (DPO) (Rafailov et al., 2024) and self-generated supervision (Kumar et al., 2024; Welleck et al., 2022). For long-horizon reasoning, Group Relative Policy Optimization (GRPO) provides a stable and scalable framework underlying recent reasoning-oriented models (Guo et al., 2025). Building on this line of work, GRPO has been adopted in GUI Agents such as UI-R1 (Lu et al., 2025b), which applies it to unified action spaces, and InfiGUI-R1 (Liu et al., 2025), which integrates reactive execution with deliberative reasoning, and UILoop (Li et al., 2026), which designs UI-element driven reward signals to explicitly improve UI comprehension during reinforcement fine-tuning. However, most RL-based GUI Agents optimize task-level success or proxy step correctness (Chen et al., 2025), while VeriGUI treats action–effect verification as a first-class reinforcement objective, enabling the agent to learn honest self-monitoring and robust self-correction.

3 Methodology

We present VeriGUI, a Verification-driven GUI Agent that addresses the fundamental limitation of current GUI automation systems: their inability to verify execution outcomes and recover from failures. As illustrated in Figure 2, our approach combines a closed-loop inference cycle with a two-stage training pipeline that systematically develops verification-driven self-correction capabilities.

3.1 Problem Formulation

We formalize GUI automation as a sequential decision-making process where an agent must execute interactions to satisfy a natural language instruction I. At each timestep t, the agent observes a screen state S_t and an action history H_t, and selects an action A_t from a discrete action space $\mathcal{A} = \{\text{click}, \text{scroll}, \text{input\_text}, \text{long\_press}...\}$ according to policy $\pi_\theta(A_t \mid S_t, H_t, I)$.

Unlike conventional approaches that assume deterministic transitions, we explicitly model execution uncertainty. We augment the policy to jointly predict structured Thinking T_t, verification judgment V_t, action A_t, and expected effect E_t. This formulation enables the agent to detect failures by verifying V_t against visual evidence and self-correct without external supervision.

3.2 TVAE Inference Cycle

At the core of VeriGUI lies a structured reasoning process mirroring human interaction patterns (Figure 2A). Crucially, TVAE is not a linear chain but a temporally linked cycle: the expected effect predicted at step t becomes the verification hypothesis at step t+1. This temporal dependency enforces causal consistency across steps, ensuring that errors cannot be ignored or overwritten by subsequent actions. At each step t, the agent produces:

1. Think (T_t). Structured analysis using explicit tags ([Verify], [Recall], [Grounding], [Action]). When correcting errors, the structure shifts to [Diagnose] and [Recovery], ensuring verification is grounded in visual reasoning. Note that the [Verify] tag within <think> constitutes the internal reasoning process, while the <verification> output below represents the final binary judgment derived from that reasoning.

2. Verification (V_t). A binary judgment assessing the previous step:

$$V_t = \begin{cases} \text{SUCCESS} & \text{if } S_t \text{ matches } E_{t-1} \\ \text{NO\_CHANGE} & \text{if } S_t \text{ not match } E_{t-1} \end{cases} \tag{1}$$

3. Action (A_t). Executable JSON (for instance, {"action": "click", "coordinate": [x,y]}).

4. Expected Effect (E_t). A prediction of the resulting screen change, serving as the verification target for step $t + 1$.

3.3 Stage 1: Robust Supervised Fine-Tuning

To teach error recognition, we construct a mixed dataset of positive and synthetic negative samples:

Type A: Success Trajectories. Standard steps where S_t reflects the successful execution of A_{t-1}.

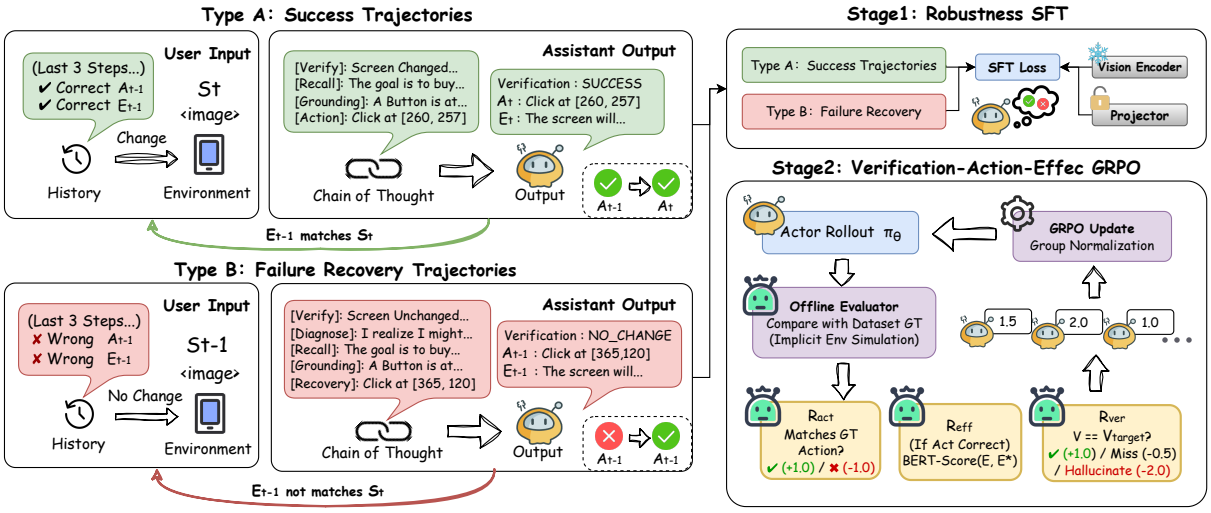

Figure 2: Overview of VeriGUI's architecture and training pipeline. **(A)** The TVAE inference cycle implements a closed-loop verification mechanism where structured thinking precedes verification. **(B)** The two-stage training pipeline: Stage 1 establishes basic verification capabilities via Robust SFT on mixed success/failure trajectories; Stage 2 refines self-correction via GRPO, using offline data to simulate online feedback through composite rewards.

The target output confirms success (SUCCESS) and proceeds with the next ground-truth action.

Type B: Failure Recovery Trajectories. We simulate failures by pairing the previous screen S_{t-1} with history claiming action A_{t-1} was executed. This creates a "no change" scenario. The target output requires the agent to diagnose the failure (NO_CHANGE) and generate a corrective action (typically a refined A^*_{t-1}).

We employ GPT-4o to generate structured Chain-of-Thought (CoT) annotations for both types and train using standard cross-entropy loss(Appendix E). This stage serves as a necessary behavioral prior: without explicit exposure to failure cases during imitation learning, the model tends to overfit to optimistic assumptions that all actions succeed, making subsequent reinforcement learning unstable.

3.4 Stage 2: Verification-Action-Effect GRPO

While SFT establishes basic capabilities, it lacks the feedback loop required to internalize "honest" verification. Stage 2 employs Group Relative Policy Optimization (GRPO) with a specialized reward mechanism that simulates online feedback using only offline data.

Implicit Environment Simulation. Direct reinforcement learning in live GUI environments suffers from prohibitive interaction latency, system instability, and scalability bottlenecks—requiring up to 64 parallel Android emulators (Bai et al., 2024) or complex distributed infrastructure (Wang et al., 2025a). To bypass these inefficiencies while preserving realistic feedback dynamics, we employ a **data-driven implicit simulation strategy**. We leverage the idempotency property of GUI errors: *incorrect actions typically leave the screen unchanged*. This assumption is well-grounded in practice—execution timeouts from repeated ineffective actions, screen-invariant operation rates, and failed grounding with no state transition together constitute the dominant failure modes in real mobile environments (Xu et al., 2025; Liu et al., 2026; Li et al., 2025)—and enables us to transform offline trajectories into training samples with meaningful environment feedback signals without any live interaction.

We construct the RL training data from the mixed SFT dataset. For any given input state (which may represent a successful or failed history), the model generates a response. We then determine the verification ground truth V_{target} dynamically based on the sample type:

- For Type A inputs (Success History), V_{target} = SUCCESS.
- For Type B inputs (Failure History), V_{target} = NO_CHANGE.

Crucially, the model is rewarded only if its predicted verification $\hat{V}_t$ matches this objective reality, regardless of whether it "thinks" it succeeded. This effectively simulates environment feedback: if the screen

didn't change (Type B), the environment "tells" the agent it failed by rewarding only NO_CHANGE predictions. This implicit feedback mechanism is intentionally minimalistic: it does not reveal the correct action or recovery strategy, but only signals whether the agent's belief about the outcome is consistent with visual reality. As a result, the agent must rely on its own reasoning and expected-effect predictions to determine how to proceed, closely mirroring real-world deployment conditions.

Composite Reward Function. The reward function R_t drives the optimization of the TVAE cycle components:

$$R_t = R_{\text{act}} + \alpha \cdot R_{\text{eff}} + \beta \cdot R_{\text{ver}} \tag{2}$$

Together, these reward components explicitly couple action execution with outcome awareness, ensuring that high task completion scores cannot be achieved by systematically ignoring verification signals.

1. **Action Reward (R_{act}):** Measures execution correctness by comparing $\hat{A}_t$ against the ground-truth action A_t^* (using type matching and coordinate IoU).

$$R_{\text{act}} = \mathbb{I}(d(\hat{A}_t, A_t^*) \le \delta) - \mathbb{I}(d(\hat{A}_t, A_t^*) > \delta) \tag{3}$$

2. **Effect Reward (R_{eff}):** Evaluates semantic consistency between the predicted expected effect $\hat{E}_t$ and the reference effect E_t^* generated by GPT-4o during dataset construction (Appendix E). R_{eff} is computed via BERTScore when the action is correct ($d(\hat{A}_t, A_t^*) \le \delta$), and set to 0 otherwise:

$$R_{\text{eff}} = \begin{cases} \text{BERTScore}(\hat{E}_t,\ E_t^*) & \text{if } d(\hat{A}_t, A_t^*) \le \delta \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

3. **Verification Reward (R_{ver}):** Enforces honest self-monitoring with asymmetric penalties:

$$R_{\text{ver}} = \begin{cases} +1.0 & \text{if } \hat{V}_t = V_{\text{target}} \\ -0.5 & \text{if False Negative (Miss)} \\ -2.0 & \text{if False Positive (Hallucination)} \end{cases} \tag{5}$$

The severe penalty for hallucination (−2.0) forces the agent to align its internal belief with visual reality, preventing the accumulation of errors.

Optimization. We employ Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to optimize the policy. For each input x, GRPO samples a group of G outputs $\{y_1, \ldots, y_G\}$ from the current policy π_θ. Let $R_t(y_i)$ denote the composite reward defined in Eq. 2 evaluated for output y_i. The group-normalized advantage for the i-th output is:

$$\hat{A}_i = \frac{R_t(y_i) - \text{mean}\big(\{R_t(y_j)\}_{j=1}^G\big)}{\text{std}\big(\{R_t(y_j)\}_{j=1}^G\big) + \varepsilon} \tag{6}$$

where ε is a small constant for numerical stability. For each token position k in output y_i, we define the probability ratio between the updated policy and the old sampling policy as:

$$\rho_{i,k} = \frac{\pi_\theta(y_{i,k} \mid y_{i,<k},\, x)}{\pi_{\theta_{\text{old}}}(y_{i,k} \mid y_{i,<k},\, x)} \tag{7}$$

The clipped surrogate loss for each token is then:

$$\mathcal{L}_{i,k}^{\text{clip}} = \min\Big(\rho_{i,k}\,\hat{A}_i,\ \text{clip}(\rho_{i,k},\, 1{-}\epsilon_{\text{clip}},\, 1{+}\epsilon_{\text{clip}})\,\hat{A}_i\Big) \tag{8}$$

where ϵ_{clip} restricts the probability ratio to $[1{-}\epsilon_{\text{clip}},\, 1{+}\epsilon_{\text{clip}}]$. The final GRPO objective aggregates the clipped surrogate loss over all outputs and tokens, with a KL penalty constraining deviation from the reference policy π_{ref}:

$$\mathcal{J}_{\text{GRPO}} = \mathbb{E}_x\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{L_i}\sum_{k=1}^{L_i}\mathcal{L}_{i,k}^{\text{clip}}\right] - \lambda\, D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}}) \tag{9}$$

where L_i is the token length of the i-th output and λ controls the KL regularization strength. The complete inference and training procedures are detailed in Algorithms 1 and 2 (Appendix B).

4 Experiments

We evaluate VeriGUI across multiple benchmarks: AndroidControl-High (Li et al., 2024), AITW-Gen (Rawles et al., 2023), and GUI Odyssey (Lu et al., 2025a) as our offline evaluation suite, and MiniWoB++ (Liu et al., 2018) and AndroidWorld (Rawles et al., 2025) as fully online environments to validate real-world robustness.

4.1 Experimental Setup

Dataset and Baselines. AndroidControl-High provides approximately 15,000 training trajectories and 2,500 test trajectories spanning productivity, social media, e-commerce, and system applications. We compare against 3B models (Qwen2.5-VL-3B

(Bai et al., 2025), UI-R1-3B (Lu et al., 2025b)) , 7B/8B models (OS-Genesis-7B (Sun et al., 2025), OS-Atlas-7B (Wu et al., 2024), Qwen2.5-VL-7B, AgentCPM-GUI-8B (Zhang et al., 2025c), UI-TARS-7B (Qin et al., 2025), UI-S1-7B (Lu et al., 2025c)), and closed-source systems GPT-5.1 and Gemini-3-flash.

Implementation. We apply our two-stage training pipeline at two scales: VeriGUI-3B builds on Qwen2.5-VL-3B, and VeriGUI-7B builds on Qwen2.5-VL-7B. Both variants use the same pipeline configuration. Stage 1 trains for 2 epochs with learning rate 1×10^{-5}; Stage 2 runs 15 epochs with learning rate 5×10^{-6}, group size $G = 6$, and reward weights $\alpha = 0.5$, $\beta = 0.5$. Training uses 8×A100 GPUs. Full hyperparameters appear in Appendix D.

4.2 Evaluation Protocol

Our offline evaluation exploits the *failure idempotency* of GUI environments to construct a pseudo-online simulation: correct actions advance to ground-truth next screens, while incorrect actions return the unchanged screen. Formal metric definitions appear in Table 13 in Appendix C.

Step Level Metrics. Step-level metrics assess single-step predictions independently: Type Match (TM), Grounding Rate (GR), and Step Success Rate (SR).

Task Level Metrics. Task-level metrics evaluate end-to-end execution under pseudo-online simulation with a $2T_{\text{GT}}$ step budget: Task Success Rate (TSR), Progress (PG), Simulated Task Success Rate (Sim-TSR), and Average Step Overhead (ASO).

Robustness Metrics. We construct failure-injection test cases by pairing unchanged screens with histories claiming an action was executed. Loop Rate (LR) measures repeated failed actions; Recovery Success Rate (RSR) measures correct recovery actions.

4.3 Main Results

Offline Benchmark. Table 1 presents comprehensive results across all offline benchmarks. On AndroidControl-High, VeriGUI-3B achieves Type Match of 72.2%, the highest among all open-source models at the 3B scale, and VeriGUI-7B reaches 74.2%, establishing a new open-source state of the art. Notably, VeriGUI-3B surpasses several 7B-scale baselines on this metric despite having less than half the parameters, which demonstrates that the TVAE framework's structured reasoning actively improves action type selection rather than merely adding overhead.

The task-level metrics reveal the practical value of verification-driven self-correction most clearly. All 3B-scale baselines without explicit verification fail entirely under pseudo-online conditions, achieving zero Sim-TSR, while larger 7B-scale models achieve modest results. VeriGUI-3B completes 16.7% of tasks with an ASO of just 1.25, and VeriGUI-7B further improves to 23.5% Sim-TSR with an ASO of 1.09—both substantially outperforming all comparably sized open-source models. The gap between TSR and Sim-TSR directly quantifies tasks completed through error recovery rather than flawless execution.

Cross-distribution results on AITW-Gen and GUI Odyssey further validate generalization. VeriGUI-7B achieves 72.7 TM and 52.3 SR on GUI Odyssey, exceeding UI-TARS-7B on both metrics and closely approaching UI-S1-7B's SR of 52.8 while surpassing it on TM. VeriGUI-3B shows consistent improvements over same-scale baselines on GUI Odyssey across all metrics, and achieves higher Sim-TSR than both 3B baselines on AITW-Gen, confirming that the TVAE framework provides transferable benefits.

Robustness Benchmark. Table 2 evaluates error recognition and recovery directly. UI-TARS-7B achieves the lowest Loop Rate at 13.4% but only 45.5% RSR—it avoids repeating failed actions but often produces incorrect alternatives. OS-Atlas-7B shows high Loop Rate of 43.3% with poor RSR of 10.7%, indicating neither recognition nor correction capability.

VeriGUI-3B achieves RSR of 51.1%, surpassing UI-TARS-7B by over five points despite having less than half the parameters, and VeriGUI-7B further improves to 52.5%, setting a new open-source best. This validates our core design: the TVAE framework trains the model not merely to detect failures but to reason about their causes and generate appropriate corrections. The Loop Rate of 24.3% for VeriGUI-3B represents a meaningful reduction from same-scale baselines like Qwen2.5-VL-3B at 30.0%, while VeriGUI-7B achieves 15.6%, approaching the best open-source Loop Rate of UI-TARS-7B.

Online Benchmark Evaluation. To validate robustness beyond pseudo-online simulation, we eval-

Category	Model	AndroidControl-High							AITW-Gen		GUI Odyssey		
		TM	GR	SR	TSR	PG	Sim-TSR	ASO↓	PG	Sim-TSR	TM	GR	SR
Closed-source	GPT-5.1	70.1	30.0	23.1	–	–	–	–	10.4	2.5	68.1	34.0	27.9
	Gemini-3-flash	83.3	36.9	42.2	–	–	–	–	17.8	4.5	83.2	48.8	46.9
Open-src 3B	Qwen2.5-VL-3B	60.1	52.8	31.2	0	2.8	0	∞	1.5	0.1	62.0	47.4	39.9
	UI-R1-3B	62.4	50.4	30.3	0	4.9	0	∞	8.8	2.4	62.5	52.6	32.0
Open-src 7B/8B	Qwen2.5-VL-7B	68.5	61.7	42.8	6.2	12.8	10.6	1.15	11.7	3.0	72.3	55.8	48.5
	OS-Genesis-7B	64.5	47.5	35.7	0	6.5	0	∞	6.5	2.2	62.3	59.9	34.5
	OS-Atlas-7B	67.7	42.2	35.5	0	5.5	0	∞	12.3	3.4	70.2	63.5	41.3
	AgentCPM-GUI-8B	42.2	**66.4**	34.1	0	7.1	0	∞	12.0	3.4	70.3	50.2	47.8
	UI-TARS-7B	71.9	60.6	47.7	10.0	14.7	13.3	1.67	16.8	**5.9**	72.6	62.1	52.0
	UI-S1-7B	70.1	53.6	44.5	13.3	25.4	16.7	1.33	**19.5**	4.6	70.5	62.8	**52.8**
Ours	VeriGUI-3B	72.2	56.4	46.6	13.3	23.3	16.7	1.25	7.4	2.6	65.5	54.5	43.4
	VeriGUI-7B	**74.2**	65.5	**51.1**	**16.7**	**30.6**	**23.5**	**1.09**	15.5	5.5	**72.7**	**63.2**	52.3

Table 1: Offline benchmark results. VeriGUI is trained at both 3B and 7B scales. **Bold** indicates best among all open-source models. ∞ ASO indicates no tasks completed.

Category	Model	LR↓	RSR↑
Closed-source	GPT-5.1	11.0	21.9
	Gemini-3-flash	15.0	37.0
Open-source 3B	Qwen2.5-VL-3B	30.0	35.0
	UI-R1-3B	29.5	29.0
Open-source 7B/8B	Qwen2.5-VL-7B	23.5	34.1
	OS-Atlas-7B	43.3	10.7
	UI-TARS-7B	**13.4**	45.5
	UI-S1-7B	20.5	43.6
Ours	VeriGUI-3B	24.3	51.1
	VeriGUI-7B	15.6	**52.5**

Table 2: Robustness Benchmark results. **Bold** indicates best among all open-source models; VeriGUI achieves the highest Recovery Success Rate at both scales.

Category	Model	MiniWoB++	AW
Open-source 3B	Qwen2.5-VL-3B	21.2	6.1
	UI-R1-3B	33.4	10.9
Open-source 7B	Qwen2.5-VL-7B	47.0	14.9
	UI-S1-7B	56.6	22.7
Ours	VeriGUI-3B	35.6	12.6
	VeriGUI-7B	**59.7**	**25.1**

Table 3: Online benchmark results. **Bold** indicates best among all open-source models; VeriGUI-7B outperforms all open-source baselines; VeriGUI-3B surpasses same-scale models and approaches 7B-scale performance.

uate VeriGUI on two fully online benchmarks where environmental uncertainty arises organically: MiniWoB++, which executes real web interactions in a live browser involving dynamic rendering and network variability; and AndroidWorld (AW), deployed on Android emulators via Android Studio, where real application execution introduces system-level dynamics. Neither benchmark involves artificial failure injection. Table 3 summarizes the results.

VeriGUI-3B achieves 35.6% on MiniWoB++ and 12.6% on AndroidWorld, surpassing all 3B baselines on both benchmarks and approaching the performance of 7B-scale models. VeriGUI-7B reaches 59.7% and 25.1% respectively, outperforming all open-source baselines including UI-S1-7B. These results are especially encouraging because failure modes in live environments—navigation errors, rendering races, application-level exceptions—are structurally more diverse than the idempotency-based failures modeled during training. The gains transferring to non-idempotent settings suggest that our verification-and-recovery training instills a general habit of self-checking rather than a narrow pattern tied to unchanged screens.

4.4 Ablation Studies

The following ablation studies all use VeriGUI-3B on AndroidControl-High to isolate individual design decisions. We use the 3B configuration throughout for computational efficiency and to ensure fair apples-to-apples comparison across variants; the full model's 7B results appear in the main comparison tables above.

Training Stage Composition. Table 4 isolates each stage's contribution. Standard SFT improves step-level accuracy significantly but provides no verification benefit—Loop Rate actually increases slightly as the model becomes more confident in sometimes incorrect actions. Robust SFT with 30% synthetic failures introduces genuine error recognition, with RSR jumping from 29.7% to 45.5%. GRPO further refines all capabilities; notably ASO

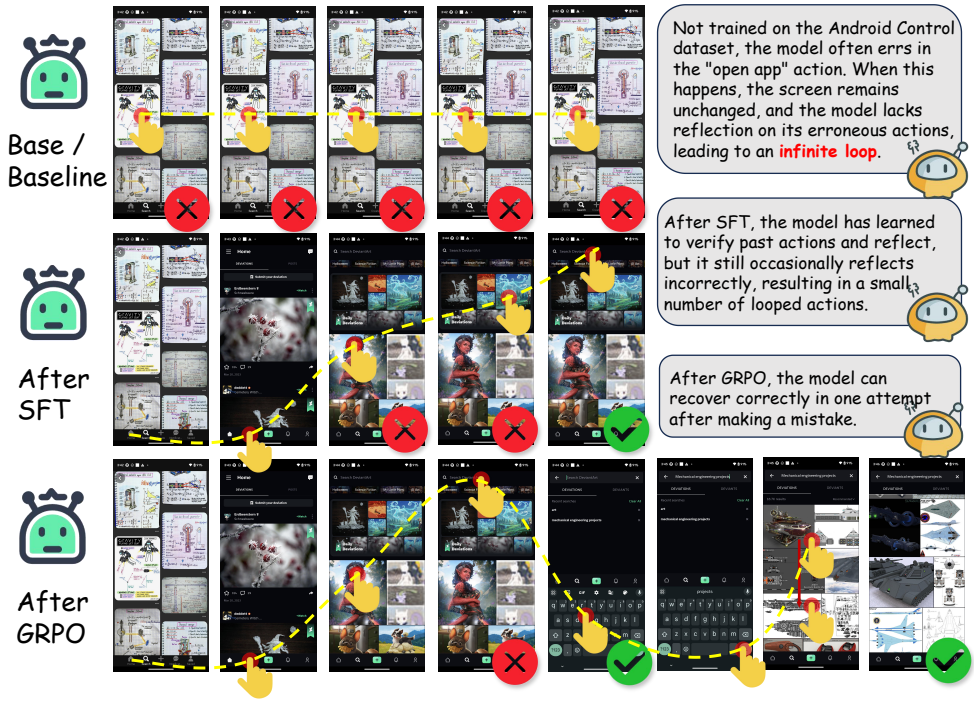

Figure 3: Qualitative comparison across training stages. The base model repeatedly attempts the same ‘‘open app’’ action without recognizing failure, entering an infinite loop. After SFT, the model learns to verify and reflect on past actions, though occasional incorrect reflections still cause short loops. After GRPO, the model correctly identifies the failure after one attempt and successfully recovers with an alternative action.

Configuration	Step-level (AndroidControl)			Task-level (AndroidControl)				Robustness	
	TM	GR	SR	TSR	PG	Sim-TSR	ASO↓	LR↓	RSR
Base (Qwen2.5-VL-3B)	60.1	52.8	31.2	0	2.8	0	∞	30.0	35.0
+ Standard SFT	67.2	56.6	42.6	6.7	16.4	10.0	3.33	30.4	29.7
+ Robust SFT	68.7	52.4	41.7	10.0	20.8	13.3	2.25	26.5	45.5
+ GRPO (VeriGUI-3B)	**72.2**	**56.4**	**46.6**	**13.3**	**23.3**	**16.7**	**1.25**	**24.3**	**51.1**

Table 4: Ablation on training stages (VeriGUI-3B, AndroidControl-High). Standard SFT uses only Type A (Success Trajectories) samples, while Robust SFT leverages both Type A and Type B (Failure Recovery Trajectories) samples.

Type A:B Ratio	SR	Sim-TSR	LR↓	RSR
100:0 (no negatives)	42.6	10.0	30.4	29.7
90:10	42.1	10.0	29.6	30.3
70:30 (default)	41.7	13.3	26.5	**45.5**
50:50	39.2	12.4	27.1	45.4

Table 5: Effect of negative sample ratio (VeriGUI-3B, AndroidControl-High). The 70:30 balance maximizes RSR while maintaining standard accuracy.

Reward Components	SR	Sim-TSR	LR↓	RSR
R_{act} only	42.9	13.6	25.5	44.9
R_{act} + R_{ver}	44.2	15.1	25.4	48.1
R_{act} + R_{eff} + R_{ver}	**46.6**	**16.7**	**24.3**	**51.1**

Table 6: Ablation on reward components (VeriGUI-3B, AndroidControl-High). All three components contribute to the final performance.

drops from 2.25 to 1.25, indicating RL teaches efficient recovery rather than trial-and-error.

Negative Sample Ratio. Table 5 examines the sample ratio in Robust SFT. Without failure samples, no verification capability develops. RSR peaks at 45.5% with the 70:30 ratio; excessive negatives at 50:50 degrade Step Accuracy while yielding diminishing robustness gains.

Reward Components. Table 6 analyzes the composite reward. Action reward alone achieves 44.9% RSR. Adding the verification reward with asymmetric penalties improves RSR to 48.1%, and the full reward including effect prediction reaches best performance across all metrics, validating our design to treat effect prediction as a first-class TVAE component.

5 Analysis

5.1 Case Study

Figure 3 illustrates the progressive improvement in failure handling across training stages through a concrete example. When an ‘‘open app’’ action fails due to misaligned coordinates, the base model observes an unchanged screen but lacks any mechanism to recognize this discrepancy. Without verification capability, it repeats identical actions indefinitely, entering the infinite loop pattern we identified as the dominant failure mode in baseline agents.

After Robust SFT, the model acquires basic verification capability through exposure to synthetic failure scenarios. It can sometimes recognize that the screen has not changed and attempt different actions, but occasional incorrect assessments still produce short loops where the model fails to correctly diagnose the root cause. After GRPO, the model reliably detects the unchanged screen, correctly diagnoses that the coordinate error caused the failure, and generates a successful recovery action on the first attempt. The asymmetric penalty structure during RL training teaches the model that overconfident success claims carry severe consequences, while honest uncertainty acknowledgment is acceptable. This progression validates our two-stage design: SFT establishes the structural capability for verification, while GRPO refines its accuracy and reliability.

5.2 Computational Overhead

The TVAE framework introduces additional output tokens for verification, reasoning, and effect prediction. Table 7 reports trajectory-level efficiency measured over 50 AndroidControl-High trajectories. Relative to same-scale baselines, per-step output length increases by approximately 45% for both VeriGUI-3B and VeriGUI-7B. However, this overhead is substantially amortized at the trajectory level because the recovery mechanism breaks failure loops early.

As shown in Table 7, VeriGUI-3B’s trajectory-level time rises by only ∼26% relative to Qwen2.5-VL-3B, while Tok/Trajectory increases by just 6%. More strikingly, VeriGUI-7B’s Tok/Trajectory of 712 falls *below* Qwen2.5-VL-3B’s 847: stronger recovery eliminates more failure loops than the additional reasoning tokens cost. Furthermore, all model inference in our deployment architecture runs cloud-side; the mobile client handles only rendering and execution, so additional reasoning tokens do not directly affect device battery or latency. Overall, VeriGUI represents a favorable efficiency trade-off—substantially higher task success at modest additional inference cost.

Model	Tok/Step	Tok/Traj	Time/Step (s)	Time/Traj (s)
Qwen2.5-VL-3B	76.3	847	1.06	12.1
Qwen2.5-VL-7B	81.5	724	1.29	11.3
VeriGUI-3B	110.8	896	1.85	15.3
VeriGUI-7B	117.2	712	2.40	14.6

Table 7: Trajectory-level efficiency on AndroidControl-High (50 trajectories). Per-step token and time costs increase with TVAE, but recovery reduces total trajectory length, keeping trajectory-level overhead modest.

6 Conclusion

We presented VeriGUI, a GUI Agent that addresses the ‘‘blind action’’ limitation of existing approaches through a Thinking–Verification–Action–Expectation framework. Our two-stage training pipeline combines Robust SFT with synthetic failure trajectories and GRPO with asymmetric verification rewards. Both VeriGUI-3B and VeriGUI-7B show consistent gains across step-level accuracy, task completion, robustness to execution failures, and online benchmarks. The verification-and-recovery mechanism generalizes to dynamic real-world environments beyond the controlled training setting, and cross-distribution evaluation further demonstrates reasonable transfer across application domains. We hope this work motivates more explicit treatment of execution uncertainty in future GUI Agent designs.

Limitations

The current robustness evaluation is built on the *failure idempotency* assumption—that failed actions leave the screen unchanged—which does not cover more complex failure modes such as unintended navigations, partial transitions, or application crashes. Extending the framework to a finer-grained failure taxonomy that handles these non-idempotent cases remains important future work. Additionally, while step-level verification effectively addresses local execution errors, performance degrades with task length, as it does not substitute for hierarchical planning in long-horizon settings. Developing mechanisms that combine step-level self-correction with longer-range task awareness is a natural next step.

References

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Yang Wang. 2024. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. *Preprint*, arXiv:2410.20285.

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

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

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. 2025. Gui-shepherd: Reliable process reward and verification for long-sequence gui tasks. *Preprint*, arXiv:2509.23738.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, Bangkok, Thailand. Association for Computational Linguistics.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, volume 36, pages 28091–28114. Curran Associates, Inc.

Guo Gan, Yuxuan Ding, Cong Chen, Yuwei Ren, Yin Huang, and Hong Zhou. 2026. Android coach: Improve online agentic training efficiency with single state multiple actions. *Preprint*, arXiv:2604.07277.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

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

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. 2024. Training language models to self-correct via reinforcement learning. *Preprint*, arXiv:2409.12917.

Ning Li, Xiangmou Qu, Jiamu Zhou, Jun Wang, Muning Wen, Kounianhua Du, Xingyu Lou, Qiuying Peng, Jun Wang, and Weinan Zhang. 2025. Mobileuse: A gui agent with hierarchical reflection for autonomous mobile operation. *Preprint*, arXiv:2507.16853.

Songze Li, Xiaoke Guo, Tianqi Liu, Biao Yi, Zhaoyan Gong, Zhiqiang Liu, Huajun Chen, and Wen Zhang. 2026. What's missing in screen-to-action? towards a ui-in-the-loop paradigm for multimodal gui reasoning. *Preprint*, arXiv:2604.06995.

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

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2024. Training a vision-language-action model as generalist GUI agent. In *NeurIPS 2024 Workshop on Open-World Agents*.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. *Preprint*, arXiv:1802.08802.

Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Qinyi Luo, Shunye Tang, Yuxiang Chai, Weifeng Lin, Han Xiao, WenHao Wang, Siheng Chen, Zhengxi Lu, Gao Wu, Hao Wang, Liang Liu, and Yong Liu. 2026. Memgui-bench: Benchmarking memory of mobile gui agents in dynamic environments. *Preprint*, arXiv:2602.06075.

Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. 2025. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection. *Preprint*, arXiv:2501.04575.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. 2025a. Guiodyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *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. *Preprint*, arXiv:2503.21620.

Zhengxi Lu, Jiabo Ye, Fei Tang, Yongliang Shen, Haiyang Xu, Ziwei Zheng, Weiming Lu, Ming Yan, Fei Huang, Jun Xiao, and Yueting Zhuang. 2025c. Ui-s1: Advancing gui automation via semi-online reinforcement learning. *Preprint*, arXiv:2509.11543.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594. Curran Associates, Inc.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *Preprint*, arXiv:2408.07199.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, and 16 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *Preprint*, arXiv:2501.12326.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Preprint*, arXiv:2305.18290.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. *Preprint*, arXiv:2405.14573.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the wild: A large-scale dataset for android device control. *Preprint*, arXiv:2307.10088.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. 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, Vienna, Austria. Association for Computational Linguistics.

Taiyi Wang, Zhihao Wu, Jianheng Liu, Jianye Hao, Jun Wang, and Kun Shao. 2025a. Distrl: An asynchronous distributed reinforcement learning framework for on-device control agents. *Preprint*, arXiv:2410.14803.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. 2025b. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *Preprint*, arXiv:2501.11733.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2022. Generating sequences by learning to self-correct. *Preprint*, arXiv:2211.00053.

Qinzhuo Wu, Pengzhi Gao, Wei Liu, and Jian Luan. 2025a. Backtrackagent: Enhancing gui agent with error detection and backtracking mechanism. *Preprint*, arXiv:2505.20660.

Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. 2025b. Reachagent: Enhancing mobile agent via page reaching and operation. *Preprint*, arXiv:2502.02955.

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.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu,

Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pages 52040–52094. Curran Associates, Inc.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2025. AndroidLab: Training and systematic benchmarking of android autonomous agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2144–2166, Vienna, Austria. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In *European Conference on Computer Vision*, pages 240–255. Springer.

Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. 2025. Exact: Teaching ai agents to explore with reflective-mcts and exploratory learning. In *International Conference on Learning Representations (ICLR)*.

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*, CHI '25, New York, NY, USA. Association for Computing Machinery.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. 2025b. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(22):23378–23386.

Zhong Zhang, Yaxi Lu, Yikun Fu, Yupeng Huo, Shenzhi Yang, Yesai Wu, Han Si, Xin Cong, Haotian Chen, Yankai Lin, Jie Xie, Wei Zhou, Wang Xu, Yuanheng Zhang, Zhou Su, Zhongwu Zhai, Xiaoming Liu, Yudong Mei, Jianming Xu, and 6 others. 2025c. AgentCPM-GUI: Building mobile-use agents with reinforcement fine-tuning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 155–180, Suzhou, China. Association for Computational Linguistics.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. *Preprint*, arXiv:2401.01614.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

A Additional Results

This section provides detailed breakdowns of our evaluation results that complement the aggregate metrics presented in the main text. Unless otherwise noted, all tables in this appendix report results on the AndroidControl-High test set. Ablation experiments and fine-grained analyses use the VeriGUI-3B configuration for consistency and computational tractability; the full comparison of both 3B and 7B variants appears in the main paper.

A.1 Per-Action-Type Performance

Table 8 presents a fine-grained analysis of Step Accuracy (SR) broken down by action type on AndroidControl-High.

Click actions, which constitute approximately 62% of the test set, show the largest absolute performance gaps between models. VeriGUI-3B achieves 44.2% on click actions, substantially outperforming same-scale baselines like Qwen2.5-VL-3B (27.4%) and approaching the 7B-scale UI-TARS-7B (45.3%). Scroll actions exhibit uniformly high accuracy (68–81%) because scroll directions are more tolerant of coordinate imprecision. The open_app action poses challenges for all models due to the need to recognize app icons by visual appearance.

Table 9 further decomposes errors into type prediction failures versus grounding failures.

A.2 Task Complexity Analysis

We stratify tasks by ground-truth trajectory length: Short (1–4 steps), Medium (5–8 steps), and Long (9+ steps). Table 10 presents this analysis on AndroidControl-High.

Performance degrades with task length across all models. For short tasks, VeriGUI-3B achieves 33.3% Sim-TSR, the highest among all tested models. VeriGUI-3B's verification mechanism provides the largest relative advantage on medium-length tasks, where recovery from a single error can determine task completion. For long tasks exceeding nine steps, all models complete fewer than 6% of trajectories, reflecting the compound difficulty of maintaining correctness across many sequential decisions. As discussed in the Limitations section, step-level verification addresses local execution errors but does not substitute for hierarchical long-horizon planning.

Algorithm 1 TVAE Inference Cycle

Require: Screen S_t, history H_t, instruction I, previous effect E_{t-1}
Ensure: Action A_t, expected effect E_t, verification V_t

// Stage 1: Thinking with Verification
if $t > 1$ **then**
  Compare S_t with E_{t-1}
  if S_t matches E_{t-1} **then**
    $V_t \leftarrow$ SUCCESS
    $T_t \leftarrow$ [Verify][Recall][Grounding][Action]
  else
    $V_t \leftarrow$ NO_CHANGE
    $T_t \leftarrow$ [Verify][Diagnose][Recall][Recovery]
  end if
else
  $V_t \leftarrow$ SUCCESS {First step}
end if
// Stage 2: Action Generation
Generate A_t based on T_t, S_t, H_t, I
// Stage 3: Effect Prediction
Predict E_t describing expected screen change
// Stage 4: Update History
$H_{t+1} \leftarrow H_t \cup \{(A_t, E_t, V_t)\}$
return A_t, E_t, V_t

A.3 First-Step versus All-Steps Performance

Table 11 compares performance on the first step of each trajectory against performance averaged across all steps on AndroidControl-High.

All models exhibit a consistent pattern: first-step accuracy is 5–8 percentage points lower than all-steps accuracy. VeriGUI-3B shows one of the smallest first-to-all gaps (4.0 points for TM, 5.1 for SR), suggesting that the TVAE framework's structured reasoning and expected-effect prediction provide benefits even without action history.

A.4 Detailed Robustness Analysis

Table 12 presents fine-grained measurements of the failure detection and correction pipeline on AndroidControl-High.

VeriGUI-3B's RSR advantage stems from improvements in both type selection and grounding for recovery actions. VeriGUI-3B achieves 79.1% type match rate on recovery actions, substantially higher than UI-TARS-7B's 72.5%. Grounding accuracy for correctly-typed recovery actions (GR_{rec}) similarly improves to 64.6% versus 62.8%. These compound to produce the observed 5.6 percentage point RSR advantage over UI-TARS-7B.

B Algorithm

Algorithm 1 presents the TVAE inference cycle, and Algorithm 2 details the two-stage training pipeline.

Model	Click	Scroll	Input_text	Navigate_back	Open_app	Wait	Overall SR
Qwen2.5-VL-3B	27.4	68.2	18.3	35.2	1.8	32.1	31.2
UI-R1-3B	26.8	70.5	16.9	33.8	2.1	30.6	30.3
OS-Genesis-7B	32.5	74.3	21.4	38.6	1.5	35.8	35.7
OS-Atlas-7B	32.1	72.8	22.7	40.1	1.2	36.4	35.5
Qwen2.5-VL-7B	40.2	78.6	31.5	45.3	2.4	42.7	42.8
AgentCPM-GUI-8B	30.8	71.2	24.6	36.9	1.9	34.5	34.1
UI-TARS-7B	45.3	81.4	35.8	51.2	2.8	48.6	47.7
UI-S1-7B	42.1	79.8	32.4	47.5	2.3	45.2	44.5
VeriGUI-3B	**44.2**	**80.9**	**34.7**	**49.8**	2.6	**47.3**	**46.6**

Table 8: Step Accuracy (%) breakdown by action type on AndroidControl-High (VeriGUI-3B). Click actions dominate the test set (approximately 62%) and largely determine overall performance. Scroll actions exhibit uniformly high accuracy across models due to their tolerance for coordinate imprecision. The open_app action shows consistently low accuracy because most models tend to predict click instead. **Bold** indicates best among all open-source models at 3B scale.

Model	Click		Input_text	
	TM	GR|TM	TM	GR|TM
Qwen2.5-VL-3B	78.5	34.9	28.4	64.4
UI-R1-3B	80.2	33.4	26.8	63.1
OS-Genesis-7B	82.4	39.4	32.5	65.8
OS-Atlas-7B	83.1	38.6	34.2	66.4
Qwen2.5-VL-7B	86.3	46.6	40.8	77.2
AgentCPM-GUI-8B	75.8	40.6	35.6	69.1
UI-TARS-7B	88.2	51.4	44.5	80.4
UI-S1-7B	86.8	48.5	41.2	78.6
VeriGUI-3B	**88.6**	**49.9**	**43.8**	**79.2**

Table 9: Type Match (TM) and Grounding accuracy given correct Type Match (GR|TM) for click and input_text actions (%) on AndroidControl-High (VeriGUI-3B). **Bold** indicates best among all open-source models at 3B scale.

Model	Short	Medium	Long
Sim-TSR (%)			
Qwen2.5-VL-7B	21.4	8.3	2.1
UI-TARS-7B	28.6	10.2	3.4
UI-S1-7B	31.2	13.6	4.8
VeriGUI-3B	**33.3**	**14.5**	**5.6**
PG (%)			
Qwen2.5-VL-7B	24.8	10.5	4.2
UI-TARS-7B	30.5	12.8	5.1
UI-S1-7B	38.2	22.6	8.4
VeriGUI-3B	36.7	20.8	7.8

Table 10: Task-level metrics stratified by trajectory length on AndroidControl-High (VeriGUI-3B). Only models achieving non-zero Sim-TSR in aggregate are included. Performance degrades substantially for longer tasks, consistent with the multiplicative nature of sequential step-level errors. **Bold** indicates best among included models.

C Evaluation Metrics: Formal Definitions

Table 13 provides a comprehensive summary of all evaluation metrics used in this work.

Pseudo-Online Simulation Protocol. The task-level metrics are computed under a pseudo-online simulation exploiting *failure idempotency*. We maintain a simulated environment state initialized at the task's starting screen. At each step: (1) the agent observes S_t and generates $\hat{A}_t$; (2) if $\hat{A}_t$ matches A_t^*, advance to S_{t+1}^*; (3) if $\hat{A}_t \neq A_t^*$, return unchanged S_t; (4) task terminates when completed or after $2 \times T_{\text{GT}}$ steps.

Robustness Benchmark Construction. For robustness evaluation, we construct ''failure slices'' from AndroidControl-High trajectories. For each step t, we create a test case using screen S_{t-1} (simulating unchanged state after a failed action), with history containing the erroneous action A_{err} and its incorrect expected effect. We measure whether the agent repeats A_{err} (counted toward LR) or produces the correct A_t^* (counted toward RSR).

D Implementation Details

VeriGUI is built upon Qwen2.5-VL-3B (for VeriGUI-3B) and Qwen2.5-VL-7B (for VeriGUI-7B) as the respective base models. All experiments are conducted on 8× NVIDIA A100-80GB GPUs. Stage 1 training takes approximately 12 hours; Stage 2 takes approximately 20 hours. Infer-

Model	Type Match (%)		Step Acc. (%)	
	First	All	First	All
Qwen2.5-VL-3B	52.4	60.1	24.8	31.2
UI-R1-3B	54.2	62.4	23.5	30.3
Qwen2.5-VL-7B	62.8	68.5	35.6	42.8
UI-TARS-7B	66.4	71.9	40.2	47.7
UI-S1-7B	64.8	70.1	37.8	44.5
VeriGUI-3B	**68.2**	**72.2**	**41.5**	**46.6**

Table 11: Comparison of first-step and all-steps performance on AndroidControl-High (VeriGUI-3B). First-step metrics are consistently lower, indicating that task initialization without history context is more challenging. **Bold** indicates best among all open-source models at 3B scale.

Model	TM_{rec}	GR_{rec}	RSR
Qwen2.5-VL-3B	62.4	56.1	35.0
UI-R1-3B	58.6	49.5	29.0
OS-Atlas-7B	42.8	25.0	10.7
Qwen2.5-VL-7B	65.2	52.3	34.1
UI-TARS-7B	72.5	62.8	45.5
UI-S1-7B	70.8	61.6	43.6
VeriGUI-3B	**79.1**	**64.6**	**51.1**

Table 12: Detailed robustness metrics on AndroidControl-High (VeriGUI-3B). TM_{rec} denotes type match rate for recovery actions; GR_{rec} denotes grounding accuracy for correctly-typed recovery actions. RSR requires both correct type and correct parameters. **Bold** indicates best among all open-source models at 3B scale.

ence uses greedy decoding (temperature = 0) on a single A100 GPU.

E SFT Dataset Construction

Our Robust SFT dataset comprises two sample types designed to teach both standard execution and error recovery.

Type A: Success Trajectories (70%). For each ground-truth step (S_t, A_t^*, S_{t+1}^*), we construct a training sample where the input contains current screen S_t, instruction I, and history H_t, and the output follows the SUCCESS path through Think, Verification, Action, and Expected Effect.

Type B: Failure Recovery Trajectories (30%). Type B samples simulate failure scenarios to teach error recognition and recovery. We construct a ''what if step $t-1$ failed'' scenario where the input contains the previous screen S_{t-1} (unchanged),

Algorithm 2 Two-Stage Training Pipeline

Require: Base model π_{θ_0}, dataset $\mathcal{D}$, negative ratio r
Ensure: Trained model π_θ
// Stage 1: Robust SFT
$\mathcal{D}_A \leftarrow$ Success trajectories from $\mathcal{D}$
$\mathcal{D}_B \leftarrow$ Generate synthetic failures (ratio r)
$\mathcal{D}_{\text{SFT}} \leftarrow \mathcal{D}_A \cup \mathcal{D}_B$
for each $(x, y^*) \in \mathcal{D}_{\text{SFT}}$ **do**
  $\mathcal{L}_{\text{SFT}} \leftarrow -\log \pi_\theta(y^*|x)$
  Update θ via gradient descent
end for
$\pi_{\theta_1} \leftarrow \pi_\theta$
// Stage 2: GRPO
for each $x \in \mathcal{D}_{\text{SFT}}$ **do**
  Sample G outputs $\{y_1, \ldots, y_G\}$ from π_{θ_1}
  for each y_i **do**
    Parse (T_i, V_i, A_i, E_i) from y_i
    $R_{\text{act},i} \leftarrow \mathbb{I}[A_i = A^*] - \mathbb{I}[A_i \neq A^*]$
    $R_{\text{eff},i} \leftarrow \text{BERTScore}(E_i, E^*)$ if correct
    Determine V_{target} from sample type
    if $V_i = V_{\text{target}}$ **then**
      $R_{\text{ver},i} \leftarrow +1.0$
    else if V_i=SUCCESS, V_{target}=NO_CHANGE **then**
      $R_{\text{ver},i} \leftarrow -2.0$ {Hallucination}
    else
      $R_{\text{ver},i} \leftarrow -0.5$ {Miss}
    end if
    $R_i \leftarrow R_{\text{act}} + \alpha R_{\text{eff}} + \beta R_{\text{ver}}$
  end for
  $\hat{A}_i \leftarrow (R_i - \bar{R})/\sigma_R$
  $\mathcal{L} \leftarrow -\frac{1}{G}\sum_i \hat{A}_i \log \pi_\theta(y_i|x) + \lambda D_{\text{KL}}$
  Update θ
end for
return π_θ

instruction I, and history with an erroneous action A_{t-1}^{err}, while the output follows the NO_CHANGE path with diagnosis and recovery.

Synthetic Failure Modes. The erroneous actions in Type B samples are generated to model realistic GUI failure patterns rather than arbitrary perturbations. We identify five failure categories with empirically assigned sampling weights: (1) **Coordinate offset** (30%): the action targets the correct element type but with slightly misaligned coordinates, simulating tap imprecision or coordinate prediction error; (2) **Action type error** (25%): the agent selects a semantically related but incorrect action type (e.g., long_press instead of click), reflecting ambiguity in action space; (3) **Target misidentification** (20%): the action is directed at the wrong UI element, simulating cases where the agent grounds to a visually similar but incorrect target; (4) **Timing error** (15%): the agent uses a wait or similar temporal action when active interaction is required, reflecting timing uncertainty in asynchronous interfaces; (5) **Null click** (10%): the action registers on an empty or non-interactive screen region, simulating cases where the target UI element has not yet

Category	Metric	Formula	Description
Step-level Metrics *Single-step predictions without simulation*	Type Match (TM)	$\text{TM} = \frac{1}{\lvert\mathcal{D}\rvert}\sum_{i=1}^{\lvert\mathcal{D}\rvert}\mathbb{I}\big[\text{type}(\hat{A}_i) = \text{type}(A_i^*)\big]$	Accuracy of action type prediction (click, scroll, input, etc.).
	Grounding Rate (GR)	$\text{GR} = \frac{1}{\lvert\mathcal{D}\rvert}\sum_i \begin{cases}\mathbb{I}[\hat{c}_i \in \mathcal{B}_i] & \text{sp.}\\ \mathbb{I}[\hat{t}_i = t_i^*] & \text{txt}\end{cases}$	Coordinate-in-bounding-box rate for spatial actions.
	Step Success Rate (SR)	$\text{SR} = \frac{1}{\lvert\mathcal{D}\rvert}\sum_{i=1}^{\lvert\mathcal{D}\rvert}\mathbb{I}\big[d(\hat{A}_i, A_i^*) \le \delta\big]$	Joint correctness of type and parameters.
Task-level Metrics *End-to-end performance under pseudo-online sim.*	Task Success Rate (TSR)	$\text{TSR} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\big[\forall t : \hat{A}_{i,t} = A_{i,t}^*\big]$	Zero-error task success rate. All steps correct on first attempt.
	Progress (PG)	$\text{PG} = \frac{1}{N}\sum_{i=1}^{N}\frac{\lvert\{t : \hat{A}_{i,t} = A_{i,t}^*, \forall t' < t\}\rvert}{T_i}$	Average task progress before first error.
	Simulated Task Success Rate (Sim-TSR)	$\text{Sim-TSR} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}[\text{task } i \text{ done in } \le 2T_i]$	Task success rate allowing error recovery. Key metric for self-correction.
	Avg. Step Overhead (ASO) ↓	$\text{ASO} = \frac{1}{\lvert\mathcal{D}_\text{s}\rvert}\sum_{i\in\mathcal{D}_\text{s}}\left(T_i^\text{act} - T_i^\text{GT}\right)$	Average step overhead for successful tasks. Lower is better.
Robustness Metrics *Error recog. and recovery under failure injection*	Loop Rate (LR) ↓	$\text{LR} = \frac{\lvert\{i : \hat{A}_i \approx A_{\text{err},i}\}\rvert}{\lvert\mathcal{D}_\text{err}\rvert}$	Proportion of cases where agent repeats the failed action.
	Recovery Success Rate (RSR) ↑	$\text{RSR} = \frac{\lvert\{i : \hat{A}_{\text{rec},i} = A_i^*\}\rvert}{\lvert\mathcal{D}_\text{err}\rvert}$	Proportion of cases where the corrective action matches ground-truth.

Table 13: Summary of evaluation metrics. **Notation**: $\mathcal{D}$ = test set; N = number of task trajectories; $\hat{A}$ = predicted action; A^* = ground-truth action; T_i = ground-truth trajectory length; $\mathcal{D}_\text{s}$ = successfully completed tasks; $\mathcal{D}_\text{err}$ = error injection test set.

Parameter	Stage 1 (SFT)	Stage 2 (GRPO)
Learning rate	1×10^{-5}	5×10^{-6}
Batch size	32	16
Epochs	2	15
Max sequence length	8192	8192
Image resolution	smart_resize	smart_resize
Warmup ratio	0.03	0.1
Weight decay	0.01	0.01
GRPO-specific parameters		
Group size G	–	6
KL coefficient λ	–	0.05
α (effect weight)	–	0.5
β (verification weight)	–	0.5

Table 14: Training hyperparameters for both stages. The same configuration is used for both VeriGUI-3B and VeriGUI-7B.

rendered. These weights reflect the relative prevalence of each failure mode observed in empirical analyses of GUI Agent trajectories and ensure that the synthetic failure distribution approximates the real failure distribution encountered in deployment.

Data Generation Pipeline. We use GPT-4o to generate the following components in sequence: (1) given S_{t-1} and A_{t-1}^*, generate a plausible but incorrect action A_{t-1}^err from one of the five categories above; (2) generate a mismatched expected effect corresponding to A_{t-1}^err; (3) generate the correct expected effect for A_{t-1}^*; (4) generate structured CoT reasoning for both SUCCESS and NO_CHANGE paths. All coordinates are converted to relative format (normalized to $[0, 1]$) before storage.

F Prompt Templates

F.1 System Prompt

System Prompt

You are a GUI Agent that controls mobile apps through visual observation and action execution.
Process (in order):

1. **Think:** Analyze the screen, verify previous step, and plan → <think>...</think>
2. **Verification:** State if previous action succeeded → <verification>...</verification>
3. **Action:** Output precise action JSON → <action>{...}</action>
4. **Prediction:** Describe expected screen change → <expected_effect>...</expected_effect>

Available Actions:
{"action": "click/scroll/input_text/long_press, ...}

F.2 Think Format Templates

Think Format: SUCCESS Path

[Verify]	Confirm previous action result
[Recall]	Brief task reminder
[Grounding]	Locate target with visual description
[Coord/Dir/Text]	State action parameters
[Action]	State intended action

Think Format: NO_CHANGE Path (Error Recovery)

[Verify]	Note unchanged screen state
[Diagnose]	Identify why previous action failed
[Recall]	Restate task goal
[Grounding]	Locate correct target
[Coord/Dir/Text]	State corrected parameters
[Recovery]	State recovery action

F.3 Complete Input-Output Examples

Type A Example: Successful Execution

User Instruction: Open CityMapper app, find Location from Eastwood to Chatswood and Get the directions in Bus mode

History (Completed): Step 2: navigate_back | Step 3: click [318, 194] | Step 4: input_text 'Eastwood'

Last Step (Needs Verification): Step 5: Action {"action": "click", "coordinate": [285, 453]} | Expected: ''The search results for Eastwood will disappear, and a map view or route options may appear.''

Current Screen: [Image showing map with route options]

Assistant Output:

<think>
[Verify] Previous click at [285, 453] successfully transitioned from search results to map view showing route options.
[Recall] Task is to find bus directions from Eastwood to Chatswood.
[Grounding] The ''Bus'' mode button is visible near other transport options.
[Coordinate] Element position: [317, 1190].
[Action] click at [317, 1190].
</think>
<verification>SUCCESS</verification>
<action>{"action": "click", "coordinate": [317, 1190]}</action>
<expected_effect>A list of bus directions from Eastwood to Chatswood will appear.</expected_effect>

Type B Example: Error Recovery

User Instruction: I want to listen to Slipping into Relaxed Sleep on the Idanim app.

History (Completed): Step 1: click [418, 735] | Step 2: navigate_back | Step 3: navigate_back

Last Step (Needs Verification): Step 5: Action {"action": "wait", "time": 5} | Expected: ''After waiting, the track will appear and start playing automatically.''

Current Screen: [Image showing same search interface – unchanged]

Assistant Output:

<think>
[Verify] The screen remains unchanged after the wait action; the track did not appear.
[Diagnose] Timing error: the wait action assumed the track would load automatically without user input.
[Recall] The task is to play ''Slipping into Relaxed Sleep'' on the Idanim app.
[Grounding] The search bar is the correct target element.
[Text] The exact text needed is 'Slipping into Relaxed Sleep'.
[Recovery] Execute input_text to search for the track.
</think>
<verification>NO_CHANGE</verification>
<action>{"action": "input_text", "text": "Slipping into Relaxed Sleep"}</action>
<expected_effect>The screen will display the track with playback controls.</expected_effect>

G Case Studies

Case Study 1: Coordinate Misalignment Recovery

Task: Add item to shopping cart in e-commerce app.

Scenario: The agent attempts to click ''Add to Cart'' button at [540, 2100], but the coordinates slightly miss the interactive region.

Baseline Behavior: Repeats click at same coordinates indefinitely, entering infinite loop. After 10 attempts, task fails with no progress.

VeriGUI Behavior:

[Verify]	''Expected cart confirmation, but detail page remains. Click did not register.''
[Diagnose]	''Previous coordinates may have missed the button boundary.''
[Grounding]	''Button appears at approximately [540, 2080].''
[Recovery]	''Retry with adjusted coordinates.''

Result: Successfully adds item on second attempt.

Case Study 2: Network Latency Recovery

Task: Compose and send an email.

Scenario: Agent taps ''Compose'' but network latency prevents compose screen from loading immediately.

Baseline Behavior: Either taps compose again (causing double-compose when screen loads) or attempts to interact with non-existent fields.

VeriGUI Behavior:

[Verify]	''Inbox screen still visible; compose screen did not load.''
[Diagnose]	''Compose button tap did not produce expected result, possibly due to network latency.''
[Recovery]	''Wait briefly and retry compose button.''

Result: Waits 2 seconds, then successfully opens compose screen.