UI-Copilot: Advancing Long-Horizon GUI Automation via Tool-Integrated Policy Optimization

Zhengxi Lu[1], **Fei Tang**[1], **Guangyi Liu**[1], **Kaitao Song**[2], **Xu Tan**[3], **Jin Ma**[3]
Wenqi Zhang[1], **Weiming Lu**[1], **Jun Xiao**[1], **Yueting Zhuang**[1], **Yongliang Shen**[1*]
[1]Zhejiang University [2]Apple [3]Tencent
{zhengxilu, syl}@zju.edu.cn

Abstract

MLLM-based GUI agents have demonstrated strong capabilities in complex user interface interaction tasks. However, long-horizon scenarios remain challenging, as these agents are burdened with tasks beyond their intrinsic capabilities, suffering from memory degradation, progress confusion, and math hallucination. To address these challenges, we present **UI-Copilot**, a collaborative framework where the GUI agent focuses on task execution while a lightweight copilot provides on-demand assistance for memory retrieval and numerical computation. We introduce memory decoupling to separate persistent observations from transient execution context, and train the policy agent to selectively invoke the copilot as `Retriever` or `Calculator` based on task demands. To enable effective tool invocation learning, we propose **T**ool-**I**ntegrated **P**olicy **O**ptimization (**TIPO**), which separately optimizes tool selection through single-turn prediction and task execution through on-policy multi-turn rollouts. Experimental results show that UI-Copilot-7B achieves state-of-the-art performance on challenging MemGUI-Bench, outperforming strong 7B-scale GUI agents such as GUI-Owl-7B and UI-TARS-1.5-7B. Moreover, UI-Copilot-7B delivers a 17.1% absolute improvement on AndroidWorld over the base Qwen model, highlighting UI-Copilot's strong generalization to real-world GUI tasks.

1 Introduction

Graphical User Interface (GUI) agents are designed to interact with digital environments in a human-like manner (Hu et al., 2025; Zhang et al., 2025a; Wang et al., 2025a; Tang et al., 2025b; Liu et al., 2025b). Recent multimodal large language model (MLLM)-based GUI agents (Ye et al., 2025; Gu et al., 2025; Qin et al., 2025), trained via supervised fine-tuning (SFT) and reinforcement learning (RL), have demonstrated strong capability in solving short-horizon tasks, which typically require fewer than 10 interaction steps (Rawles et al., 2024; Zhao et al., 2025; Chen et al., 2026), as shown in Figure 1 and 17).

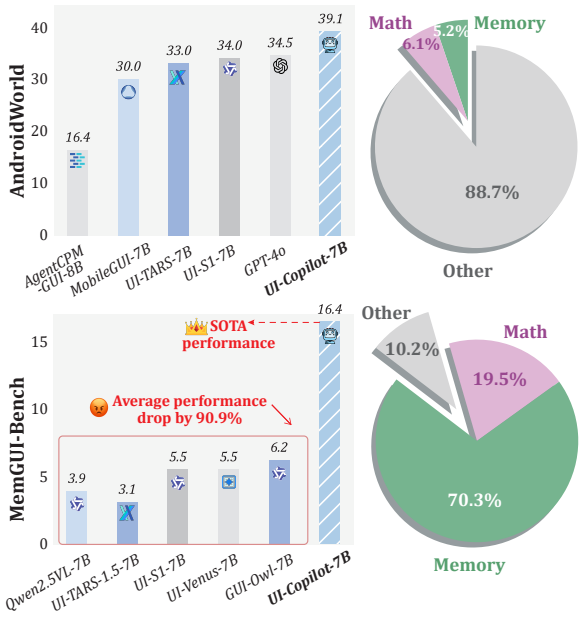


Figure 1: **Left**: Performance on dynamic GUI benchmarks. **Right**: Task distribution of these benchmarks.

However, deploying these agents in long-horizon, memory-intensive scenarios exposes fundamental limitations. As shown in Figure 1, existing 7B models suffer an average performance drop of 90.90% on MemGUI-Bench (Liu et al., 2026). We identify three interconnected challenges underlying these failures (as illustrated in Figure 2): **1) Memory Degradation**: overloaded context causes agents to lose or misrecall critical information from earlier steps; **2) Progress Confusion**: interleaving reasoning traces with action histories obscures task state, leading to redundant actions, disordered subtask execution, or premature termination; and **3) Math Hallucination**: numerical reasoning errors compound as incorrect intermediate results propagate through subsequent computations.

*Corresponding author

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 19741–19762
July 2-7, 2026

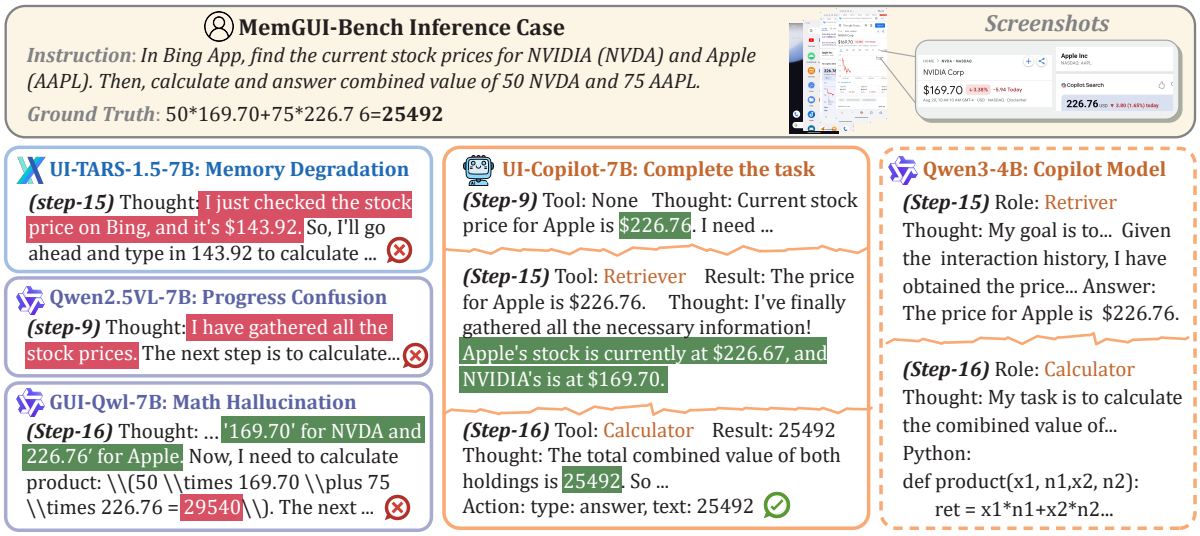


Figure 2: **MemGUI-Bench Inference Case.** Our method successfully completes the task by invoking Copilot Model, whereas other models fail due to memory degradation, progress confusion, and math hallucinations.

Existing approaches address these limitations through multi-agent workflows (Agashe et al., 2025; Wang et al., 2025b, 2024a) or retrieval augmentation (Liu et al., 2025a; Li et al., 2025b; Xu et al., 2025). However, multi-agent workflows rely on predefined pipelines that invoke external modules regardless of actual necessity, resulting in prohibitive inference costs. Retrieval-augmented methods depend heavily on retrieval quality and fail to resolve progress confusion. We attribute these limitations to a shared underlying cause:

> *Agents are burdened with challenges beyond its capabilities, leading to confusion under increasingly overloaded context.*

Our key insight is that ***GUI agents should focus on task execution with lightweight context, while memory and computation are decoupled and invoked only when needed.*** Building on this insight, we introduce **UI-Copilot**, a collaborative framework that decouples persistent observations from transient execution context. Detailed reasoning traces are stored locally while only concise progress summaries remain in the dialogue history, keeping the context window focused and enabling on-demand information retrieval. The policy agent selectively invokes a lightweight copilot model as `Retriever` or `Calculator`, enabling adaptive tool usage that responds to actual task demands.

To enable more effective tool invocation, we propose **T**ool-**I**ntegrated **P**olicy **O**ptimization (**TIPO**), which decouples tool prediction and task execution during training. Tool selection is trained via single-turn supervision, while action generation learns through multi-turn rollouts conditioned on self-generated histories, aligning training dynamics with deployment conditions. Extensive experiments demonstrate that UI-Copilot-7B achieves SOTA performance on MemGUI-Bench, where over 90% of tasks require persistent memory, and attains 39.1% accuracy on AndroidWorld, validating the generalization of the proposed framework. In summary, our contributions are:

- We propose **UI-Copilot**, a collaborative framework where the GUI agent selectively invokes a lightweight copilot for memory retrieval and numerical computation, enabling efficient long-horizon GUI navigation.
- We introduce **memory decoupling** to separate persistent observations from transient context, effectively mitigating context overload.
- We develop **TIPO**, a reinforcement learning algorithm that separately trains GUI agents' tool invocation and action generation.

2 Related Work

2.1 Reinforcement Learning for GUI Agent

Recent advances in multimodal models have catalyzed significant progress in GUI automation (Hu et al., 2025; Zhang et al., 2025a; Wang et al., 2025a; Tang et al., 2025b; Liu et al., 2025b; Ye et al., 2025; Wu et al., 2026). Inspired by DeepSeek-R1 (Guo et al., 2025), recent work (Lu et al., 2025b; Luo et al., 2025a; Qin et al., 2025; Lu et al., 2025a; Gu

et al., 2025; Tang et al., 2025a; Du et al., 2025) has begun applying Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to GUI automation. However, external tool calling for GUI agent training remain un-explored.

2.2 Memory for GUI Agent

Memory remains a fundamental challenge for GUI agents due to the limited context windows. Recent attempts, including multi-agent workflows (Wang et al., 2024a, 2025b; Agashe et al., 2025) and few-shot Retrieval-Augmented Generation (RAG) (Liu et al., 2025a; Li et al., 2025b; Xu et al., 2025), aim to enhance agent performance without model fine-tuning. While effective in certain scenarios, these approaches often suffer from limited scalability and high deployment costs. Additional works incorporate history-aware training mechanisms (Zhou et al., 2025; Liu et al., 2025c; Wang et al., 2025c; Lu et al., 2026), but still face memory degradation for memory-intensive, long-horizon GUI tasks.

3 Method

3.1 UI-Copilot

Problem Definition. We formulate GUI automation as a sequential decision-making problem. Given a task instruction I and initial screen state S_0, the agent generates a sequence of actions $\{a_1, a_2, \ldots, a_T\}$ to complete the task. At each step t, the agent observes the current screenshot state S_t and samples an action a_t from policy $\pi_\theta(a_t|I, S_t, H_t)$, where θ denotes model parameters and H_t represents action history. The action space $\mathcal{A}$ includes coordinate-based operations (`click`, `swipe`, `long_press`), text-based operations (`type`, `answer`) and system-based operations (`system_button`, `open`, `wait`, `terminate`), as shown in Table 3. The environment $\mathcal{E}$ transitions to the next state according to $S_{t+1} = \mathcal{E}(S_t, a_t)$ and rollout continues until task finish or failure.

Rollout Paradigm. Given a policy agent $\mathcal{M}$ (initialized from Qwen2.5VL-7B) and a copilot model $\mathcal{M}_c$ (Qwen3-4B), we define a *tool-integrated multi-turn summary rollout* paradigm for $\mathcal{M}$:

<tool> $\mathcal{T}$: tool call </tool>
↘ call copilot model
<result> $\mathcal{R}$: tool return </result>
<think> thought </think>
<action> a </action>
<summary> summary </summary>

where the prompt is illustrated in Figure 27. The sampling policy is then written as

$$a_t, \texttt{summary}_t, \mathcal{T}_t, \texttt{thought}_t \sim \pi_\theta(\cdot|I, S_t, H_t) \quad (1)$$

where $\mathcal{T}_t \in \{\texttt{Calculator}, \texttt{Retriever}, \texttt{none}\}$ denotes the role assigned to the copilot model $\mathcal{M}_c$ at step t. For memory-intensive tasks, the **`Retriever`** is activated using $\texttt{prompt}_R$ in Figure 28. It takes as input the history knowledge $\mathcal{K}$ (stored as a JSON file), the task instruction I, and progress summaries $\texttt{summary}_{<t}$, and returns tool results to the policy agent in textual form: $\mathcal{R}_t = \mathcal{M}_c(\texttt{prompt}_R, \mathcal{K}, I, \texttt{summary}_{<t})$. For numerical calculation tasks, the **`Calculator`** is invoked with $\texttt{prompt}_C$ in Figure 29 to generate executable Python code. The generated code is then executed by a Python interpreter, and the resulting output is returned to the rollout process: $\mathcal{R}_t = \texttt{PythonExecutor}(\mathcal{M}_c(\texttt{prompt}_C, I, \texttt{summary}_{<t}))$.

Memory Decoupling. Existing agents (Lu et al., 2025c; Ye et al., 2025) maintain the full reasoning content during multi-step rollouts, updating the history as $H_t = H_{t-1} \cup \{a_{t-1}, \texttt{thought}_{t-1}\}$, which we refer to as a **multi-turn context (MC)**. However, it could lead to progress confusion (e.g., redundant steps, disordered sub-task execution, or premature termination) due to content overload, as illustrated in Figure 2 and 9. To mitigate this, we propose a **multi-turn summary (MS)** paradigm that decouples progress tracking from detailed reasoning. In the dialogue history, we maintain only a concise summary reporting current completion status (e.g., *"I have finished sub-task A"*): $H_t = H_{t-1} \cup \{a_{t-1}, \texttt{summary}_{t-1}\}$. The full reasoning content $\texttt{thought}_{t-1}$, which includes the agent's planning and explicit history observations (e.g., *"The stock price is 45 dollars"*), is stored locally in a text file $\mathcal{K}$: $\mathcal{K} = \mathcal{K} \cup \texttt{thought}_{t-1}$. This decoupling reduces context overload in the multi-turn dialogue while preserving detailed information for retrieval or numerical reasoning.

Conclusion. We formalize a *multi-turn summary* interaction paradigm that leverages a copilot model as tools. This offers several advantages: **1) Lightweight Context Window.** By maintaining only concise progress summaries in the dialogue, the agent can focus on task execution, reducing the risk of planning hallucinations. **2) Decoupled Memory.** Detailed observations are stored locally and retrieved on demand, mitigating information

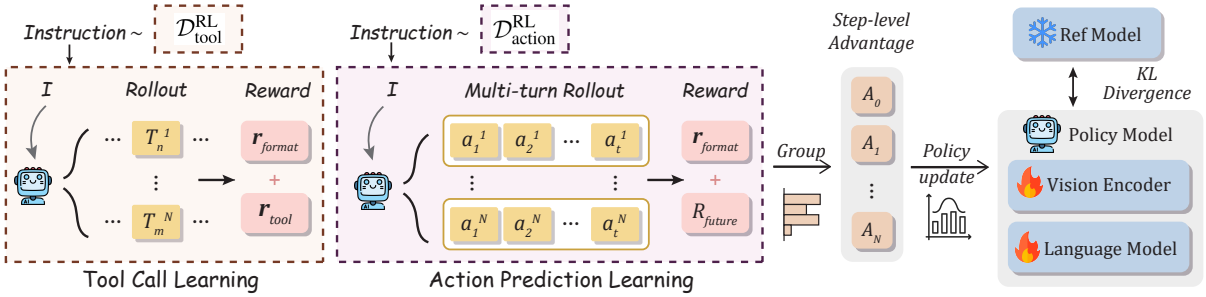


Figure 3: **Overview of TIPO Pipeline.** Policy model jointly learns tool invocations and multi-turn action prediction.

loss and memory hallucination. **3) Efficient Inference.** Unlike previous agent workflows (Wang et al., 2024a), the agent selectively invokes external models, simplifying the execution pipeline.

3.2 Dataset Curation

We collect N diverse, human-annotated trajectories $\tau^* = \{(S_1^*, a_1^*), \ldots, (S_T^*, a_T^*)\}$ from AndroidControl (Li et al., 2024). Further, we use GPT-4o to synthesize tool call content $\texttt{tool}_t^*$, reasoning content $\texttt{thought}_t^*$ and summary $\texttt{summary}_t^*$ for each step t in each trajectory, forming the expert dataset $\mathcal{D}_{\text{expert}} = \{\tau_i^*\}_{i=1}^N$. For tool call learning, we use GPT-4o (Hurst et al., 2024) to form memory-intensive or calculation-needed queries based on $\mathcal{D}_{\text{expert}}$, named as $\mathcal{D}_{\text{tool}}$. Then we merge them as $\mathcal{D}_0 = \mathcal{D}_{\text{expert}} \cup \mathcal{D}_{\text{tool}}$, then randomly split $\mathcal{D}_0$ into $\mathcal{D}^{\text{SFT}}$, $\mathcal{D}_{\text{tool}}^{\text{RL}}$ and $\mathcal{D}_{\text{action}}^{\text{RL}}$ respectively for SFT and RL training. This process is detailed in Figure 4.

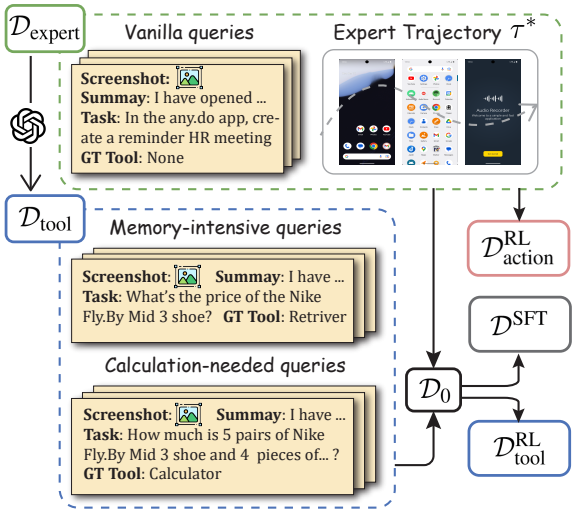


Figure 4: **Training Dataset Curation Pipeline.**

3.3 Tool-Integrated Policy Optimization

Our GUI agent $\mathcal{M}$ is efficiently trained on the pseudo-labeled $\mathcal{D}_0$, during which the copilot model $\mathcal{M}_c$ is not involved.

Cold Start. We initialize the policy via SFT on Qwen2.5VL-7B using $\mathcal{D}^{\text{SFT}}$. Training is performed with a standard cross-entropy loss for next-token prediction, which enables both format learning and behavior cloning from expert trajectories.

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(I,\{S_t\},\{a_t\})\sim\mathcal{D}^{\text{SFT}}} \left[\sum_{t=1}^{T} \log \pi_\theta(a_t \mid I, S_t, H_t)\right] \tag{2}$$

Decoupled Sampling. We model the agentic reasoning process in Equation 7 as explicit tool invocation, then our sampling is written as:

$$P_\theta(\mathcal{T}, a \mid I; \mathbb{T}) = \underbrace{\prod_{t=1}^{T} P_\theta(\mathcal{T}_t \mid H_t, I; \mathbb{T})}_{\text{Tool Calling}} \cdot \underbrace{\prod_{t=1}^{T} P_\theta(a_t \mid a_{<t}, H_t, I; \mathbb{T}))}_{\text{Action Generation}} \tag{3}$$

where $\mathbb{T}$ denotes the set of available tools.

Accordingly, we adopt a decoupled strategy that separates (i) tool-calling rollouts from action-generation rollouts, and (ii) tool-call learning from action learning, depending on the source of instruction I. For $I \sim \mathcal{D}_{\text{tool}}^{\text{RL}}$, we only conduct single-turn tool prediction, conditioned on off-policy history $H_t^* = \texttt{summary}_{<t}^*$. For $I \sim \mathcal{D}_{\text{action}}^{\text{RL}}$, we follow Lu et al. (2025c) and conduct multi-turn action prediction, conditioned on self-generated history $H_t^\pi = \texttt{summary}_{<t}^\pi$. Equation 3 is modified as:

$$P_\theta \approx \begin{cases} \prod_{t=1}^{T} P_\theta(\mathcal{T}_t \mid H_t^*, I; \mathbb{T}), \text{if } I \sim \mathcal{D}_{\text{tool}}^{\text{RL}}, \\ \prod_{t=1}^{T} P_\theta(a_t \mid a_{<t}, H_t^\pi, I), \text{if } I \sim \mathcal{D}_{\text{action}}^{\text{RL}} \end{cases}$$

Tool Call Learning. Unlike agentic reasoning that provides tool feedback from environment, we compute rule-based reward. For i-th rollout,

$$R_t^i = 0.1 \cdot r_{\text{format}} + 0.9 \cdot \mathbb{I}_{[r_{\text{format}}=1]} \cdot r_{\text{tool}} \tag{4}$$

Models	Type	#Cross App								Difficulty Level							
		1 App		*2 App*		*3 App*		*4 App*		*Easy*		*Med*		*Hard*		*Avg*	
		p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3	p@1	p@3
CogAgent	*AO*	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Qwen2.5VL-7B*	*AT*	14.3	17.9	1.8	1.8	0.0	0.0	0.0	0.0	10.4	12.5	0.0	0.0	0.0	0.0	3.9	4.7
UI-Venus-7B	*AT*	21.4	28.6	1.8	1.8	0.0	2.9	0.0	0.0	14.6	20.8	0.0	0.0	0.0	0.0	5.5	7.8
UI-S1-7B*	*MC*	17.9	21.4	3.6	3.6	0.0	0.0	0.0	0.0	12.5	14.6	2.6	2.6	0.0	0.0	5.5	6.2
UI-TARS-1.5-7B	*MC*	14.3	21.4	0.0	1.8	0.0	2.9	0.0	0.0	8.3	16.7	0.0	0.0	0.0	0.0	3.1	6.2
GUI-Owl-7B	*MC*	21.4	35.7	1.8	1.8	2.9	5.9	0.0	0.0	14.6	22.9	0.0	2.4	2.6	2.6	6.2	10.2
UI-Copilot-7B*	*TC*	**42.9**	**50.0**	**12.5**	**16.1**	**5.9**	**8.8**	0.0	**10.0**	**29.2**	**33.3**	**13.2**	**18.4**	**4.8**	**7.1**	**16.4**	**20.3**
Mobile-Agent-V2	*MW*	14.3	17.9	0.0	0.0	0.0	0.0	0.0	0.0	8.3	10.4	0.0	0.0	0.0	0.0	3.1	3.9
SeeAct	*MW*	10.7	25.0	0.0	0	0.0	0	0.0	0	6.2	12.5	0.0	2.4	0.0	0.0	2.3	5.5
AppAgent	*MW*	14.3	42.9	0.0	0.0	0.0	0.0	0.0	0.0	8.3	22.9	0.0	2.4	0.0	0.0	3.1	9.4
Mobile-Agent-E	*MW*	25.0	42.9	0.0	1.8	0.0	0.0	0.0	0.0	12.5	22.9	2.4	2.4	0.0	2.6	5.5	10.2
T3A	*MW*	42.9	60.7	16.1	37.5	23.5	38.2	0.0	30.0	31.2	45.8	16.7	45.2	18.4	34.2	22.7	42.2
M3A	*MW*	46.4	64.3	28.6	41.1	29.4	44.1	30.0	50.0	39.6	47.9	35.7	50.0	21.1	44.7	32.8	47.7
Agent-S2	*MW*	50.0	78.6	19.6	35.7	26.5	52.9	10.0	30.0	41.7	64.6	19.0	42.9	18.4	36.8	27.3	49.2

Table 1: **Results on MemGUI-Bench.** p@k denotes pass@k. Results marked with * are evaluated by ourselves with tool usage, while the remaining results are reported from the benchmark paper, where pass@3 is tested with long-term memory. The best performance in each column is highlighted in **bold**, and the second best is underlined. ***AO*** denotes **A**ction-**O**nly rollout without history. ***AT*** denotes **A**ction-**T**hought history management. ***MC*** denotes **M**ulti-turn **C**ontext. ***TC*** denotes our **T**ool-Integrated multi-turn **C**ontext rollout. ***MW*** denotes **M**ulti-agent **W**orkflow, with Gemini-2.5 Pro (Comanici et al., 2025) as planning agents.

Action Prediction Learning. For multi-turn execution learning, we train on a dataset without tool calling. First, we compute the step-wise reward as:

$$r_t^i = 0.1 \cdot r_{\text{format}} + 0.4 \cdot \mathbb{I}_{[r_{\text{format}}=1]} \cdot r_{\text{type}} + 0.5 \cdot \mathbb{I}_{[r_{\text{format}} \cdot r_{\text{type}}=1]} \cdot r_{\text{acc}} \tag{5}$$

where all the rewards are defined in Appendix B. Then we introduce discounted future reward $R_t^i = \sum_{k=t}^{t_{\text{end}}} \gamma^{,k-t} r_k^i$ and compute the advantage for the individual tokens using the normalized reward R_i: $A_{i,t} = \frac{R_t^i - \text{mean}(\{R_t^i\}_{i=1}^G)}{\text{std}(\{R_t^i\}_{i=1}^G)}$, where G is the total number of samples within a group, which is set as 8. The training objective of TIPO is:

$$\mathcal{J}_{\text{TIPO}}(\theta) = \mathbb{E}_{\substack{I \sim \mathcal{D}^{\text{RL}}, \{o_{i,t}\}_{1,1}^{G,T} \\ \sim \pi_{\theta_{\text{old}}}(\cdot|I)}} \frac{1}{K} \sum_{i=1}^{G} \sum_{t=1}^{T} \sum_{k=1}^{|o_{i,t}|} \min(\rho(\theta) A_{i,t}, \text{clip}(\rho(\theta), 1 \pm \epsilon) A_{i,t})) - \beta \, D_{KL}(\pi_\theta \,\|\, \pi_{\text{ref}}) \tag{6}$$

where $\mathcal{D}^{\text{RL}} = \mathcal{D}_{\text{tool}}^{\text{RL}} \cup \mathcal{D}_{\text{action}}^{\text{RL}}$ denotes the RL dataset, K is the total number of tokens, $\rho(\theta) = \frac{\pi_\theta(o_{i,t,k}|I, o_{i,t,<k})}{\pi_{\theta_{\text{old}}}(o_{i,t,k}|I, o_{i,t,<k})}$ is the importance sampling ratio, and β controls the KL penalty strength. To ensure effective learning, we enforce minimum advantage variance: $\sigma(\{A_{i,t}\}) > \eta$ (η set as 0.3), performing dynamic sampling until this threshold is met.

4 Experiment

4.1 Experiment Setup

Baselines. To comprehensively assess the performance of UI-Copilot, we include three kinds of baselines: (1) advanced proprietary models, including GPT-4o (Hurst et al., 2024) and Claude (Anthropic, 2024); (2) SOTA open-source models, such as AgentCPM-GUI (Zhang et al., 2025b), GUI-Owl (Ye et al., 2025), UI-TARS-1.5 (Qin et al., 2025) and UI-Venus (Gu et al., 2025); (3) multi-agent workflows, such as Mobile-Agent-E (Wang et al., 2025b), Mobile-Agent-V2 (Wang et al., 2024a) and Agent-S2 (Agashe et al., 2025).

Benchmarks. To highlight UI-Copilot-7B's strengths in memory- and math-intensive tasks, we first evaluate all models on the challenging MemGUI-Bench (Liu et al., 2026), which consists of 70.3% memory-intensive and 19.5% math-intensive tasks, with an average of 36 golden steps. We then assess UI-Copilot on the widely used dynamic benchmarks AndroidWorld and MiniWob++ (Rawles et al., 2024) to validate its improvements in multi-turn performance. We additionally include `AC-Real` (referred to as SOP in UI-S1 (Lu et al., 2025c)), reporting both progress (PG) and task success rate (TSR). We also adopt the static GUI navigation benchmarks AndroidControl (Li et al., 2024) and GUI Odyssey (Lu et al., 2024) to evaluate comprehensive GUI understanding under high-level instructions, with action type match accuracy (TM), grounding accuracy rate (GR) and step success rate (SR) reported. To further evaluate the generalization ability of UI-Copilot-7B, we assess grounding and low-level interaction capabilities, as reported in

Models	AC-High			GUI Odyssey			AC-Real		Wob	AW	*Avg*
	TM	*GR*	*SR*	*TM*	*GR*	*SR*	*PG*	*TSR*	*SR*	*SR*	***SR***
Closed-source Models											
Claude-CU (SoM) (Anthropic, 2024)	63.7	0.0	12.5	60.9	0.0	3.1	–	–	–	27.9	–
GPT-4o (SoM) (Hurst et al., 2024)	66.3	0.0	20.8	34.3	0.0	3.3	–	–	**62.0**	34.5	–
Open-source Models											
Qwen2VL-2B (Wang et al., 2024b)	42.3	18.7	13.6	24.4	12.2	12.6	2.0	1.0	20.8	0.0	*7.3*
ShowUI-2B (Lin et al., 2024)	41.8	32.8	19.7	34.8	24.6	21.4	6.8	2.6	27.1	7.0	*11.7*
OS-Genesis-7B (Sun et al., 2024)	65.9	–	44.4	11.7	–	3.6	7.6	3.0	19.8	17.4	*13.4*
OS-Atlas-7B (Wu et al., 2024)	57.4	54.9	29.8	60.4	39.7	27.0	14.3	8.6	35.2	12.1	*18.6*
Qwen2.5VL-3B (Bai et al., 2025)	47.8	46.5	38.9	37.4	26.5	26.7	3.4	1.4	24.1	5.0	*10.2*
Qwen2.5VL-7B (Bai et al., 2025)	62.2	72.5	52.7	67.4	56.3	52.4	17.4	9.8	54.0	22.0	*28.6*
UI-R1-3B (Lu et al., 2025b)	57.9	55.7	45.4	52.2	34.5	32.5	8.4	4.1	26.1	8.2	*12.8*
UI-R1-7B (Lu et al., 2025b)	72.4	62.8	54.2	67.1	41.3	43.5	16.9	10.8	45.2	15.1	*23.7*
AgentCPM-GUI-8B (Zhang et al., 2025b)	77.7	–	69.2	90.8	–	75.0	17.1	10.6	37.8	16.4	*21.6*
UI-S1-7B (Lu et al., 2025c)	79.9	73.4	68.2	76.3	61.7	59.5	**32.4**	**16.3**	60.9	34.0	*37.1*
UI-TARS-7B (Qin et al., 2025)	**83.7**	**80.5**	**72.5**	**94.6**	**90.1**	**87.0**	28.1	14.0	58.7	33.0	*35.2*
Ours Models											
UI-Copilot-3B	64.3	54.5	50.3	52.4	36.8	35.8	15.6	6.9	25.9	15.7	*16.2*
UI-Copilot-7B	82.9	72.2	71.8	74.5	63.8	57.2	31.5	15.8	61.2	**39.1***	***38.7***

Table 2: **Results on Other GUI Benchmarks.** * shows the result with tool calling, and UI-Copilot-7B achieves 32.2% accuracy without tool usage. Wob denotes MiniWob++ and AW denotes AndroidWorld. Average SR is computed as average of AC-Real-TSR, Wob-SR and AW-SR. The highest value is in **bold**, the second is underlined.

Table 4.

4.2 Main Results

Model Comparison. As shown in Table 1, UI-Copilot-7B achieves SOTA performance among 7B models on the challenging MemGUI-Bench. It attains a pass@1 accuracy of 16.4% and a pass@3 accuracy of 20.3%, substantially outperforming strong baselines like GUI-Owl-7B and UI-TARS-1.5-7B, which achieve up to 10.2% accuracy. Moreover, UI-Copilot-7B achieves performance comparable to agentic workflows, including Mobile-Agent-E (5.5%), AppAgent (3.1%), and T3A (22.7%), highlighting the efficiency of our UI-Copilot paradigm. Notably, UI-Copilot-7B successfully solves some ***Hard*** tasks that require over 40 steps or 4 Apps, underscoring its potential for long-horizon, memory-intensive GUI tasks.

Training Effect. Under Tool-Integrated setting, TIPO yields substantial improvements over the base model, *Qwen2.5VL-7B*, demonstrating the effectiveness of our rollout and training strategy. A detailed analysis of the improvements introduced by TIPO is provided in Figure 9.

General Performance. As shown in Table 2, both our 3B and 7B models achieve substantial improvements over their base models on AC-High and GUI Odyssey, demonstrating the effectiveness of TIPO for GUI grounding and high-level understanding. Furthermore, UI-Copilot-7B attains advancing performance among 7B models on dynamic benchmarks such as MiniWob++ (61.2%) and AndroidWorld (39.1%), with results comparable to closed-source models like GPT-4o. Taken together, these results indicate that our model serves as a comprehensive GUI agent, excelling not only in long-horizon tasks but also in general scenarios.

4.3 Training Dynamics

The training dynamics in Figure 5 reveal critical insights. ***[Insight 1] Accuracy dynamics:*** Model accuracy steadily improves as training proceeds, and converges after approximately 40 training steps, indicating sufficient policy optimization. ***[Insight 2] Tool-calling dynamics:*** The frequency of tool in-

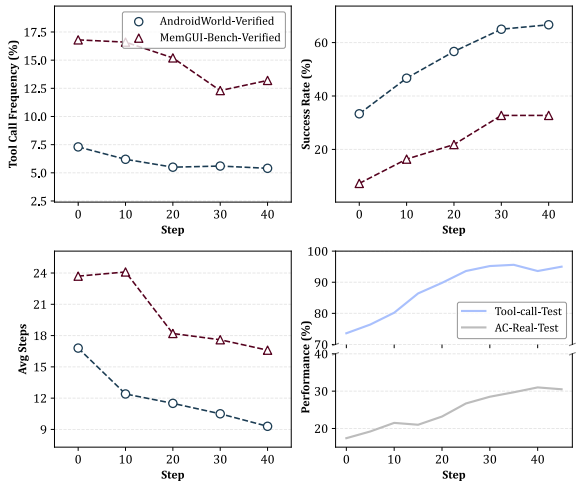


Figure 5: **Training Dynamics** of UI-Copilot-7B on our selected AndroidWorld-Verified (60 tasks), MemGUI-Bench-Verified (55 tasks) subsets, Tool-call-Test (1000 tasks from $\mathcal{D}_0$) and AC-Real-Test (1536 tasks).

Method	$\mathcal{M}_c$		MemGUI-Bench*		AndroidWorld*		Avg	
	Cal	Ret	*Acc(%)*↑	*step*↓	*Acc(%)*↑	*step*↓	*Acc(%)*↑	*step*↓
MW		✓	25.5	33.4	68.3	27.8	46.9	30.6
MW	✓		21.8	35.2	53.3	25.0	37.5	30.1
AT			9.1	20.3	35.0	13.0	22.1	16.6
MC			10.9	19.5	58.3	14.4	34.6	16.9
MS			10.9	18.7	65.0	13.1	38.0	15.9
MS	✓		20.0	19.4	66.7	13.4	43.4	16.4
MS		✓	21.8	20.1	67.3	14.0	44.5	17.1
MS	✓	✓	36.4	19.3	66.7	13.8	51.5	16.6
Copilot Model								
UI-Copilot-7B			23.6	20.8	51.7	15.3	37.7	18.0
Qwen2.5VL-7B			30.9	21.2	53.3	14.2	42.1	17.7
Qwen3-0.6B			27.3	19.9	63.3	13.9	45.3	16.9
Qwen3-1.7B			30.9	19.5	61.7	13.6	46.3	16.6
Qwen3-4B			36.4	19.3	66.7	13.8	51.5	16.6

Figure 6: **Ablation Study on Inference Strategies**. * denotes the verified subset. `Cal` and `Ret` denote `Calculator` and `Retriever`, respectively. All models $\mathcal{M}$ are fine-tuned on $\mathcal{D}^{\text{RL}}$. Multi-agent workflow (MW) includes UI-Copilot-7B and Qwen3-4B. The *step* includes tool invocations.

Tool	AC-Real		Avg	SFT	RL	RL
Acc↑	PG↑	TSR↑	Acc↑		Tool	Action
73.6	17.4	9.88	33.6			
86.4	17.6	10.1	38.0	✓		
94.4	16.8	9.95	40.4	✓	✓	
91.2	28.6	14.1	44.6		✓	On-policy
83.6	32.4	16.5	44.2	✓		On-policy
86.2	22.6	10.3	39.7	✓		Off-policy
95.6	21.5	10.2	42.4	✓	✓	Off-policy
95.0	31.0	16.1	47.4	✓	✓	On-policy
				$\lvert\mathcal{D}^{\text{RL}}_{\text{action}}\rvert : \lvert\mathcal{D}^{\text{RL}}_{\text{tool}}\rvert$		
91.0	30.8	15.8	45.9	200:600		
91.2	31.5	16.3	46.3	300:1000		
95.6	29.8	15.2	46.9	600:1000		
95.0	31.0	16.1	47.4	600:2000		
93.4	31.3	16.2	47.0	600:2400		

Figure 7: **Ablations on Training Paradigms and Dataset.** Tool calling and multi-turn performance are tested on Tool-call-Test (1000 tasks) and AC-Real (1536 tasks). On/Off-policy depends on the history summary.

vocations consistently decreases during training, indicating the model's improving ability to use external tools. Notably, compared to AndroidWorld (approximately 6% tool usage), more complex tasks such as MemGUI-Bench demand significantly higher tool utilization (nearly 13%) and require a longer training phase for tool invocation to stabilize. ***[Insight 3] Execution efficiency dynamics:*** As training progresses, the average execution steps decrease, suggesting that RL effectively reduces redundant actions and mitigates progress confusion, leading to more efficient completion.

4.4 Ablation Analysis

Ablations on Rollout Paradigm. As shown in Figure 6 (upper part), under the setting without tool calling, ours rollout paradigm, **Multi-turn Summary (MS)**, consistently achieves higher accuracy with fewer execution steps than Action–Thought (AT) and Multi-turn Context (MC). This indicates that MS effectively mitigates redundant actions and mitigates progress confusion during multi-turn execution. Based on the MS paradigm, we further conduct tool-set ablations. The results show that both the `Calculator` and `Retriever` contribute to improved performance on MemGUI-Bench. Notably, the full collaborative tool set achieves the best overall performance, with 51.5% average accuracy and 16.6 average steps. It also attains competitive performance against the multi-agent workflow which invokes the copilot model at every step, demonstrating the efficiency of UI-Copilot and TIPO.

Ablations on Copilot Model. We compare different copilot models in Figure 6 (bottom part). Among all models, Qwen3-4B achieves the best performance, outperforming Qwen3-0.6B, Qwen-1.7B, and MLLMs such as Qwen2.5VL-7B. This result highlights the strong capability of Qwen3-4B in context understanding and summarization, which is crucial for effective copilot assistance.

Ablations on TIPO. We conduct ablations on training paradigms (TIPO) in Figure 7 (upper part). The results indicate that SFT plays a critical role as a cold start, providing a reliable initialization for subsequent RL training. Furthermore, both tool call RL and action prediction RL are essential for effective tool calling (see Tool-call-Test) and stable multi-turn execution (see AC-Real), respectively. For action prediction learning, on-policy (multi-turn self-generated) histories consistently outperform off-policy (expert-collected) histories, due to the better alignment with multi-turn evaluation.

Ablations on Training Dataset. Figure 7 (bottom part) demonstrates 600:2000 as an optimal data ratio for ($\lvert\mathcal{D}^{\text{RL}}_{\text{action}}\rvert : \lvert\mathcal{D}^{\text{RL}}_{\text{tool}}\rvert$), which achieves a favorable balance between tool-call learning and action generation learning. Increasing the size of $\mathcal{D}^{\text{RL}}_{\text{action}}$ or $\mathcal{D}^{\text{RL}}_{\text{tool}}$ does not yield further improvements, indicating diminishing returns from additional action-level or tool-level learning samples.

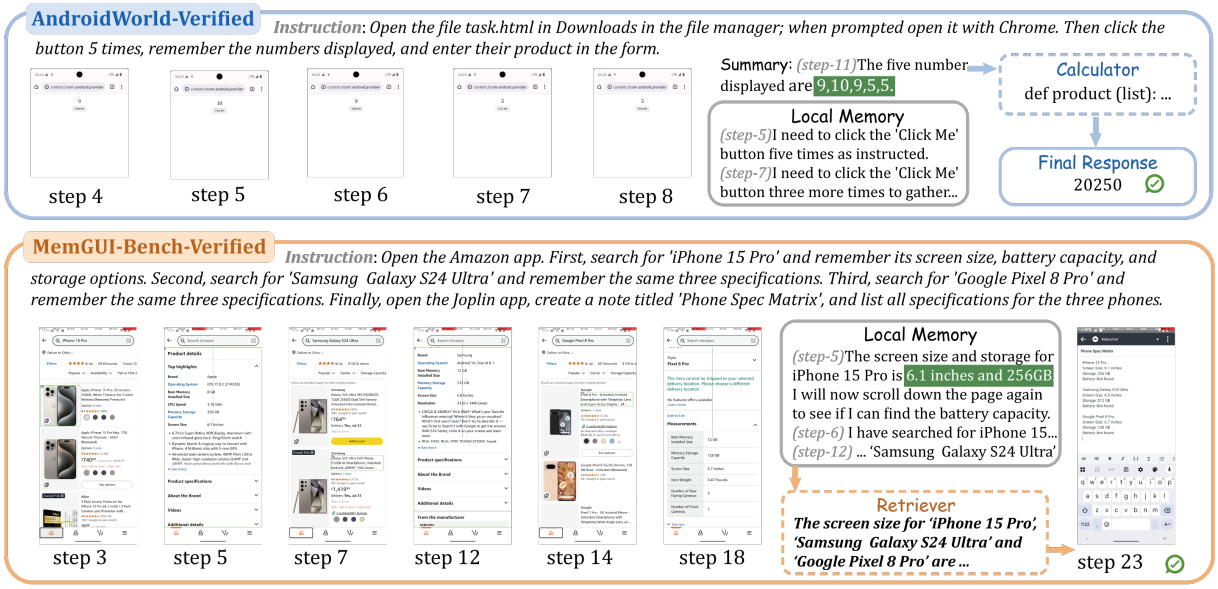


Figure 8: **Case Study.** UI-Copilot-7B successfully completes a math-related task from AndroidWorld (top) and a memory-related task from MemGUI-Bench (bottom).

4.5 Case Study and Analysis

Case Study. Figure 8 presents two challenging cases from AndroidWorld and MemGUI-Bench, which require numerical computation and information retrieval, respectively. In both scenarios, UI-Copilot-7B successfully invokes the Copilot model as the appropriate external tool. Specifically, in the AndroidWorld example, the Calculator generates executable code at step 9 and returns the computed results to UI-Copilot-7B. In the MemGUI-Bench case, the Retriever gathers critical information at steps 5, 12, and 18, enabling UI-Copilot-7B to correctly complete the form-filling task at step 23. Additional successful and failed cases are provided in Appendix F, further demonstrating the effectiveness of our tool-invocation and training method.

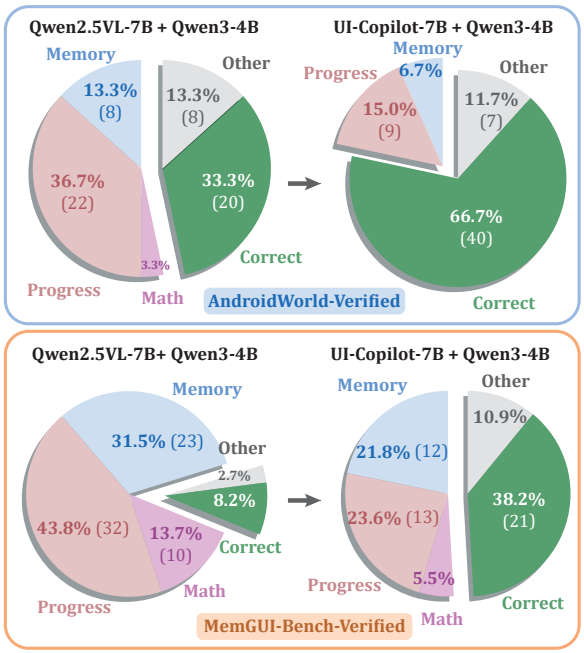


Figure 9: **Error Type Analysis with Tool Usage.** Errors are categorized into Memory Degradation, Progress Confusion, Math Hallucination and Other Fault.

Error Type Analysis. Figure 9 illustrates several representative error types categorized using GPT-4o between TIPO (**MS with tool**) and Qwen2.5VL-7B (**AT with tool**). Among all error categories, *Progress Confusion* emerges as the dominant failure mode on both benchmarks. For the more challenging MemGUI-Bench (only 8.2% success rate orignally), MLLMs further suffer from more reasoning limitations, including *Memory Degradation* and *Math Hallucination*. As results demonstrate, our UI-Copilot-7B achieves substantial improvements over the base Qwen2.5VL-7B (+33.4% on AndroidWorld and +30.0% on MemGUI-Bench). From the error-type perspective, *Progress Confusion* is almost halved, while *Memory Degradation* and *Math Hallucination* are significantly mitigated, demonstrating the effectiveness of our TIPO. Error analysis of tool invocation is shown in Figure 20.

5 Conclusion

We incorporate Copilot Model as external tools and propose TIPO for tool-integrated learning, aiming to solve complex and long-horizon GUI tasks. Our UI-Copilot-7B achieves consistently strong performance across several challenging GUI benchmarks.

Limitations

Currently, our tool set is limited to Calculator and Retriever. However, real-world GUI scenarios often require a broader spectrum of tools, such as web search and visual cropping. Extending our framework to support more diverse tool integrations remains an important direction for future work.

Acknowledgement

This work was supported by National Natural Science Foundation of China (No. 62436007), National Natural Science Foundation of China (No. 62506332), "Pioneer" and "Leading Goose" R&D Program of Zhejiang (NO. 2026C02A1223), and CCF-Tencent Rhino-Bird Open Research Fund.

References

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. Agent s2: A compositional generalist-specialist framework for computer use agents. *Preprint*, arXiv:2504.00906.

Anthropic. 2024. Developing a computer use model. https://www.anthropic.com/news/developing-computer-use.

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*.

Tongbo Chen, Zhengxi Lu, Zhan Xu, Guocheng Shao, Shaohan Zhao, Fei Tang, Yong Du, Kaitao Song, Yizhou Liu, Yuchen Yan, Wenqi Zhang, Xu Tan, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. 2026. Knowu-bench: Towards interactive, proactive, and personalized mobile agent evaluation. *Preprint*, arXiv:2604.08455.

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*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, and 1 others. 2025. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*.

Yong Du, Yuchen Yan, Fei Tang, Zhengxi Lu, Chang Zong, Weiming Lu, Shengpei Jiang, and Yongliang Shen. 2025. Test-time reinforcement learning for gui grounding via region consistency. *arXiv preprint arXiv:2508.05615*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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*.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, and 1 others. 2025. Os agents: A survey on mllm-based agents for general computing devices use. *arXiv preprint arXiv:2508.04482*.

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*.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025a. Screenspot-pro: Gui grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*.

Runze Li, Yuwen Zhai, Bo Xu, LiWu Xu, Nian Shi, Wei Zhang, Ran Lin, and Liang Wang. 2025b. Echotrail-gui: Building actionable memory for gui agents via critic-guided self-exploration. *arXiv preprint arXiv:2512.19396*.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on computer control agents. *arXiv e-prints*, pages arXiv–2406.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025c. Torl: Scaling tool-integrated RL. *CoRR*, abs/2503.23383.

Shuquan Lian, Yuhang Wu, Jia Ma, Yifan Ding, Zihan Song, Bingqi Chen, Xiawu Zheng, and Hui Li. 2025. Ui-agile: Advancing gui agents with effective

reinforcement learning and precise inference-time grounding. *arXiv preprint arXiv:2507.22025*.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2024. Showui: One vision-language-action model for gui visual agent.

Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Qinyi Luo, Shunye Tang, Yuxiang Chai, Weifeng Lin, Han Xiao, WenHao Wang, Siheng Chen, and 1 others. 2026. Memgui-bench: Benchmarking memory of mobile gui agents in dynamic environments. *arXiv preprint arXiv:2602.06075*.

Guangyi Liu, Pengxiang Zhao, Liang Liu, Zhiming Chen, Yuxiang Chai, Shuai Ren, Hao Wang, Shibo He, and Wenchao Meng. 2025a. Learnact: Few-shot mobile gui agent with a unified demonstration benchmark. *arXiv preprint arXiv:2504.13805*.

Guangyi Liu, Pengxiang Zhao, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, Hao Wang, and 1 others. 2025b. Llm-powered gui agents in phone automation: Surveying progress and prospects. *arXiv preprint arXiv:2504.19838*.

Tao Liu, Chongyu Wang, Rongjie Li, Yingchen Yu, Xuming He, and Bai Song. 2025c. Gui-rise: Structured reasoning and history summarization for gui navigation. *arXiv preprint arXiv:2510.27210*.

Zikang Liu, Junyi Li, Wayne Xin Zhao, Dawei Gao, Yaliang Li, and Ji-rong Wen. 2025d. Pal-ui: Planning with active look-back for vision-based gui agents. *arXiv preprint arXiv:2510.00413*.

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*.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. 2026. Skill0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*.

Zhengxi Lu, Jiabo Ye, Fei Tang, Yongliang Shen, Haiyang Xu, Ziwei Zheng, Weiming Lu, Ming Yan, Fei Huang, Jun Xiao, and 1 others. 2025c. Ui-s1: Advancing gui automation via semi-online reinforcement learning. *arXiv preprint arXiv:2509.11543*.

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

Zhihao Luo, Wentao Yan, Jingyu Gong, Min Wang, Zhizhong Zhang, Xuhong Wang, Yuan Xie, and Xin Tan. 2025b. Navimaster: Learning a unified policy for gui and embodied navigation tasks. *arXiv preprint arXiv:2508.02046*.

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*.

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*.

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

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, and 1 others. 2024. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*.

Fei Tang, Zhangxuan Gu, Zhengxi Lu, Xuyang Liu, Shuheng Shen, Changhua Meng, Wen Wang, Wenqi Zhang, Yongliang Shen, Weiming Lu, and 1 others. 2025a. Gui-g^2: Gaussian reward modeling for gui grounding. *arXiv preprint arXiv:2507.15846*.

Fei Tang, Haolei Xu, Hang Zhang, Siqi Chen, Xingyu Wu, Yongliang Shen, Wenqi Zhang, Guiyang Hou, Zeqi Tan, Yuchen Yan, and 1 others. 2025b. A survey on (m) llm-based gui agents. *arXiv preprint arXiv:2504.13865*.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024a. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, and 1 others. 2024b. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo,

Yiheng Xu, Chen Henry Wu, and 1 others. 2025a. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*.

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. *arXiv preprint arXiv:2501.11733*.

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

Jinyang Wu, Shuo Yang, Changpeng Yang, Yuhao Shen, Shuai Zhang, Zhengqi Wen, and Jianhua Tao. 2026. Spark: Strategic policy-aware exploration via dynamic branching for long-horizon agentic learning. *arXiv preprint arXiv:2601.20209*.

Junde Wu, Jiayuan Zhu, and Yuyuan Liu. 2025. Agentic reasoning: Reasoning llms with tools for the deep research. *arXiv preprint arXiv:2502.04644*.

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*.

Ran Xu, Kaixin Ma, Wenhao Yu, Hongming Zhang, Joyce C Ho, Carl Yang, and Dong Yu. 2025. Retrieval-augmented gui agents with generative guidelines. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17877–17886.

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

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

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.

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*.

Pengxiang Zhao, Guangyi Liu, Yaozhen Liang, Weiqing He, Zhengxi Lu, Yuehao Huang, Yaxuan Guo, Kexin Zhang, Hao Wang, Liang Liu, and 1 others. 2025. Mas-bench: A unified benchmark for shortcut-augmented hybrid mobile gui agents. *arXiv preprint arXiv:2509.06477*.

Xurui Zhou, Gongwei Chen, Yuquan Xie, Zaijing Li, Kaiwen Zhou, Shuai Wang, Shuo Yang, Zhuotao Tian, and Rui Shao. 2025. Hiconagent: History context-aware policy optimization for gui agents. *arXiv preprint arXiv:2512.01763*.

A Action Space

Action Type	Description
`click`	Tap a specific coordinate (x, y) on the screen.
`long_press`	Press and hold at (x, y) for a specified duration.
`swipe`	Perform a swipe gesture from (x_1, y_1) to (x_2, y_2).
`answer`	Output a textual answer to the task.
`type`	Enter text into the currently focused input field.
`system_button`	Trigger a system-level button (e.g., Home, Back).
`open`	Launch an APP on the device.
`wait`	Pause execution for a given number of seconds to allow UI changes.
`terminate`	Stop execution and report task success or failure.

Table 3: Action space in AndroidWorld automation.

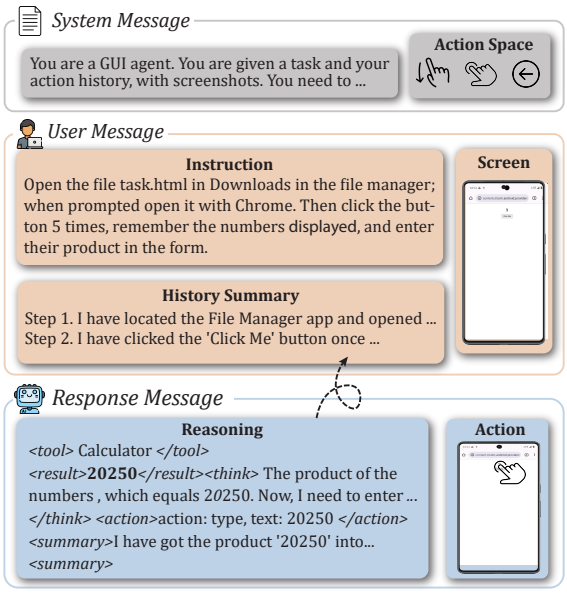


Figure 10: **Interaction example for UI-Copilot-7B.**

B Reward Definition

B.1 Type Reward (r_{type})

$r_{\text{type}} \in \{0, 1\}$ indicates whether the predicted action type matches the ground-truth action type. Let a^{pred} and a^{gt} denote the predicted and ground-truth action types, respectively. The type reward is defined as $r_{\text{type}} = \mathbb{I}[a^{\text{pred}} = a^{\text{gt}}]$.

B.2 Accuracy Reward (r_{acc})

$r_{\text{acc}} \in \{0, 1\}$ evaluates whether the predicted action is accurate given the ground-truth action, conditioned on the action type being correct and after coordinate normalization. Let $\mathbf{p}^{\text{pred}}$ and $\mathbf{p}^{\text{gt}}$ denote the predicted and ground-truth coordinates, respectively.

- **Wait / Terminate**

  The prediction is accurate if the action type exactly matches: $r_{\text{acc}} = \mathbb{I}[a^{\text{pred}} = a^{\text{gt}}]$.

- **System Button**

  Let $\text{button}^{\text{pred}}$ and $\text{button}^{\text{gt}}$ denote the predicted and ground-truth system button names. The prediction is accurate if the button names match in a case-insensitive manner: $r_{\text{acc}} = \mathbb{I}[\text{button}^{\text{pred}} = \text{button}^{\text{gt}}]$.

- **Type / Answer / Key / Open**

  Let $\text{text}^{\text{pred}}$ and text^{gt} denote the predicted and ground-truth input strings. The prediction is accurate if the texts match under relaxed string matching: $r_{\text{acc}} = \mathbb{I}[\text{text}^{\text{pred}} \sim \text{text}^{\text{gt}}]$.

- **Swipe**

  Let $\mathbf{p}_1^{\text{pred}}, \mathbf{p}_2^{\text{pred}}$ denote the start and end points of the predicted swipe, and $\text{dir}(\cdot, \cdot)$ be the function that infers swipe direction. The prediction is accurate if the inferred swipe direction matches the ground truth: $r_{\text{acc}} = \mathbb{I}[\text{dir}(\mathbf{p}_1^{\text{pred}}, \mathbf{p}_2^{\text{pred}}) = \text{dir}^{\text{gt}}]$.

- **Click / Long Press**

  Let $\mathcal{B}^{\text{gt}}$ denote the enlarged ground-truth bounding box and ϵ be a fixed distance threshold. The prediction is accurate if the predicted point falls inside the bounding box or is sufficiently close to the ground-truth point: $r_{\text{acc}} = \mathbb{I}[\mathbf{p}^{\text{pred}} \in \mathcal{B}^{\text{gt}} \ \vee \ \|\mathbf{p}^{\text{pred}} - \mathbf{p}^{\text{gt}}\|_2 \leq \epsilon]$.

C Theoretical Analysis

C.1 Preliminaries: Agentic RL

Agentic RL incorporates tool-call feedback during the reasoning process (Gou et al., 2024; Li et al., 2025c; Wu et al., 2025; Dong et al., 2025). The rollout sampling can be decomposed as:

$$P_\theta(\mathcal{R}, a \mid I; \mathbb{T}) = \underbrace{\prod_{t=1}^{t_\mathcal{R}} P_\theta(\mathcal{R}_t \mid \mathcal{R}_{<t}, I; \mathbb{T})}_{\text{Agentic Reasoning}} \cdot \underbrace{\prod_{t=1}^{t_a} P_\theta(a_t \mid a_{<t}, \mathcal{R}, I; \mathbb{T}))}_{\text{Action Generation}}, \tag{7}$$

where $\mathbb{T}$ denotes the set of available tools, $\mathcal{R}$ is the reasoning trajectory of length $t_\mathcal{R}$, interleaved with tool-call feedback, and a is the performed action with length t_a.

C.2 Multi-turn Action Prediction Learning

We prove in this section that why we use on-policy multi-turn rollout for action prediction learning.

Setup. Let $\pi(\cdot \mid \theta)$ denote the training policy and $\mu(\cdot \mid \theta)$ the rollout (inference) policy. Given a high-level instruction I, the deployment objective is

$$\mathcal{J}(\theta) = \mathbb{E}_{I\sim p_\mathcal{I}}\Big[\mathbb{E}_{(a_{1:T},\mathcal{T}_{1:T})\sim\mu(\cdot|I)}\big[R(I, a_{1:T}, \mathcal{T}_{1:T})\big]\Big].$$

Single-turn Training Mismatch. In single-turn (ST) training, each step conditions on off-policy histories H_t^*, yielding

$$\widehat{\nabla_\theta \mathcal{J}}_{\mathrm{ST}} = \mathbb{E}_{I\sim p_\mathcal{I}, a_t\sim\pi(\cdot|I,S_t,H_t^*)} \Big[\nabla_\theta \log \pi(a_t \mid I, S_t, H_t^*)\, R(I, a_{1:T}, \mathcal{T}_{1:T})\Big].$$

Evaluation, however, uses self-generated histories H_t^π: $a_t^\pi \sim \mu(\cdot \mid I, S_t, H_t^{\mathrm{MT}})$, so that

$$\widehat{\nabla_\theta \mathcal{J}}_{\mathrm{ST}} \neq \nabla_\theta \mathbb{E}_{(a_{1:T}^\pi,\mathcal{T}_{1:T}^\pi)\sim\mu}[R(I, a_{1:T}^\pi, \mathcal{T}_{1:T}^\pi)].$$

Equivalently,

$$\arg\max_\theta \mathbb{E}_{a_{1:T}\sim\pi}[R(I, a_{1:T})] \;\neq\; \arg\max_\theta \mathbb{E}_{a_{1:T}^\pi\sim\mu}[R(I, a_{1:T}^\pi)]$$

which illustrates the biased gradient and deployment gap.

Multi-turn Training Alignment. Multi-turn (MT) training conditions on self-generated histories H_t^π at each step, producing full trajectories $(a_{1:T}^\pi, \mathcal{T}_{1:T}^\pi) \sim \mu(\cdot \mid I)$ and the gradient estimator

$$\widehat{\nabla_\theta \mathcal{J}}_{\mathrm{MT}} = \mathbb{E}_{I\sim p_\mathcal{I}}\Big[\sum_{t=1}^{T} \nabla_\theta \log \mu(a_t^\pi \mid I, S_t, H_t^\pi)\, R(I, a_{1:T}^\pi, \mathcal{T}_{1:T}^\pi)\Big].$$

By aligning training histories with rollout histories, MT training better approximates the deployment objective:

$$\arg\max_\theta \mathbb{E}_{(a_{1:T}^\pi,\mathcal{T}_{1:T}^\pi)\sim\mu}[R(I, a_{1:T}^\pi)] \approx \arg\max_\theta \mathcal{J}(\theta)$$

reducing the train–inference mismatch.

Conclusion. Single-turn training suffers from biased gradients due to off-policy histories H_t^*, whereas multi-turn training uses self-generated histories H_t^π, leading to more consistent and stable optimization toward deployment-time performance.

C.3 Decoupled Sampling in RL Training

We consider the policy gradient

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}\Big[A(I, \mathcal{T}, a)\,\big(\nabla_\theta \log P_\theta(\mathcal{T} \mid I) + \nabla_\theta \log P_\theta(a \mid \mathcal{T}, I)\big)\Big].$$

where $A(\cdot)$ denotes the advantage function.

Tool-learning instructions. For $I \sim \mathcal{D}_{\mathrm{tool}}^{\mathrm{RL}}$, tool calls are supervised and consistent across trajectories. Conditioned on a fixed tool $\mathcal{T}$, the action distribution $P_\theta(a \mid \mathcal{T}, I)$ becomes highly concentrated, yielding a small action-level advantage:

$$\mathbb{E}_{a\sim P_\theta(\cdot|\mathcal{T},I)}\big[|A(I, \mathcal{T}, a)|\big] \approx 0.$$

As a result, the action-related gradient term contributes negligibly and can be ignored when estimating the policy gradient.

Progress-learning instructions. For $I \sim \mathcal{D}_{\mathrm{action}}^{\mathrm{RL}}$, no tool is required and we explicitly fix the tool set to $\mathcal{T} = \mathsf{None}$ in the prompt. In this case, the policy reduces to action generation only, and the gradient simplifies to

$$\nabla_\theta \mathcal{J}_{\mathrm{prog}}(\theta) = \mathbb{E}[A(I, \mathsf{None}, a)\, \nabla_\theta \log P_\theta(a \mid I)]$$

where rewards and advantages are computed solely based on task progress.

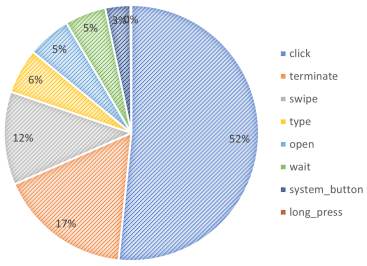


Figure 11: Action Type Distribution of $\mathcal{D}^{\text{RL}}_{\text{action}}$.

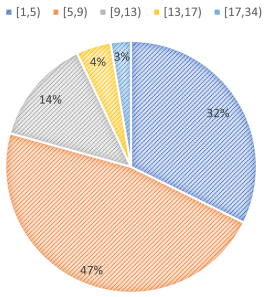


Figure 12: Trajectory Length Distribution of $\mathcal{D}^{\text{RL}}_{\text{action}}$.

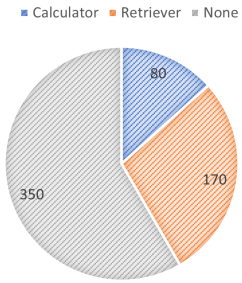


Figure 13: Tool Type Distribution of $\mathcal{D}^{\text{RL}}_{\text{tool}}$.

D Data Description

D.1 Training Dataset

$\mathcal{D}^{\text{RL}}_{\text{action}}$ Composition. Figures 11 and 12 illustrate the distributions of action types and trajectory lengths across 2000 training trajectories in $\mathcal{D}^{\text{RL}}_{\text{action}}$, respectively. Among action types, `CLICK` is the most prevalent, followed by `TERMINATE`, which consistently serves as the final action in all successfully completed trajectories, and `SWIPE`. In terms of trajectory length, most trajectories comprise between 5 and 9 interaction steps. We provide some cases below,

```
Open the Zoho Meet app, view the
↪ scheduled meetings.
Go to the mobile category after closing
↪ the pop up and browse the products.
Check out all of the suggested products
↪ and compare pricing because I'm
↪ looking for a budget friendly sofa.
I want to view the recipe for Welsh
↪ Cakes in the kitchen stories app.
Open the Yahoo Mail App, Select the
↪ Starva Mail and Unmark an Email as
↪ read.
If I lose connection to the internet, I
↪ want to make the agents.txt file
↪ accessible offline in Google Drive
↪ so that I can readily access it.
I want to share "Oscar and the wolf
↪ -somebody wants" music to my friend
↪ karin.iversen@example.com via gmail.
In the HealthifyMe app, view your today
↪ activity
Add paintings to cart on the Rtistiq
↪ app.
Set 10 minutes before notification for
↪ the birthday event in gmail calendar
Im going to Deutsches Museum from Ulm
↪ city, so get the traffic update on
↪ the route of the Deutsches Museum
↪ from my location.
I want to search for some interesting
↪ activities in Hawaii.
Open the TickTick app and mark microsoft
↪ training update classes as complete.
I want to read the reviews of the Nike
↪ Fly.By Mid 3 shoe in the Nike app.
Search for Kayak mail in gmail app.
Use the gmail address
↪ karin.iversen@example.com to send
↪ Karin information about the bus that
↪ leaves at 1:55 a.m.
In the Simple Habit app, In order to
↪ improve my meditation, I would like
↪ to listen to the sound of ocean.
```

$\mathcal{D}^{\text{RL}}_{\text{tool}}$ Composition. The dataset $\mathcal{D}^{\text{RL}}_{\text{tool}}$ is constructed using GPT-4o and includes 170 memory-intensive queries, 80 calculation-required queries, and 350 tasks requiring no tool usage, as illustrated in Figure 13. We also provide some cases below,

```
How much would it cost to buy two pairs
↪ of Nike Fly.By Mid 3 shoes, and
↪ which option is more budget-friendly
↪ compared to similar models?
Among the suggested sofas, which one is
↪ the most budget-friendly based on
↪ their prices and features?
What is the total price after adding the
↪ Rockrider City Cycle Btwin bicycle
↪ to my cart, including any applicable
↪ discounts?
What is the total cost of all selected
↪ paintings added to the cart in the
↪ Rtistiq app?
```

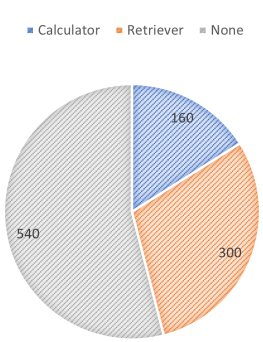


Figure 14: Tool Type Distribution of Tool-call-Test.

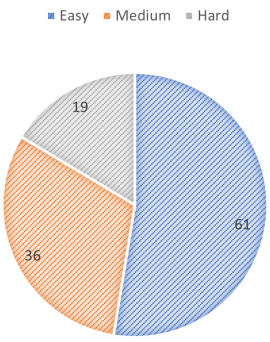


Figure 15: Difficulty Level Distribution of AndroidWorld.

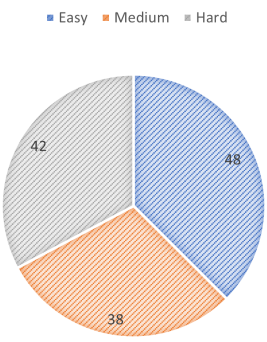


Figure 16: Difficulty Level Distribution of MemGUI-Bench.

```
What is the cheapest available flight
↪ from Knoxville to Hawaii, and how do
↪ prices vary across different dates?
What is the estimated travel time and
↪ cost when traveling from Los Angeles
↪ to Oakland by train?
What are the available bus options from
↪ Amsterdam Centraal to Rotterdam
↪ Centraal on October 26, and how long
↪ does each trip take?
How long will it take to travel from Ulm
↪ to the Deutsches Museum based on
↪ current traffic conditions?
How much earlier will I be notified if I
↪ set the calendar reminder to 10
↪ minutes before the birthday event?
Which file was made available offline in
↪ Google Drive when preparing for loss
↪ of internet connection?
Which music track was shared via Gmail,
↪ and to which recipient was it sent?
Which bus departure information was sent
↪ to karin.iversen@example.com?
What route and traffic conditions were
↪ shown for traveling from Ulm to the
↪ Deutsches Museum?
Which activities were shown after
↪ searching for things to do in
↪ Hawaii?
Which Microsoft training classes were
↪ marked as complete in the TickTick
↪ app?
What account or session status was shown
↪ after signing out of the Babbel app?
```

Training Details. We train Qwen2.5VL-7B with TIPO for 50 steps on 8 A100 GPUs. Each batch samples 16 prompts, with 8 rollouts per prompt. We define the maximum response length as 12288 tokens and learning rate as 1×10^{-6}.

D.2 Evaluation Dataset

Tool-call-Test. As shown in Figure 14, the Tool-call-Test subset consists of 1000 tasks generated using GPT-4o and is carefully aligned with $\mathcal{D}_{\text{tool}}^{\text{RL}}$ in terms of task-type distribution.

AndroidWorld vs MemGUI-Bench. As illustrated in Figures 15 and 16, more than half of the tasks in AndroidWorld are categorized as ***Easy***. In contrast, MemGUI-Bench contains a substantially larger proportion of challenging tasks, with the distribution of ***Easy***, ***Medium***, and ***Hard*** tasks being nearly uniform (approximately 1:1:1). Furthermore, MemGUI-Bench exhibits a much longer average trajectory length, requiring 36.2 optimal steps on average, which significantly exceeds that of AndroidWorld (8.4 steps), as shown in Figure 17. Despite the increased task difficulty and longer interaction horizons, our UI-Copilot-7B achieves an accuracy of 20.3% on MemGUI-Bench and 39.1% on AndroidWorld, demonstrating robust and relatively balanced performance across benchmarks of varying complexity.

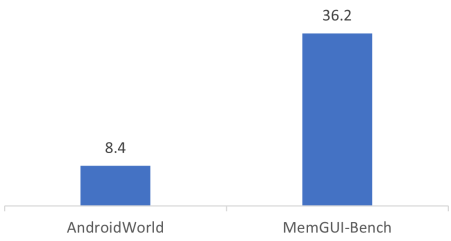


Figure 17: Golden steps comparison between AndroidWorld and MemGUI-Bench.

Models	ScreenSpot			AC-Low			AC-High			GUI Odyssey		
	V2	*Pro*	*Avg*	*TM*	*GR*	*SR*	*TM*	*GR*	*SR*	*TM*	*GR*	*SR*
Closed-source Models												
GPT-4o (Hurst et al., 2024)	18.3	0.8	9.6	74.3	0.0	19.4	66.3	0.0	20.8	34.3	0.0	3.3
Claude-CU (Anthropic, 2024)	83.0	17.1	50.1	74.3	0.0	19.4	63.7	0.0	12.5	60.9	0.0	3.1
Open-source Models												
OS-Atlas-4B (Wu et al., 2024)	71.9	3.7	37.8	91.9	83.8	80.6	49.0	49.5	22.8	49.6	34.6	20.3
OS-Atlas-7B (Wu et al., 2024)	84.1	18.9	51.5	93.6	88.0	85.2	57.4	54.9	29.8	60.4	39.7	27.0
Qwen2.5VL-3B (Bai et al., 2025)	80.9	28.7	54.8	62.0	74.1	59.3	47.8	46.5	38.9	37.4	26.5	26.7
Qwen2.5VL-7B (Bai et al., 2025)	89.0	28.7	58.9	83.4	87.0	62.5	62.2	72.5	52.7	67.4	56.3	52.4
SeeClick (Cheng et al., 2024)	55.1	1.1	28.1	93.0	73.4	75.0	82.9	62.9	59.1	71.0	52.4	53.9
UI-R1-3B (Lu et al., 2025b)	85.4	17.8	51.6	79.2	82.4	66.4	57.9	55.7	45.4	52.2	34.5	32.5
UI-R1-7B (Lu et al., 2025b)	90.0	33.5	61.8	86.6	83.7	69.7	72.4	62.8	54.2	67.1	41.3	43.5
GUI-R1-3B (Luo et al., 2025a)	85.0	28.6	56.8	83.7	81.6	64.4	58.0	56.2	46.6	54.8	41.5	41.3
GUI-R1-7B (Luo et al., 2025a)	88.2	31.3	59.8	85.2	85.4	66.5	71.6	65.6	51.7	65.5	43.6	38.8
OS-Genesis-7B (Sun et al., 2024)	–	–	–	90.7	–	74.2	65.9	–	44.4	11.7	–	3.6
Aguvis-7B (Xu et al., 2024)	81.8	22.9	52.4	93.8	–	89.4	65.6	–	54.2	26.7	–	13.5
NaviMaster-7B (Luo et al., 2025b)	–	–	–	–	93.9	69.5	72.9	–	54.0	64.4	–	36.9
PAL-UI-3B (Liu et al., 2025d)	–	–	–	–	–	–	60.4	58.7	49.3	56.7	36.9	34.6
PAL-UI-7B (Liu et al., 2025d)	–	–	–	–	–	–	71.3	70.5	57.8	65.1	46.8	41.7
UI-AGILE-3B (Lian et al., 2025)	88.6	37.9	63.3	85.4	87.6	74.3	78.6	60.7	56.8	–	–	–
UI-AGILE-7B (Lian et al., 2025)	92.1	44.0	68.1	87.7	88.1	77.6	80.1	61.9	60.6	–	–	37.0
UI-S1-7B (Lu et al., 2025c)	90.1	30.6	60.4	92.2	89.3	89.2	79.9	73.4	68.2	76.3	61.7	59.5
AgentCPM-GUI-8B (Zhang et al., 2025b)	–	–	–	94.4	–	90.2	77.7	–	69.2	90.8	–	75.0
UI-TARS-7B (Qin et al., 2025)	91.6	35.7	63.7	95.2	89.3	91.8	83.7	80.5	72.5	94.6	90.1	87.0
Ours 7B Models												
UI-Copilot-7B	90.0	31.6	60.8	93.6	88.2	89.2	82.9	72.2	71.8	74.5	63.8	57.2

Table 4: **Model Comparison on Single-turn Benchmarks.**

E Supplementary Results

Single-turn Benchmarks Single-turn tasks evaluate the grounding capability and GUI Understanding capability of the end-to-end GUI model in conversations without historical context. We use ScreenSpot-V2 (Cheng et al., 2024) and ScreenSpot-Pro (Li et al., 2025a) to evaluate the grounding ability. We also adopt AndroidControl-Low, AndroidControl-High (Li et al., 2024) and GUI Odyssey (Lu et al., 2024), for comprehensive GUI understanding evaluation. The action type match accuracy (TM), grounding accuracy rate (GR) and step success rate (SR) are reported.

Single-turn Performance. Table 4 demonstrates that UI-Copilot-7B maintains competitive single-turn performance. Compared to the base model, UI-Copilot-7B achieves consistent improvements, with gains of +19.1% on AC-High SR and +4.8% on GUI Odyssey SR. Notably, although models trained with single-turn RL (e.g., AgentCPM-GUI-8B) excel on single-turn tasks, they struggle with multi-turn execution (only 16.4% on AndroidWorld). This performance gap can be attributed to two primary factors: **(1) a mismatch between the training and the evaluation dynamics**, particularly regarding whether the the historical context is on-policy or not (see Appendix C); and **(2) overfitting to local reward signals**, leading to ignorance of global training objectives.

Pass@k Validation. We evaluate UI-Copilot-7B 's `pass@k` accuracy (k=1,2,3,4) in Figure 18, under the setting without cross-session long-term memory. The results indicate that increasing `k` consistently improves performance, as larger `k` provides more opportunities to succeed in the presence of instability in dynamic online environments. Notably, UI-Copilot-7B exhibits stronger `pass@k` scaling behavior than Qwen2.5VL-7B on MemGUI-Bench, highlighting its superior potential for long-horizon tasks.

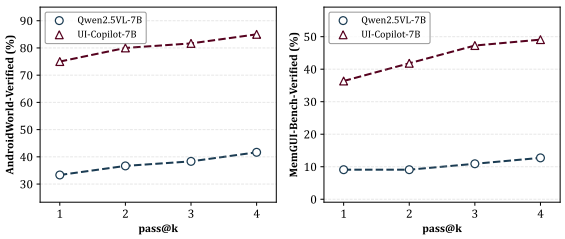
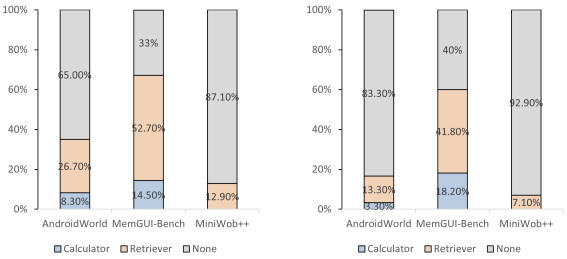


Figure 18: Pass@k validation on AndroidWorld-Verified (60 tasks) and MemGUI-Bench-Verified (55 tasks).

Tool Distribution. Figure 19a and Figure 19b illustrate the tool usage distributions of Qwen2.5VL-7B and UI-Copilot-7B. Overall, higher tool usage is strongly correlated with increased task difficulty, with MemGUI-Bench emerging as the most challenging benchmark and MiniWob++ as the easiest. Across all benchmarks, `Retriever` is invoked more frequently than `Calculator`. After applying TIPO, the overall frequency of tool invocation is slightly reduced, indicating more efficient and deliberate tool utilization.

Tool Error Type Analysis. Figure 20 compares the distributions of tool invocation error types for *Qwen2.5VL-7B* and UI-Copilot-7B on MemGUI-Bench-Verified. Compared to the base model, UI-Copilot-7B exhibits fewer tool type errors and execution errors, highlighting the improved reliability of tool invocation.



(a) Qwen2.5VL-7B (b) UI-Copilot-7B

Figure 19: **Tool Type Distribution** on AndroidWorld-Verified, MemGUI-Bench-Verified, and MiniWob++.

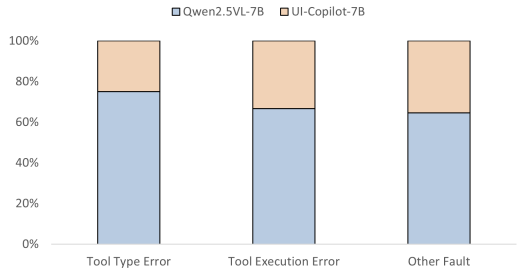


Figure 20: **Error Type Analysis of Tool Invocation.**

F More Cases

F.1 Successful Cases

Vanilla Execution. Figure 21 illustrates a case from MemGUI-Bench that does not require any tool invocation. Over six interaction steps, the agent successfully completes the search task by sequentially opening the target application, navigating to the correct section, and accessing the desired view, demonstrating its capability for coherent multi-turn execution without external tools.

Numerical Calculation. Figure 22 presents a MemGUI-Bench example involving mathematical computation. At step 12, the agent first retrieves the stock price and revenue growth rate, then invokes the Calculator to compute $price \times (1 + growth\_rate)$. The returned result, 306.89, exactly matches the ground-truth answer, validating the agent's ability to correctly delegate and integrate numerical reasoning via tool calling.

Memory Retrieval. Figure 23 shows two memory-intensive cases, one from AndroidWorld (top) and one from MemGUI-Bench (bottom). Both tasks require the agent to retain critical information from early interaction steps and utilize it for form filling in later stages, posing a challenge for long-horizon memory management. In both scenarios, the agent successfully invokes the Memory Retriever (Qwen3-4B) with the task goal and locally stored history, and accurately retrieves essential information (e.g., top-3 drivers' points and age) to complete the tasks.

F.2 Failed Cases

Reasoning Hallucination. As shown in Figure 24, UI-Copilot-7B fails to complete a maze navigation task that requires moving an object (X) to the bottom-right cell. At step 5, the object is located in the top-left cell, with a black barrier immediately below. Despite this visual constraint, the agent repeatedly plans and executes the `Down` action, resulting in a loop of incorrect behavior. This failure highlights limitations in visual perception and spatial reasoning.

Progress Hallucination. Figure 25 illustrates a case where UI-Copilot-7B exhibits progress confusion during multi-turn execution. The agent misinterprets the current task state and attempts to change the country, which is unrelated to the active sub-goal. As a result, the task terminates prematurely and is considered a failure.

Action Inconsistency As shown in Figure 26, UI-Copilot-7B 's internal plan indicates an intention to continue reviewing remaining transactions. However, it unexpectedly executes the `terminate` action, ending the task. This discrepancy between planning and execution reveals limitations in action consistency and reflective control.

Open the BBC Sports app, navigate into the Football section, open the 'Scores & Fixtures' view, and then execute a search for 'Bayern Munich'

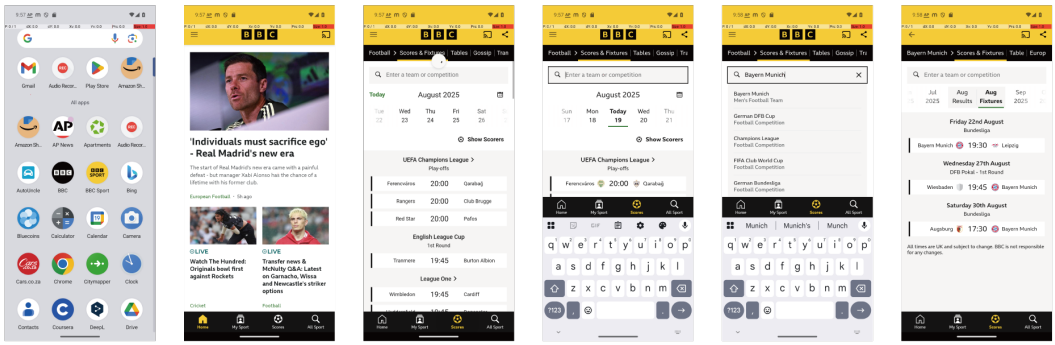
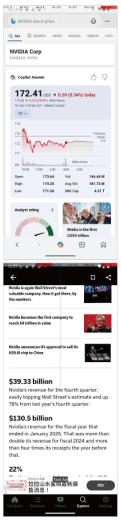


Figure 21: **Successful Cases**: Vanilla Rollout without tool usage.

*Open the bing app, search for 'NVIDIA stock price', and remember the current price. Then open the AP News app, find the latest news about NVIDIA earnings report, and remember the reported revenue growth rate. Finally, calculate stock price * (1 + revenue growth rate).*



Step-4

<tool> None </tool> <result> </result>

<think>I have successfully searched for the NVIDIA stock price. ***Looking at the screen, the current stock price is displayed as 172.41.*** I will remember this value and proceed to the next step, which is to open the AP News app to find the latest earnings report news.</think>...

Step-11

<think>I have successfully found the NVIDIA stock price (172.41) and the revenue growth rate (78%). Now, I need to perform the calculation: 172.41 * (1 + 0.78).
</think><summary>I need to calculate 172.41 * (1 + 0.78).</summary>

Step-12

def calculator(price,rate):...

<tool> ***Calculator*** </tool> <result> ***306.89*** </result> .

<action>{"action": "type", "text": "***306.89***"}</action>

Figure 22: **Successful Cases**: Tool-Integrated Rollout with Math Calculator.

Create a file in Markor, called receipt.md with the transactions from the receipt.png. Use Simple Gallery to view the receipt. Please enter transactions in csv format including the header "Date, Item, Amount".

Step-9

<tool> None </tool> <result> </result>

<think>I have extracted the transaction details from the receipt image. ***The Date, Item, Amount is 2023-03-23,Monitor Stand,$33.22, respectively.*** Now, I need to return to the Markor app to continue entering these details into the \"receipt.md\" file. </think>

<action>{"action": "system_button", "button": "Back"}</action>

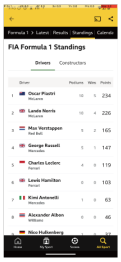
<summary>I have extracted the transaction details.</summary>

Step-12

<tool> ***Retriever*** </tool> <result> ***The Date, Item, Amount is 2023-03-23, Monitor Stand,$33.22, respectively***. </result>

<think>I need to enter the transaction details from the receipt image into the "receipt.md" file in CSV format. The Retriever result is 2023-03-23,Monitor Stand, $33.22 </think> <action>{"action": "type", "text": "2023-03-23,Monitor Stand,$33.22"} </action> <summary>I have extracted the transaction details.</summary>

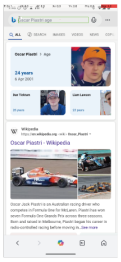
In the BBC Sports app, go to the 'Formula 1' -> 'Drivers'' standings. Remember the names and points of the top three drivers. Then, for each driver, search in the Bing app for their age. In the Joplin app, create a note titled F1 Top 3 Ages listing each driver with their points and age.



Step-5

<tool> None </tool> <result> </result>

<think>I have successfully identified the top three drivers and their points from the BBC Sports app. The next step is to find their ages. To do this, I will open the Bing app and search for each driver's age, starting with the first driver, Oscar Piastri.\n\n ***1. **Oscar Piastri**: 234 points\n2. **Lando Norris**: 226 points\n3. **Max Verstappen**: 165 points***</think>...



Step-8

<tool> None </tool> <result> </result>

<think>I have found the age of ***Oscar Piastri (24 years old)***. I need to go back to the search bar to search for the age of Lando Norris. I will click the search bar to initiate a new search.</think><action>{"action": "click", "coordinate": [469,146]}</action>

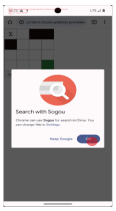
<summary> After finding Oscar Piastri is 24 years old, I clicked the search bar to search for the next driver's age.</summary>

Step-17

<tool> ***Retriever*** </tool> <result> The top3 drivers' names and points are Oscar Piastri: 234 points, 24 years old\\nLando Norris: 226 points, 25 years old\\nMax Verstappen: 165 points, 27 years old\ "</result> <think>I have already set the title of the note. Now I need to add the content of the note. The content should be the names, points, and ages of the top three F1 drivers. I will input this information into the body of the note. </think> ...

Figure 23: **Successful Cases**: Tool-Integrated Rollout with Memory Retriever.

Open the file task.html in Downloads in the file manager; when prompted open it with Chrome. Then navigate the X to the bottom-right cell, by using the direction buttons.



Step-4

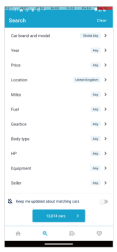
<tool> None </tool> <result> </result>
<think>I need to dismiss the Sogou search prompt to proceed with navigating the grid. The OK button is likely the way to close this dialog.</think>
<action>{'action": "click", "coordinate": [890, 1630]} </action>
<summary>I have opened the task.html file and dismissed the search prompt.</summary>

Step-5

<tool> None </tool> <result> </result>
<think>I need to navigate the 'X' to the bottom-right cell. The current position of 'X' is in the top-left corner. To move it to the bottom-right, I will need to use the 'Down' button repeatedly until the 'X' reaches the bottom row, then use the 'Right' button to move it to the rightmost column.</think> <action>{"action": "click", "coordinate": [200, 834]}</action>
<summary>I have cliked the 'Down' button to move 'X' downward.</summary>

Figure 24: **Bad Cases**: Reasoning Hallucination.

In the AutoUncle app (Germany), find a 'BMW 3 Series', an 'Audi A4', and a 'Mercedes-Benz C-Class'. For each car, remember its mileage, transmission type, and price. In the Joplin app, create a note titled German Sedan Comparison and list all three cars with all three of their specs.

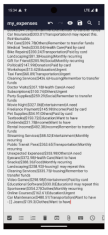


Step-7

<tool> None </tool> <result> </result>
<think>I have been unsuccessful in changing the location from "United Kingdom" to "Germany". Clicking on the location filter and the country name did not provide the expected result. Since I'm unable to change the country to Germany, which is a requirement of the task, I cannot proceed with finding the specified cars. Thus, I am concluding the task as infeasible.
</think> <action>{"action": "terminate", "status": failure}</action>
<summary>I think the task hase been completed.</summary>

Figure 25: **Bad Cases**: Progress Confusion.

Go through the transactions in my_expenses.txt in Markor. Log the reimbursable transactions in the pro expense.



Step-7

<tool> None </tool> <result> </result>
<think>I have logged the reimbursable transaction "Car Insurance" in the pro expense section Now, I need to continue reviewing the remaining transactions to identify more reimbursable ones. </think> <action>{"action": "terminate", "status": failure}</action>
<summary>I have l.</summary>

Figure 26: **Bad Cases**: Action Inconsistency.

Prompt for UI-Copilot-7B.

```
You are a GUI agent. You are given a task and your action history, with screenshots. You need
↪  to perform the next action to complete the task with the help of tools.

# Output Format
<tool> ... </tool>
<result> ... </result>
<think> ... </think>
<action> ... </action>
<summary> ... </summary>

# Tool Space

You can call the following tools, and the results are returned into `result` part:
- Calculator: Perform mathematical and numerical computations.
- Retriever: Retrieve valuable information from historical screenshots.
- None: Directly perform action without the need of tools.

# Action Space

You can perform the following actions:
- key: Perform a key event on the mobile device using adb's `keyevent` syntax.
- click: Click the point on the screen with specified (x, y) coordinates.
- long\_press: Press the point on the screen with specified (x, y) coordinates for a specified
↪  number of seconds.
- swipe: Swipe from starting point with specified (x, y) coordinates to endpoint with specified
↪  (x2, y2) coordinates.
- type: Input the specified text into the activated input box.
- answer: Output the specified answer.
- system\_button: Press the specified system button: Back, Home, Menu, or Enter.
- open: Open an application on the device specified by text.
- wait: Wait for a specified number of seconds for changes to occur.
- terminate: Terminate the current task and report its completion status: success or failure.

The arguments you can use are:
- coordinate: (x, y): The x and y pixels coordinates from the left and top edges.
- coordinate2: (x, y): The x and y pixels coordinates from the left and top edges for the
endpoint of a swipe.
- text: Text input required by actions like `key`, `type`, `answer`, and `open`.
- time: The time in seconds required by actions like `long\_press` and `wait`.
- button: System buttons available for pressing: Back, Home, or Enter. Possible values: Back,
↪  Home, Menu, Enter.
- status: The completion status of a terminated task. Possible values: success, failure.

# Example Output


<tool>Calculator</tool>
<result>...</result>
<think>...</think>
<action>{"action": "key", "text": "<value>"}
<summary> I have finished ... </summary>

# Note
- Format your output as a JSON object with the selected action and its arguments at the same
↪  level.
- Write the summary of your current progress in `summary` part.
- Plan the task and explain your reasoning step-by-step in `think` part.
- Write your action in the `action` part according to the action space.
- If the query asks a question, please answer the question through the answer action before
↪  terminating the process.
- Swipe the screen to find the File Manager app if needed.

# User prompt:
User Instruction:

# Assistant prompt
History Summary:
```

Figure 27: **Prompt for UI-Copilot-7B.**

Prompt for Retriever.

```
You are a GUI assitant for memory retriever. Given the task intruction and the interaction
↪  history, you need to provide all the information helpful for the task.
The overall task instruction is: '{task}'. The history information is as follows.

# History Information
step 1: '{step_1}'
step 2: '{step_2}'
...

Output the thinking process in the `think` part and the information summary in `answer` part.
# Example Output
<think>...</think> <answer> The five numbers displayed are 10,9,6,5,5. </answer>

The output format should be <think> ... </think> <answer> </answer>. Please strictly follow the
↪  format.
```

Figure 28: **Prompt for Retriever.**

Prompt for Calculator.

```
You are a GUI assistant for numerical calculation. Given the task instruction and the
↪  interaction history, you need to write executable python code to support numerical
↪  calculation.
The overall task instruction is: '{task}'. The history summary is as follows.

# Interaction History
step 1: '{step_1}'
step 2: '{step_2}'

Output the thinking process in `think` part and the executable python code in `python` part.

# Note

- You must directly output executable Python code in the <answer> section.

- The code should be self-contained, deterministic, and ready to execute without any additional
↪  explanation.

# Example Output
<think>
The task requires multiplying a list of numbers together.I will define a function in Python to
↪  compute the product accurately.
</think>
<python>
def product(nums):
    result = 1
    for n in nums:
        result *= n
    return result

nums = [10, 9, 6, 5, 5]
print(product(nums))
</python>

The output format should be <think> ... </think> <python> ... </python>. Please strictly follow
↪  the format.
```

Figure 29: **Prompt for Calculator.**