Towards Scalable Lightweight GUI Agents via Multi-role Orchestration

Ziwei Wang[1,2], Junjie Zheng[1,2], Leyang Yang[1,2], Sheng Zhou[1] †, Xiaoxuan Tang[3], Zhouhua Fang[3], Zhiwei Liu[3], Dajun Chen[3], Yong Li[3] †, Jiajun Bu[1,2]
[1]Zhejiang Key Laboratory of Accessible Perception and Intelligent Systems, Zhejiang University
[2]College of Computer Science and Technology, Zhejiang University
[3]AntGroup
{wangziwei98, jjzheng0315, yangleyang, zhousheng_zju, bjj}@zju.edu.cn
{tangxiaoxuan.txx, fangzhouhua.fzh, biao.lzw, chendajun.cdj, liyong.liy}@antgroup.com

Abstract

Autonomous Graphical User Interface (GUI) agents powered by Multimodal Large Language Models (MLLMs) enable digital automation on end-user devices. While scaling both parameters and data has yielded substantial gains, advanced methods still suffer from prohibitive deployment costs on resource-constrained devices. When facing complex in-the-wild scenarios, lightweight GUI agents are bottlenecked by limited capacity and poor task scalability under end-to-end episodic learning, impeding multi-agent systems (MAS) adaptation, while training multiple skill-specific experts remains costly. *Can we strike an effective trade-off in this cost–scalability dilemma, enabling lightweight MLLMs to participate in realistic GUI workflows?* To address these challenges, we propose LAMO framework, which endows a lightweight MLLM with GUI-specific knowledge and task scalability, allowing multi-role orchestration to expand their capability boundary for GUI automation. LAMO combines role-oriented data synthesis with a two-stage training recipe: (i) supervised fine-tuning with Perplexity-Weighted Cross-Entropy optimization for knowledge distillation and visual perception enhancement, and (ii) reinforcement learning for role-oriented cooperative exploration. Via LAMO, we develop a task-scalable native GUI agent LAMO-3B supporting monolithic execution and MAS-style orchestration. When paired with advanced planners, as a plug-and-play policy executor, LAMO-3B can continuously benefit from planner advances, enabling a higher performance ceiling. Extensive static and online evaluations validate the effectiveness of our designs.

1 Introduction

The rapid evolution of Multimodal Large Language Models (MLLMs) has significantly propelled the development of agents for Graphical User Interfaces (GUIs) (Gu et al., 2026), marking a pivotal frontier in GUI automation (Ye et al., 2025; Gonzalez-Pumariega et al., 2025). These autonomous GUI agents are transforming how humans interact with digital systems by operating mobile/computer interfaces to accomplish user goals (Tang et al., 2025).

GUI automation has progressed from static settings (Li et al., 2024) to increasingly complex in-the-wild online environments (Rawles et al., 2024; Xie et al., 2024). To address this challenging long-horizon reasoning task that integrates intent parsing, screen perception, history clues, and tool execution to achieve goals sequentially (Hu et al., 2025), current advanced methods have yielded substantial gains by scaling both parameters and data (Qin et al., 2025; Gu et al., 2025). Scaling laws (Kaplan et al., 2020) further endow large models with robust task scalability: they can build MAS via context engineering, alleviating the "lost-in-the-middle" issue (Liu et al., 2023) and enabling efficient context management (Ye et al., 2025; Chen et al., 2025b), thus improving performance in navigating realistic GUI workflows. However, these gains come at a significantly higher system cost, making large-scale models impractical to deploy on resource-constrained devices.

Against this backdrop, lightweight GUI agents have drawn growing attention (Wu et al., 2024; Park et al., 2025b; Wang et al., 2025c; Lu et al., 2025b; Lin et al., 2025), with steady progress driven by post-training techniques such as supervised fine-tuning (SFT) and reinforcement learning (RL). Despite promising results on static, step-wise settings (Li et al., 2024), small-scale MLLMs face two major constraints: inherent capacity bottlenecks due to their limited parameter size, and the end-to-end episodic learning framework (Liu et al., 2025; Luo et al., 2025) couples high-level reasoning and low-level execution into a fixed pipeline, which suffers markedly when navigating realistic

†Co-corresponding authors.
Code: https://github.com/BigTaige/LAMO

Findings of the Association for Computational Linguistics: ACL 2026, pages 22359–22388

workflows (Rawles et al., 2024; Xie et al., 2024). These constraints limit scalability and impede adaptation to MAS. Although training multiple skill-specific experts can mitigate these weaknesses, such methods remain costly (Zhao et al., 2025; Park et al., 2025a). *Can we strike an effective trade-off in this cost–scalability dilemma, enabling lightweight MLLMs to participate in realistic GUI workflows?*

To address these challenges, we propose **LAMO**, a framework for **L**ightweight **A**gent **M**ulti-role **O**rchestration in GUI automation. LAMO endows a lightweight MLLM with GUI-specific knowledge and task scalability through parameter sharing, enabling multi-role orchestration for MAS adaptation and expanding their capability boundary to solve increasingly complex in-the-wild scenarios. Unlike monolithic end-to-end agentic recipes (Liu et al., 2025; Lin et al., 2025; Luo et al., 2025), LAMO trains a lightweight MLLM to flexibly orchestrate skill-specific roles via role-oriented data synthesis and a two-stage training recipe: *(i) SFT with a Perplexity-Weighted Cross-Entropy optimization for domain knowledge distillation, instruction following, and fine-grained visual perception*; and *(ii) Multi-task RL for collaborative exploration and knowledge transfer across role-oriented GUI tasks*.

Via LAMO framework, we produce **LAMO-3B**, a lightweight GUI agent that can flexibly orchestrate skill-specific roles to participate in realistic GUI workflows. In particular, LAMO-3B functions as a reliable policy executor for precise low-level GUI execution; when paired with an advanced planner, as a plug-and-play policy executor, it can continually benefit from planner improvements, offering a higher performance ceiling than native monolithic models. Extensive experiments in both static (ScreenSpot-pro (Li et al., 2025a) and AndroidControl (Li et al., 2024)) and online (MiniWob++ (Liu et al., 2018), AndroidWorld (Rawles et al., 2024), and OSWorld (Xie et al., 2024)) environments validate the effectiveness and potential of LAMO.

Our main contributions are as follows:

- We propose the LAMO framework to endow a lightweight MLLM with task scalability for MAS adaptation, expanding its capability boundary to solve increasingly complex in-the-wild scenarios.
- Via LAMO, we train a task-scalable GUI agent, LAMO-3B, that can be orchestrated into skill-specific roles for GUI-oriented tasks. Paired with advanced planners as the plug-and-play policy executor, LAMO-3B's task scalability raises the performance ceiling for GUI automation.
- Extensive experiments on both static and online benchmarks demonstrate the potential of LAMO and the effectiveness of our designs.

2 Related Work

Post-training techniques, such as SFT and RL, have advanced MLLM-powered GUI agents. MP-GUI (Wang et al., 2025b) enhances the GUI understanding of MLLMs via multi-perceiver augmentation, thereby improving its agentic performance. The powerful UI-TARS (Qin et al., 2025), trained via an SFT then RL strategy under a data flywheel, achieves milestone results in GUI automation. Furthermore, GUI-R1 (Luo et al., 2025), InfiGUI-R1 (Liu et al., 2025), HAR-GUI (Wang et al., 2025c), and UI-S1 (Lu et al., 2025b) employ GRPO (Shao et al., 2024) to further explore the potential of RL in GUI automation. However, despite strong static performance, these lightweight GUI agents degrade sharply online, widening the gap to realistic GUI workflows (Yang et al., 2025a). To address increasingly complex in-the-wild scenarios, MAS have emerged as a promising trend (Hu et al., 2025; Chen et al., 2025b; Zhang et al., 2025a). Leveraging the robust instruction-following of large-scale MLLMs, task scalability can be achieved via context engineering, which in turn enables MAS that orchestrate multiple skill-specific agents, exemplified by the Agent-S family (Agashe et al., 2025; Gonzalez-Pumariega et al., 2025) and MobileAgent family (Wang et al., 2024; Ye et al., 2025), enable effective long-horizon reasoning. However, current advanced MAS typically rely on large-scale MLLMs (Gemini Team, 2025; OpenAI, 2025b), which are suboptimal for low-level GUI execution and thus require specialized, large-parameter GUI experts for reliable actuation, for example, Agent-S2 (Agashe et al., 2025) deploy multiple large-scale, GUI-specific executors, including UI-TARS-72B-DPO (Qin et al., 2025) for visual grounding, Tesseract OCR (Tesseract OCR, 2025) for textual grounding, and UNO (Unotools, 2025) for structural grounding, resulting in prohibitive system cost. Meanwhile, lightweight GUI agents trained via end-to-end episodic learning endow poor task scalability, restricting their adaption to MAS workflows. This has driven demand for task-scalable lightweight GUI agents.

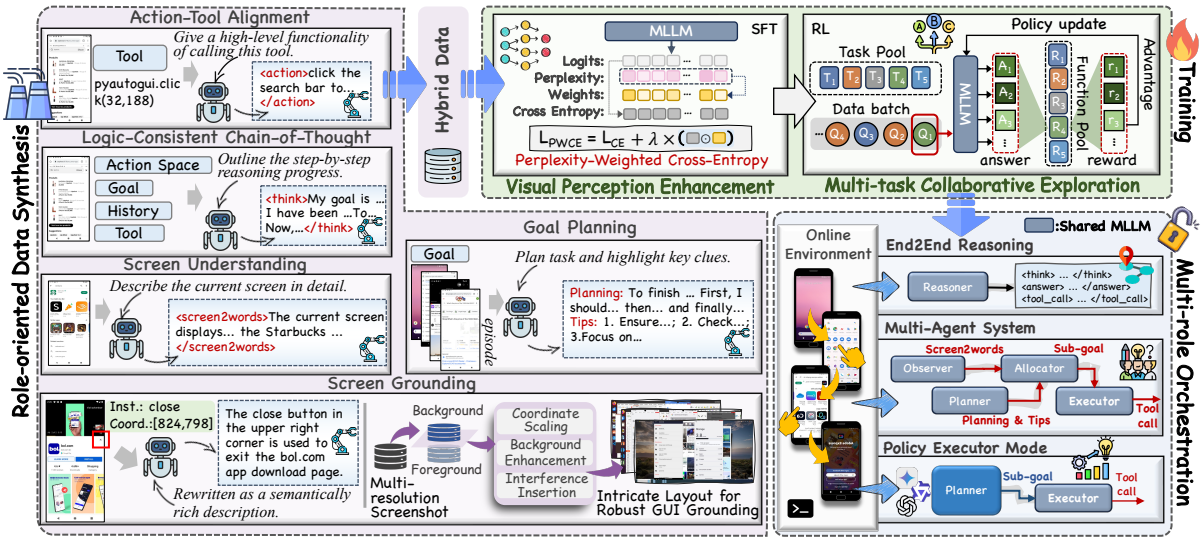

Figure 1: Overview of the Lightweight Agent Multi-role Orchestration (LAMO) framework. LAMO integrates role-oriented data synthesis with a two-round training recipe to enhance screen perception, long-horizon reasoning, and multi-role orchestration. It enables versatile inference modes, allowing a lightweight MLLM to function as end-to-end monolithic agent, coordinated MAS, or plug-and-play executor paired with advanced planners. Such scalability expands the capability boundary of lightweight MLLMs in GUI automation via MAS adaptation.

3 Problem Formulation

For step-wise GUI tasks (e.g., grounding and screen QA), the MLLM $\mathcal{M}_\theta(\cdot)$ performs end-to-end inference as $y_k{=}\mathcal{M}_\theta(y_{<k} \mid \mathcal{I}, o)$, where $\mathcal{I}$ denotes the instruction and o is a screenshot image. For long-horizon GUI tasks, let $\mathcal{T}$ denote an episode with an overall goal $\mathcal{G}$, where $\mathcal{T} = (\mathcal{G}, (o_1, a_1), ..., (o_n, a_n))$ and each observation o_t is the screenshot at t-th step. The atomic action $a_t \in \mathcal{A}$ is an operation executed by the agent, with $\mathcal{A}$ denoting the predefined `PyAutoGUI`-style action space. The agentic task is formulated as a Markov Decision Process $P(a_t \mid \mathcal{G}, \mathcal{I}, o_{\leq t}, a_{<t})$ that drives the agent step by step toward achieving the goal.

4 Methodology

Fig. 1 overviews the **LAMO** framework and the key designs are detailed below.

4.1 Role-oriented Data Synthesis

Data-centric native GUI agents, powered by post-training techniques such as SFT and RL, show significant promise (Qin et al., 2025; Wang et al., 2025c). We observe that lightweight MLLMs, though weak on long-horizon tasks where the agent must handle screen analysis, policy decisions, and tool invocation simultaneously, perform reliably when these components are handled independently. Following this insight, we aim to decompose high-level reasoning into a GUI-oriented sub-task flow: *(i) progressively improving sub-task performance to achieve overall gains, and (ii) using parameter sharing and context engineering to orchestrate the model into skill-specific roles that communicate and collaborate efficiently for GUI automation.*

To achieve this goal, we introduce a role-oriented data synthesis strategy that decomposes GUI automation into five core capabilities: ***Action–Tool Alignment (ATA)***[1] for mapping high-level instructions to low-level executable tools; ***Logic-Consistent CoT (LCC)*** for analyzing in-context clues and yielding logically coherent reasoning; ***Screen Understanding (SU)*** for interpreting screen functionality and screen details; ***Goal Planning (GP)*** for decomposing overall goals into executable subtasks and identifying the key considerations for accomplishing these tasks; and ***Screen Grounding (SG)*** for fine-grained spatial and UI layout perception. We synthesize skill-specific training data for each category using teacher models: Qwen-2.5-VL-72B-Instruct (Bai et al., 2025b) ($\mathcal{M}_1^{\mathbb{T}}$) for ATA and SG, and Gemini-2.5-Pro (Gemini Team, 2025) ($\mathcal{M}_2^{\mathbb{T}}$) for SU, LCC, and GP.

The ATA task trains the agent as a policy executor by synthesizing an action-aligned description $\mathcal{C}_{\text{act}} = \mathcal{M}_1^{\mathbb{T}}(\mathcal{I}_{\text{Tool}}, o_k, a_k)$, where $\mathcal{C}_{\text{act}}$ verbalizes the atomic action a_k. The LCC task equips

[1]This data format also supports tool-call summarization, improving history reconstruction at inference.

the agent with long-horizon reasoning by synthesizing step-wise logically rigorous CoT $\mathcal{C}_{\text{CoT}} = \mathcal{M}_2^{\mathbb{T}}(\mathcal{I}_{\text{CoT}}, \mathcal{G}, o_k, a_{\leq k})$, where $\mathcal{C}_{\text{CoT}}$ provides the rationale at step k. The SU task trains the agent as a screen observer by synthesizing multi-grained screen descriptions $\mathcal{C}_{\text{S2W}} = \mathcal{M}_2^{\mathbb{T}}(\mathcal{I}_{\text{S2W}}, o_k)$, where $\mathcal{C}_{\text{S2W}}$ summarizes screen functionality, layout, and key UI elements. The GP task trains the agent as a planner. We synthesize planning supervision as $(\mathcal{C}_{\text{Plan}}, \mathcal{C}_{\text{Tips}}) = \mathcal{M}_2^{\mathbb{T}}(\mathcal{I}_{\text{Plan}}, \mathcal{G}, o_{\leq k}, a_{\leq k})$, where $\mathcal{C}_{\text{Plan}}$ describes subtasks and $\mathcal{C}_{\text{Tips}}$ provides key considerations for accomplishing $\mathcal{G}$. The instructions $\mathcal{I}_{\text{Tool}}$, $\mathcal{I}_{\text{CoT}}$, $\mathcal{I}_{\text{S2W}}$, and $\mathcal{I}_{\text{Plan}}$ are task-specific prompts for ATA, LCC, SU, and GP, respectively.

For SG, we target two practical challenges faced by GUI agents: ***(i)*** *limited semantic understanding of element descriptions, especially for semantically sparse elements that are common in real scenarios;* ***(ii)*** *difficulty grounding targets in complex layouts with abundant distracting signals.* For the first challenge, we enrich the original element description $\mathcal{C}_{\text{orig}}$ into a semantically rich caption:$\mathcal{C}_{\text{rich}} = \mathcal{M}_1^{\mathbb{T}}(\mathcal{I}_G, o_k, \mathcal{C}_{\text{orig}})$. We then distill $\mathcal{C}_{\text{rich}}$ together with the element coordinates $\mathcal{P}_{\text{point}}$ into the agent $\mathcal{M}_\theta$ by training it to predict $(\mathcal{C}_{\text{rich}}, \mathcal{P}_{\text{point}})$ given $(o_k, \mathcal{C}_{\text{orig}})$ under the training prompt $\mathcal{I}_G^*$, fostering fine-grained semantic understanding of UI elements. $\mathcal{I}_G$ and $\mathcal{I}_G^*$ denote the prompts used for data synthesis and for training/inference, respectively.

For the second challenge, we perform rule-based augmentation on the grounding metadata $\mathcal{D}$: we sample foregrounds $\mathcal{D}^+$ and backgrounds $\mathcal{D}^-$ from $\mathcal{D}$, take a screen from $\mathcal{D}^-$ as the background view $\mathcal{O}_{\text{back}}$, and overlay a target $(o_i, P^i_{\text{point}})$ and multiple distractor screens from $\mathcal{D}^+$ with random scaling, yielding a cluttered intricate-layout screen $\ddot{\mathcal{O}}$ with scaled coordinates $\ddot{P}^i_{\text{point}}$. In this way, each meta sample $(o_i, P^i_{\text{point}}, \mathcal{C}_{\text{orig}}) \in \mathcal{D}$ is converted into an Intricate-Layout Grounding (ILG) sample $(\ddot{\mathcal{O}}, \ddot{P}^i_{\text{point}}, \mathcal{C}_{\text{orig}})$. This rule-based procedure ensures controllable augmentation quality and enables low-cost synthesis of large-scale ILG data to improve grounding robustness in complex real-world screen states (Tab. 6). Instructions and ILG data synthesis details are in App. A.6 and A.4.

4.2 Visual Perception Enhancement

SFT is a reasonable solution to equip MLLMs with domain knowledge during post-training (Wu et al., 2024), but its effectiveness degrades on agentic tasks requiring fine-grained visual perception, especially UI grounding with accurate numerical coordinates. Our empirical evaluations find that while SFT improves textual learning, yet predicted coordinates exhibit small but systematic deviations from the ground truth, dicating limited spatial awareness. Qualitative analysis suggests that coordinate tokens often exhibit higher perplexity than textual tokens during SFT, reflecting the inherent uncertainty of numerical prediction.

To mitigate this, we introduce the Perplexity-Weighted Cross-Entropy (PWCE) loss function, which reweights tokens according to their perplexity: high-perplexity tokens, particularly coordinates, receive larger loss weights, steering optimization toward uncertain, spatially critical outputs and enhancing screen details perception.

Specifically, let the shifted last-layer hidden states of the LLM for next-token prediction be $\tilde{h} \in \mathbb{R}^{b\times l\times d}$ and the corresponding labels be $\tilde{y} \in \mathbb{R}^{b\times l}$, where b, l, and d denote batch size, sequence length, and hidden dimension, respectively. The model produces logits $h^* = \tilde{h}W^\top \in \mathbb{R}^{b\times l\times V}$, where W is the embedding matrix and V is the vocabulary size. The standard cross-entropy loss over the sequence is $\mathcal{L}_{\text{CE}} = \text{CE}(h^*, \tilde{y})$. We construct a binary mask M to indicate tokens generated after the input. For each masked index $i \in M$, we compute probabilities $p_i = \text{softmax}(h_i^*)$, token entropy $E_i = -\sum_{v=1}^{V} p_{i,v}\log(p_{i,v}+\epsilon)$, and perplexity $\text{PPL}_i = \min\left(\exp(\sqrt{E_i}),\, \beta\right)$, where ϵ and β are hyperparameters. Let $\overline{\text{PPL}}$ denote the mean perplexity over tokens in M. The perplexity-oriented weights and the perplexity-weighted loss can be formulated as,

$$w_i = \frac{1+\alpha\,\frac{\text{PPL}_i}{\overline{\text{PPL}}+\epsilon}}{\frac{1}{|M|}\sum_{j\in M}\left(1+\alpha\,\frac{\text{PPL}_j}{\overline{\text{PPL}}+\epsilon}\right)}, \quad (1)$$

$$\mathcal{L}_{\text{PW}} = \frac{1}{|M|}\sum_{i\in M} w_i \cdot \text{CE}(h_i^*, \tilde{y}_i). \quad (2)$$

Finally, the PWCE objective is defined as $\mathcal{L}_{\text{PWCE}} = \mathcal{L}_{\text{CE}} + \lambda\,\mathcal{L}_{\text{PW}}$, where α and λ are hyperparameters. PWCE dynamically assigns higher weights to numerical coordinate tokens and key contextual tokens with higher generation perplexity, guiding the model to percept screen details.

4.3 Multi-task Collaboration Exploration

Following SFT, the model acquires extensive GUI-specific knowledge, improves instruction following,

and adapts to role orchestration. Then, we perform a second round of RL with multi-task cooperative exploration to facilitate the discovery of optimal reasoning pathways in role-oriented tasks[2].

Specifically, we curate the hybrid data (ATA, SU, GP, SG, and the formatted step-wise agentic task LCC) to build a task pool[3] and construct a function pool that stores task-specific, rule-based reward functions for advantage estimation. For SU and GP, the reward is defined as the normalized similarity between prediction and label under the TF-IDF metric, $r_{\text{agent}} = \text{norm}(\text{TF-IDF}(y_{\text{pred}}, y_{\text{label}})) \in [0, 1]$. For SG, we extract coordinate points from the predictions and, following Wang et al., 2025c, compute their geometric distance to the ground truth to define the reward r_{agent}. For ATA and agentic tasks, exemplified by `pyautogui.write(message='$50')`, we parse the tool class (`write`) and tool value (`'$50'`) from the prediction, compute binary $(0/1)$ scores r_{class} and r_{val} via string matching, and aggregate them as $r_{\text{agent}} = r_{\text{class}} + r_{\text{val}}$. For coordinate-based tool values (e.g., `click`, `swipe`, `dragTo`), we reuse the SG reward function. A length-penalty $r_{\text{penalty}} = -\varphi \cdot \frac{\text{length}(y_{pred})}{L_{\max}}$ is added to all reward functions to avoid uncontrolled output length. Finally, we employ GRPO (Shao et al., 2024) to conduct multi-task cooperative exploration.

4.4 Multi-role Orchestration

Using the LAMO framework, we yield a task-scalable GUI agent LAMO-3B ($\mathcal{M}_\Theta$), facilitating the following inference modes for MAS adaptation:

End-to-End Reasoning. LAMO-3B can serve as a reasoner with a ReAct-style(Yao et al., 2023) agentic reasoning paradigm, enabling it to attend to details in the dynamic context (e.g., clues from historical interactions and tool descriptions) and to analyze the current screen state to make decisions with structured outputs. At time step t, given observations $o_{\leq t}$ and interaction history $a_{<t}$, LAMO-3B jointly encodes visual and textual inputs and outputs a structured decision:

$$\mathcal{S}_t = \mathcal{M}_\Theta(\mathcal{G}, \mathcal{I}_{\text{e2e}}, o_{\leq t}, a_{<t}) \,. \tag{3}$$

The instruction $\mathcal{I}_{\text{e2e}}$ guides $\mathcal{M}_\Theta$ to perform step-wise reasoning and generate structured $\mathcal{S}_t$, formatted as `<think>CoT</think> <answer> action decision </answer> <tool_call> atomic action` a_t`</tool_call>`.

Algorithm 1 MAS workflow built on LAMO-3B

1: **Initialization:** overall goal $\mathcal{G}$, time step t, screenshot o_t, GUI agent $\mathcal{M}_\Theta$, action space $\mathcal{A}$, max execution steps $T_{\max}$, instructions $[\tilde{\mathcal{I}}_{\text{obs}}, \tilde{\mathcal{I}}_{\text{plan}}, \tilde{\mathcal{I}}_{\text{act}}, \tilde{\mathcal{I}}_{\text{exec}}]$.
2: **repeat**
3: **Observation:** $\mathcal{C}_{\text{s2w}} \leftarrow \mathcal{M}_\Theta^{\text{Observer}}(\tilde{\mathcal{I}}_{\text{obs}}, o_t)$ *// Provide richly detailed semantic descriptions of the screen.*
4: **Planning:** $(\mathcal{C}_{\text{plan}}, \mathcal{C}_{\text{tips}}) \leftarrow \mathcal{M}_\Theta^{\text{Planner}}(\tilde{\mathcal{I}}_{\text{plan}}, \mathcal{G}, o_{\leq t})$ *// Interpret the goal, decompose it into subtasks, and provide actionable guidelines during execution.*
5: **Allocation:** $\mathcal{C}_{\text{action}} \leftarrow \mathcal{M}_\Theta^{\text{Allocator}}(\tilde{\mathcal{I}}_{\text{act}}, o_t, a_{<t}, \mathcal{C}_{\text{s2w}}, \mathcal{C}_{\text{plan}}, \mathcal{C}_{\text{tips}})$ *// Assign a single executable action for the current screen based on historical interactions and contextual clues.*
6: **Execution:** $a_t \leftarrow \mathcal{M}_\Theta^{\text{Executor}}(\tilde{\mathcal{I}}_{\text{exec}}, o_t, \mathcal{C}_{\text{action}})$ *// Based on the instruction, select the optimal tool from the action space and execute it within the environment.*
7: **until** $\mathcal{G}$ reached or $t > T_{\max}$

Multi-Agent System (MAS). To address the limited reasoning of lightweight MLLMs under the Scaling Laws (Kaplan et al., 2020) and the "lost-in-the-middle" issue in long-horizon interactions (Chen et al., 2025b), we orchestrate LAMO-3B into a parameter-shared MAS that decomposes GUI automation into skill-specific roles. As shown in Fig. 1 and Alg. 1, LAMO-3B uses a shared backbone to instantiate four agents: Observer $\mathcal{M}_\Theta^{Observer}$, Planner $\mathcal{M}_\Theta^{Planner}$, Allocator $\mathcal{M}_\Theta^{Allocator}$ and Executor $\mathcal{M}_\Theta^{Executor}$, which jointly reduce task complexity, improve context management, and enable low-hallucination reasoning.

Policy Executor Mode. Given the increasing complexity and long-horizon nature of GUI automation (e.g., computer-use and cross-APP scenarios), success often depends on implicit GUI-specific priors and strong planning capabilities in the underlying foundation MLLMs (Agashe et al., 2025). Yet, constrained by limited parameters, lightweight GUI agents struggle to align with real-world environments (Yang et al., 2025a). To bridge this gap, LAMO enables LAMO-3B to act as a plug-and-play policy executor to reliably interacts with the environment, worked with an advanced large-scale MLLM planner (e.g., Gemini-2.5-Pro or GPT-5) to drive GUI automation. Compared with GUI agents with limited task scalability (Liu et al., 2025), LAMO-3B can evolve alongside advanced planners, yielding a higher performance ceiling.

Specifically, the execution can be expressed as follows, an advanced MLLMs act as a planner to provide an executable high-level instruction, then

[2]Our empirical evaluations reveal that multi-task RL facilitates the acquisition of shared representations and inter-task dependencies across GUI-related tasks, enabling effective knowledge transfer and improving overall performance.

[3]Each task is assigned a unique tag for rule-based hybrid reward computation.

Method	Development			Creative			CAD			Scientific			Office			MacOS			Avg.		
	Text	Icon	Avg.	Text	Icon	Avg.	Text	Icon	Avg.	Text	Icon	Avg.	Text	Icon	Avg.	Text	Icon	Avg.	Text	Icon	Avg.
Generic Models																					
Claude Computer Use(Anthropic, 2024)	22.0	3.9	12.6	25.9	3.4	16.8	14.5	3.7	11.9	33.9	15.8	25.8	30.1	16.3	26.9	11.0	4.5	8.1	23.4	7.1	17.1
Qwen2.5-VL-3B(Bai et al., 2025b)	38.3	3.4	21.4	40.9	4.9	25.8	22.3	6.3	18.4	44.4	10.0	29.5	48.0	17.0	40.9	33.6	4.5	20.4	37.8	6.6	25.9
Qwen2.5-VL-7B(Bai et al., 2025b)	51.9	4.8	29.1	36.9	8.4	24.9	17.8	1.6	13.8	48.6	8.2	31.1	53.7	18.9	45.7	34.6	7.9	22.4	39.9	7.6	26.8
Kimi-VL-A3B-Instruct(Team et al., 2025)	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	34.5
GUI-specific Models																					
UGround-72B(Qian et al., 2025)	55.8	4.8	31.1	**54.0**	**10.5**	**35.8**	16.8	4.7	13.8	**70.8**	22.7	**50.0**	61.0	18.9	51.3	40.2	7.9	25.5	–	–	34.5
SeeClick(Cheng et al., 2024)	0.6	0.0	0.3	1.0	0.0	0.6	2.5	0.0	1.9	3.5	0.0	2.0	1.1	0.0	0.9	2.8	0.0	1.5	1.8	0.0	1.1
UGround-7B(Qian et al., 2025)	26.6	2.1	14.7	27.3	2.8	17.0	14.2	1.6	11.1	31.9	2.7	19.3	31.6	11.3	27.0	17.8	0.0	9.7	25.0	2.8	16.5
OS-Atlas-7B(Wu et al., 2024)	33.1	1.4	17.7	28.8	2.8	17.9	12.2	4.7	10.3	37.5	7.3	24.4	33.9	5.7	27.4	27.1	4.5	16.8	28.1	4.0	18.9
GUI-R1-7B(Luo et al., 2025)	49.4	4.8	–	38.9	8.4	–	23.9	6.3	–	55.6	11.8	–	58.7	26.4	–	42.1	16.9	–	–	–	–
UI-S1-7B(Lu et al., 2025b)	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	30.6
UI-TARS-7B(Qin et al., 2025)	**58.4**	**12.4**	**36.1**	50.0	9.1	32.8	20.8	9.4	18.0	63.9	**31.8**	**50.0**	63.3	20.8	53.3	30.8	16.9	24.5	47.8	16.2	35.7
OS-Atlas-4B(Wu et al., 2024)	7.1	0.0	3.7	3.0	1.4	2.3	2.0	0.0	1.5	9.0	5.5	7.5	5.1	3.8	4.8	5.6	0.0	3.1	5.0	1.7	3.7
GUI-R1-3B(Luo et al., 2025)	33.8	4.8	–	40.9	5.6	–	26.4	7.8	–	61.8	17.3	–	53.6	17.0	–	28.1	5.6	–	–	–	–
UI-TARS-2B(Qin et al., 2025)	47.4	4.1	26.4	42.9	6.3	27.6	17.8	4.7	14.6	56.9	17.3	39.8	50.3	17.0	42.6	21.5	5.6	14.3	39.6	8.4	27.7
InfiGUI-R1-3B(Liu et al., 2025)	51.3	**12.4**	32.4	44.9	7.0	29.0	33.0	14.1	28.4	58.3	20.0	41.7	**65.5**	**28.3**	**57.0**	**43.9**	12.4	**29.6**	**49.1**	14.1	35.7
LAMO-3B (ours)	52.6	9.7	31.8	40.9	9.1	27.6	**39.1**	**21.9**	**34.9**	52.8	27.3	41.7	65.0	24.5	55.7	35.5	**21.3**	29.1	47.9	**17.1**	**36.1**

Table 1: Grounding accuracy on ScreenSpot-pro. **Bold** represents the best results, underlined is the second best.

LAMO-3B converts it into atomic screen actions. The process can be formulated as,

$$\mathcal{C}^{*}_{action} = \mathcal{M}^{Planner}_{\Delta}(\mathcal{I}^{*}_{\Delta}, \mathcal{G}, o_{\leq t}, a_{<t}), \quad (4)$$

$$a_t = \mathcal{M}^{Executor}_{\Theta}(\tilde{\mathcal{I}}_{\text{exec}}, o_t, \mathcal{C}^{*}_{action}), \quad (5)$$

where $\mathcal{M}^{Executor}_{\Delta}$ denotes an advanced foundational MLLM, conditioned on the instruction $\mathcal{I}^{*}_{\Delta}$. Our multi-role orchestration strategy enables LAMO-3B to adapt seamlessly to both monolithic reasoning and cooperative multi-agent execution, thereby enhancing robustness and scalability in GUI automation. Case study in App. A.7.

5 Experiments

5.1 Implementation Details

Using the constructed hybrid dataset (Sec. 4.1; see App. A.1 for data curation details) and our training recipe, we develop LAMO-3B from Qwen2.5-VL-3B-Instruct (Bai et al., 2025b) under the LAMO framework. The data distillation stage applies SFT for 1 epoch with a learning rate of 4e-6, warmup ratio 0.03, global batch size 256, and LoRA (rank 128, alpha 256, dropout 0.001). In the RL stage, the vision backbone is frozen while the merge layer and LLM are trained with GRPO for 1 epochs at learning rate 1e-6, rollout batch size 32, generating 8 rollouts per sample. For hparams, we set ϵ=1e-12, β=1.5, α=0.5, L_{max}=120, φ =0.3 and λ=0.09. AdamW is used as the optimizer. All experiments are conducted on 8 NVIDIA H20 96GB GPUs.

5.2 Benchmarks

LAMO-3B is evaluated across web, mobile, and desktop environments. We use ScreenSpot (Cheng et al., 2024), ScreenSpot-v2 (Li et al., 2025b), and ScreenSpot-pro (Li et al., 2025a) to assess screen grounding, and AndroidControl (Li et al., 2024), a static single-step mobile benchmark, to assess agentic performance. To align real-world usage, multi-role orchestration is evaluated in the online web environment MiniWob++ (Liu et al., 2018) and the online mobile environment AndroidWorld (Rawles et al., 2024). In addition, OS-World (Xie et al., 2024) is used to measure the effectiveness of LAMO-3B as a policy executor in computer-use online scenarios. See App. A.2 for benchmark details and the corresponding metrics.

Method	AC–Low			AC–High		
	Type	Ground.	SR	Type	Ground.	SR
GPT-4o(Hurst et al., 2024)	74.3	38.7	28.4	63.1	30.9	21.2
OS-Genesis-7B(Sun et al., 2025)	90.7	–	74.2	66.2	–	44.5
GUI-R1-3B(Luo et al., 2025)	–	–	–	58.0	56.2	46.6
GUI-R1-7B(Luo et al., 2025)	–	–	–	71.6	65.6	51.7
SeeClick(Cheng et al., 2024)	93.0	73.4	75.0	82.9	62.9	59.1
InternVL-2-4B(Chen et al., 2024)	90.9	84.1	80.1	84.1	72.7	66.7
OS-Atlas-4B(Wu et al., 2024)	91.9	83.8	80.6	**84.7**	**73.8**	**67.5**
LAMO-3B	**97.2**	**86.7**	**92.1**	77.1	72.6	65.5

Table 2: Performance comparison on AndroidControl-Low (AC-Low) and AndroidControl-High (AC-High).

5.3 GUI-Oriented Foundation Performance

We evaluate the fundamental capabilities of LAMO-3B in processing GUI-related tasks by screen grounding (Tab.1) and step-wise agentic performance (Tab.2). As shown in Tab.1, LAMO-3B achieves overall leading performance, particularly against GUI-specialized methods with substantially larger parameter scales, such as UGround-72B, UI-TARS-7B, OS-Atlas-7B and UI-S1-7B. Com-

pared with the foundational model Qwen2.5-VL-3B, our approach yields a 39.4% overall gain. Under comparable model sizes, LAMO-3B consistently outperforms previous methods, and surpasses the advanced methods on graphical UI element grounding, especially in the challenging CAD scenarios. We observe that LAMO-3B exhibits stable visual perception of small-size UI elements, enabling reliable perception across screens with varying resolutions. The stable screen grounding provides a solid foundation for precise screen interaction in GUI automation. In Tab.2, compared with the methods tailored for single-task optimization on the training set, our LAMO-3B still achieves competitive results. Particularly, the leading performance on AC-Low demonstrates that LAMO-3B can achieve accurate action-tool alignment under explicit action instructions, verifying the effectiveness of the LAMO framework.

Method	Success Rate
Qwen2.5-VL-3B (Bai et al., 2025b)	34.6
OS-Atlas-7B (Wu et al., 2024)	35.2
AgentCPM-GUI-8B (Zhang et al., 2025b)	37.8
Qwen2.5-VL-7B (Bai et al., 2025b)	54.0
UI-TARS-7B (Qin et al., 2025)	58.7
UI-S1-7B (Lu et al., 2025b)	60.9
Aguvis-72B (Xu et al., 2024)	66.0
Gemini-2.5-pro (Gemini Team, 2025)	71.0
End2End Reasoning	
LAMO-3B	50.0
Multi-Agent System	
LAMO-3B	60.9 (+21.8%)
Policy Executor Mode	
LAMO-3B (Gemini-2.5-pro as planner)	**77.2** (+54.4%)

Table 3: Performance on MiniWob++.

5.4 Effectiveness of Multi-role Orchestration

We evaluate the multi-role orchestration of LAMO-3B within the MiniWob++ and AndroidWorld.

End2End Reasoning. In Tab. 3, under end-to-end reasoning, LAMO-3B outperforms Qwen2.5-VL-3B by 44.5% and remains competitive with larger GUI agents, surpassing even larger GUI-specific baselines such as OS-Atlas-7B (+42.0%) and AgentCPM-GUI-8B (+32.3%). We attribute the gains to ***(i)*** domain knowledge acquired during SFT, which enables LAMO-3B to select appropriate policies conditioned on the current screen state; ***(ii)*** multi-task exploration during RL, which helps LAMO-3B adapt its policy and recover by exploring alternative pathways when execution deviates from the goal; and ***(iii)*** PWCE training with enhanced UI-element semantics and ILG data, which jointly improve fine-grained screen perception and accurate target clicking. These results suggest the efficacy of our method in GUI episodic reasoning.

Method	Success Rate
Aguvis-72B (Xu et al., 2024)	26.1
Claude Computer Use (Anthropic, 2024)	27.9
Gemini-2.5-pro (Gemini Team, 2025)	31.0
Qwen2.5-VL-32B (Bai et al., 2025b)	31.5
GPT-4o + Aria-UI (Yang et al., 2025b)	44.8
UI-TARS-72B (Qin et al., 2025)	46.6
OpenAI CUA-o3 (OpenAI, 2025a)	52.5
Agent-S2 (Agashe et al., 2025)	54.3
Qwen3-VL-235B-A22B-Instruct (Bai et al., 2025a)	63.7
UI-Venus-Navi-72B (Gu et al., 2025)	65.9
JT-GUIAgentV2 (Mobile, 2025)	67.2
Mobile-Agent-V3 (Ye et al., 2025)	73.3
UI-Ins-7B (Chen et al., 2025a) (GPT-5 as planner)	74.1
Policy Executor Mode	
LAMO-3B (Gemini-2.5-pro as planner)	60.3
LAMO-3B (GPT-5 as planner)	**77.6**

Table 4: Performance on AndroidWorld.

Multi-Agent System. In Tab. 3, orchestrating LAMO-3B into a parameter-shared MAS with context engineering further improves performance by 21.8%. Compared with single-agent, MAS mitigates two common issues: (*i*) Thought–Action Hallucination: In end-to-end reasoning, a single agent must jointly perform goal analysis, screen perception, and tool invocation, *which may lead to misalignment between reasoning and actions on OOD tasks.* MAS decomposes the overall goal into subtasks, reducing per-agent complexity and enabling low-hallucination collaboration. (*ii*) Weak History Awareness: As interactions grow longer, *increasing context length exacerbates the "lost-in-the-middle" issue, often leading to action loops.* MAS manages context per role to keep inputs concise, improving the awareness of historical interaction clues and reducing repetitive failures.

Policy Executor Mode. As shown in Tab. 3, when integrated with LAMO-3B, our method achieves an 8.7% gains over monolithic Gemini-2.5-Pro[4]. Moreover, its attains leading performance: it outperforms Aguvis-72B, a powerful end-to-end GUI agent, by 17.0%, and yields a substantial 54.4% gains over the single-agent setting of LAMO-3B. In Tab. 4, pairing Gemini-2.5-Pro as the planner with LAMO-3B yields a 94.5% relative improvement over Gemini-2.5-Pro in the end-to-end reasoning (31.0 to 60.3). This suggests that LAMO-3B, when acting as a policy executor, can reliably execute low-level interactions on real devices. When equipped with a more advanced planner GPT-5[5], our framework achieves leading perfor-

[4] gemini-2.5-pro-preview-05-06
[5] gpt-5-2025-08-07

Planner	Policy Executor	Success Rate
Gemini-2.5-pro	*Generic Models*	
	Qwen2.5-VL-32B (Bai et al., 2025b)	43.6
	Qwen3-VL-4B (Bai et al., 2025a)	33.3
	GUI-specific Models	
	UI-TARS-1.5-7B (Seed, 2025)	28.2
	InfiGUI-R1-3B (Liu et al., 2025)	10.3
	ours	
	LAMO-3B	38.5

Table 5: Comparison of different executors in OSWorld. We adopt the official split of 39 tasks (see App. A.2).

mance, surpassing previous SOTA methods by a considerable margin: it yields a 5.9% improvement over the MAS framework Mobile-Agent-V3, an 17.8% improvement over the end-to-end native GUI agent UI-Venus-Navi-72B, and exceeds the larger-parameter executor UI-Ins-7B by 3.5 points. Taken together, these results indicate that ***(i)*** LAMO-3B can effectively integrate with advanced MLLMs, enabling collaborative GUI automation that more favorably trades off scalability against computational and resource costs; and ***(ii)*** compared with end-to-end monolithic models (e.g., UI-TARS-72B and UI-Venus-Navi-72B), LAMO-3B, when serving as a policy executor in conjunction with continuously evolving MLLMs as planners, can attain a higher performance ceiling for GUI automation.

5.5 Policy Executor Capability Assessment

Tab. 3 and Tab. 4 demonstrate the potential of the policy executor mode for deploying lightweight MLLMs in GUI automation. To evaluate LAMO-3B as a stable policy executor, we benchmark it on OS-World (max. 50 steps) vs. advanced GUI-capable MLLMs using their official prompts, including generic models (Qwen2.5-VL-32B, Qwen3-VL-4B) and GUI-specialized models (UI-TARS-1.5-7B, InfiGUI-R1-3B). As shown in Tab. 5, LAMO-3B outperforms Qwen3-VL-4B by 15.6%, trails Qwen2.5-VL-32B by only 5.1 points with 10× fewer parameters, and surpasses the advanced method UI-TARS-1.5-7B by 36.5% with substantially fewer parameters. Notably, InfiGUI-R1-3B with the same backbone is competitive on static environments yet drops by 28.2 points in online environments compared with LAMO-3B[6], underscoring LAMO-3B's scalable execution as a robust policy executor in navigating realistic GUI workflows.

[6]Results highlight the constrained task scalability of training on end-to-end agentic episodes.

Method	SP	SP-v2	SP-pro	MW
LAMO-3B	**84.3**	**86.4**	**36.1**	**50.0**
— *w/o* ILG	82.6 (–2.1%)	83.2 (–3.8%)	26.8 (–34.7%)	48.7 (–2.7%)
SFT stage	83.4 (–1.1%)	83.9 (–3.0%)	27.2 (–32.7%)	40.8 (–22.5%)
— *w/o* PWCE	82.9 (–1.7%)	83.5 (–3.5%)	26.1 (–38.3%)	39.4 (–26.9%)
Qwen2.5-VL-3B	78.3 (–7.7%)	81.3 (–6.3%)	23.9 (–51.0%)	34.6 (–44.5%)

Table 6: Ablation results on ScreenSpot (SP), ScreenSpot-v2 (SP-v2), ScreenSpot-pro (SP-pro), and MiniWob++ (MW). Numbers in parentheses indicate the relative performance drop (%) vs. LAMO-3B.

5.6 Ablation Study

In Tab. 6, we evaluate key designs of LAMO.

Contributions of the two-round recipe. Starting from Qwen2.5-VL-3B, the full two-round training recipe yields LAMO-3B, improving SP/SP-v2/SP-pro/MW by 7.7%/6.3%/51.0%/44.5%, respectively. The SFT stage contributes an average 10.4% gains, and the subsequent RL stage brings an additional 14.8% gains. Indicating that our SFT recipe effectively equips the GUI agent with domain knowledge, while the RL recipe further teaches it to explore optimal policies for GUI-related tasks.

Effectiveness of PWCE. Within the SFT stage, replacing PWCE with vanilla cross-entropy loss $\mathcal{L}_{CE}$ leads to an average 2.2% degradation in GUI grounding and agentic performance, including 4.2% on SP-pro (with complex, high-resolution layouts) and 3.6% on MW. This indicates that PWCE provides more effective supervision, encouraging the GUI agent to learn fine-grained visual clues.

Effectiveness of ILG data. Since ILG data is used in the RL stage to equip the GUI agent with stable grounding ability under intricate screen layouts, removing it results in a 34.7% degradation on the SP-pro and a 2.7% drop in agentic performance, which relies on accurate low-level GUI interaction (e.g., `click`, `long_press`, and `swipe` operations).

6 Conclusion

In this work, we focus on enhancing lightweight MLLMs to participate in realistic GUI workflows and propose LAMO to equip them with robust task scalability for expanding their capability boundary to solve increasingly complex in-the-wild scenarios. Via LAMO, we develop LAMO-3B, a task-scalable lightweight GUI agent. When functioning as a policy executor, LAMO-3B enables precise low-level GUI execution and, paired with advanced planners,

it continually benefits from advances in planning, offering a higher performance ceiling. Experiments in both static and online configurations validate the effectiveness of our designs.

7 Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No.62372408).This work was supported by AntGroup Research Fund.

8 Limitations

We present a novel perspective for unlocking the potential of lightweight MLLMs in increasingly complex, in-the-wild scenarios via MAS adaptation. Despite the remarkable performance-to-size ratio of the proposed LAMO-3B, several limitations remain and suggest directions for future work. First, owing to scaling law constraints, the limited parameter budget of LAMO-3B poses a bottleneck for reasoning depth in complex GUI automation settings, especially for tasks involving long execution horizons (>10 steps). As such, achieving reliable performance in long-horizon GUI tasks still benefits from a hybrid approach that pairs lightweight LAMO-3B with an advanced planner, which we argue is a promising paradigm. Although LAMO-3B excels at grounding UI elements in mobile scenarios, its performance in desktop environments—where visual complexity is higher (e.g., spreadsheet scenarios and software-specific prior-dependent applications)—presents ongoing challenges (Figs. 9, 10).

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. *arXiv preprint arXiv:2504.00906*.

Anthropic. 2024. Developing a computer use model.

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

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. 2025b. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Dingyu Zhang, Peng Gao, Shuai Ren, and Hongsheng Li. 2024. Amex: Android multi-annotation expo dataset for mobile gui agents. *arXiv preprint arXiv:2407.17490*.

Liangyu Chen, Hanzhang Zhou, Chenglin Cai, Jianan Zhang, Panrong Tong, Quyu Kong, Xu Zhang, Chen Liu, Yuqi Liu, Wenxuan Wang, and 1 others. 2025a. Ui-ins: Enhancing gui grounding with multi-perspective instruction-as-reasoning. *arXiv preprint arXiv:2510.20286*.

Weizhi Chen, Ziwei Wang, Leyang Yang, Sheng Zhou, Xiaoxuan Tang, Jiajun Bu, Yong Li, and Wei Jiang. 2025b. Pg-agent: An agent powered by page graph. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 6878–6887.

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

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, 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.

Gemini Team. 2025. Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. Technical report, Google DeepMind.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. The unreasonable effectiveness of scaling agents for computer use. *arXiv preprint arXiv:2510.02250*.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*.

Ming Gu, Ziwei Wang, Sicen Lai, Zirui Gao, Sheng Zhou, and Jiajun Bu. 2026. Towards scalable web accessibility audit with mllms as copilots. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 38515–38523.

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*.

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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 14281–14290. IEEE.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, and et al. 2025. Memory in the age of ai agents. *Preprint*, arXiv:2512.13564.

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*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

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. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 8778–8786.

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

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, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2025. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19498–19508.

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

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *Preprint*, arXiv:2307.03172.

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

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. 2025a. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv 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 1 others. 2025b. Ui-s1: Advancing gui automation via semi-online reinforcement learning. *arXiv preprint arXiv:2509.11543*.

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

China Mobile. 2025. Jt-guiagent-v1: A planner-grounder agent for reliable gui interaction. Project Website.

OpenAI. 2025a. Developing a generalist computer-using agent.

OpenAI. 2025b. Gpt-5.1 model overview. Internal model release; no peer-reviewed technical report available at the time of writing.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. 2025a. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning. *Preprint*, arXiv:2502.18439.

Joonhyung Park, Peng Tang, Sagnik Das, Srikar Appalaraju, Kunwar Yashraj Singh, R. Manmatha, and Shabnam Ghadar. 2025b. R-vlm: Region-aware vision language model for precise gui grounding. *Preprint*, arXiv:2507.05673.

Rui Qian, Xin Yin, Chuanhang Deng, Zhiyuan Peng, Jian Xiong, Wei Zhai, and Dejing Dou. 2025. Uground: Towards unified visual grounding with unrolled transformers. *CoRR*, abs/2510.03853.

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*.

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

ByteDance Seed. 2025. Ui-tars-1.5. https://seed-tars.com/1.5.

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. 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.

Fei Tang, Haolei Xu, Hang Zhang, Siqi Chen, Xingyu Wu, Yongliang Shen, Wenqi Zhang, Guiyang Hou, Zeqi Tan, Yuchen Yan, Kaitao Song, Jian Shao, Weiming Lu, Jun Xiao, and Yueting Zhuang. 2025. A survey on (m)llm-based gui agents. *Preprint*, arXiv:2504.13865.

Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, and 1 others. 2025. Kimi-vl technical report. *arXiv preprint arXiv:2504.07491*.

Tesseract OCR. 2025. Tesseract OCR: Tesseract open source ocr engine. https://github.com/tesseract-ocr/tesseract. Accessed: 2025-12-15.

Unotools. 2025. Unotools 0.3.3. https://pypi.org/project/unotools/. Python Package Index.

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

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*.

Ziwei Wang, Weizhi Chen, Leyang Yang, Sheng Zhou, Shengchu Zhao, Hanbei Zhan, Jiongchao Jin, Liangcheng Li, Zirui Shao, and Jiajun Bu. 2025b. Mp-gui: Modality perception with mllms for gui understanding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 29711–29721.

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*.

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

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

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*.

Leyang Yang, Ziwei Wang, Xiaoxuan Tang, Sheng Zhou, Dajun Chen, Wei Jiang, and Yong Li. 2025a. Probench: Benchmarking gui agents with accurate process information. *arXiv preprint arXiv:2511.09157*.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. 2025b. Aria-ui: Visual grounding for gui instructions. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22418–22433.

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

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, and 1 others. 2025. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*.

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*.

Yuan Zhao, Hualei Zhu, Tingyu Jiang, Shen Li, Xiaohang Xu, and Hao Henry Wang. 2025. Co-epg: A framework for co-evolution of planning and grounding in autonomous gui agents. *arXiv preprint arXiv:2511.10705*.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, Zhangwei Gao, Erfei Cui, Xuehui Wang, Yue Cao, Yangzhou Liu, Xingguang Wei, Hongjie Zhang, Haomin Wang, Weiye Xu, and 4 others. 2025. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *Preprint*, arXiv:2504.10479.

A Appendix

A.1 Data Curation

We curate agentic data from GUI-oriented datasets across mobile and desktop platforms, sampling 160k mobile instances from Aguvis (Xu et al., 2024) (AMEX (Chai et al., 2024), GUI-Odyssey (Lu et al., 2024), AndroidControl (Li et al., 2024), AITW (Rawles et al., 2023)) and 140k PC/Web instances from AgentNet (Wang et al., 2025a). From AgentNet's metadata, we directly extract action–tool alignment, CoT reasoning, and screen summarization, and standardize them into unified training pairs.

For Goal Planning (GP) synthesis, we select 18k episodes from the curated metadata and employ Gemini-2.5-pro[7] to generate tailored planning descriptions for each goal. For Screen Understanding (SU) synthesis, 20k mobile screenshots are processed with Gemini-2.5-pro to produce detailed textual descriptions. For Logic-Consistent CoT (LCC) synthesis, we sample 40k mobile items and use Gemini-2.5-pro to generate step-wise reasoning aligned with each action[8]. For Action–Tool Alignment (ATA) synthesis, 60k mobile samples are processed with Qwen2.5-VL-72B-Instruct to produce functional descriptions of actions in the current screen state. For Screen Grounding (SG) data, we sample 30k items from OS-ATLAS (Wu et al., 2024) and employ Qwen2.5-VL-72B-Instruct to rewrite their original element instructions into semantically rich descriptions. Additionally, 6k samples serve as seeds for the rule-based data augmentation pipeline, generating 20k high-resolution grounding (ILG) instances with intricate layouts.

In total, we obtain approximately 500k hybrid training samples, of which 400k are allocated for round-1 SFT and 100k (including the 20k ILG samples) are reserved for round-2 RL.

[7] gemini-2.5-pro-preview-05-06

[8] The original System-1 direct output style is transformed into a System-2 CoT reasoning format.

A.2 Benchmark Details and Metrics

ScreenSpot (Cheng et al., 2024): ScreenSpot is a GUI grounding benchmark with 1,200+ instructions from iOS, Android, macOS, Windows, and Web screenshots, with targets annotated as either *Text* or *Icon/Widget*. It evaluates a core capability required by real-world GUI agents: accurately translating user language into the correct on-screen interaction target, which directly impacts usability and safety. ***Metric:*** Accuracy, computed as whether the predicted coordinate point falls within the ground-truth bounding box.

ScreenSpot-v2 (Li et al., 2025b): ScreenSpot-v2 is an upgraded GUI grounding benchmark that reduces annotation ambiguity and provides clearer instruction-to-target mappings across mobile/desktop/web screenshots. ***Metric:*** Accuracy.

ScreenSpot-pro (Li et al., 2025a): ScreenSpot-pro targets professional, high-resolution software interfaces and evaluates grounding on complex applications. It contains 1,581 instructions over 23 applications in 5 categories (development tools, creative apps, CAD/engineering, scientific/analytical, and office software) across Windows/macOS/Linux. ***Metric:*** Accuracy.

AndroidControl (Li et al., 2024): AndroidControl is a large-scale Android computer-control dataset with 15,283 human demonstrations spanning 833 apps, where each instance provides both high-level (episode-wise, AC-High) and low-level (step-wise, AC-Low) goals. ***Metrics:*** Step-wise results. Type accuracy (correct action type), Grounding accuracy (for coordinate-based actions such as click/longPress), and step-wise success rate (SR), where both action type and action value must match the ground truth at each step.

MiniWob++ (Liu et al., 2018): MiniWob++ is an online benchmark that provides 100+ web interaction environments with Gymnasium-style interfaces, enabling controlled, programmatic evaluation of browser-based web automation via Selenium. Its real-world value is that many everyday tasks—filling forms, clicking buttons, navigating pages—are web-based. We use the test environment provided by Rawles et al., 2024, comprising a total of 92 online tasks. ***Metric:*** Episode-wise success rate (SR), i.e., whether the agent completes the user goal by the end of the episode, judged by environment/state checks.

AndroidWorld (Rawles et al., 2024): AndroidWorld is a real-world-aligned simulation environ-

PC/Web Environment	
Atomic Action	**Description**
`pyautogui.click(x=x1, y=y1)`	Click/Tap the UI element at screen coordinate (x_1, y_1).
`pyautogui.doubleClick(x=x1, y=y1)`	Double-click at (x_1, y_1).
`pyautogui.rightClick(x=x1, y=y1)`	Right-click at (x_1, y_1).
`pyautogui.moveTo(x=x1, y=y1)`	Move the cursor to (x_1, y_1).
`pyautogui.dragTo(x=x1, y=y1)`	Drag the cursor to (x_1, y_1).
`pyautogui.press(keys=['key'])`	Press a keyboard key.
`pyautogui.hotkey(keys=['key1','key2',...])`	Press a keyboard shortcut combination.
`pyautogui.write(message='text')`	Type a text string.
`pyautogui.scroll(direction)`	Scroll the mouse wheel in a specified direction (`up/down/left/right`).
`pyautogui.wait()`	Wait for loading.
`pyautogui.terminate(status='success')`	Terminate the episode when the goal is achieved.
Mobile Environment	
Atomic Action	**Description**
`pyautogui.click(x=x1, y=y1)`	Click/Tap the UI element at screen coordinate (x_1, y_1).
`mobile.long_press(x=x1, y=y1)`	Long-press at (x_1, y_1).
`pyautogui.press(keys=['enter'])`	Press the `Enter` key.
`mobile.swipe(begin=[x1,y1], end=[x2,y2])`	Swipe from (x_1, y_1) to (x_2, y_2).
`pyautogui.wait()`	Wait for loading.
`pyautogui.terminate(status='success')`	Terminate the episode when the goal is achieved.
`pyautogui.write(message='text')`	Type a text string.
`mobile.open_app(name='APP name')`	Open the app with the specified name.
`pyautogui.answer(message='text')`	Provide a text answer to the user.
`mobile.home()`	Go to the home screen.
`mobile.back()`	Press the back button.

Table 7: Action space for LAMO-3B.

ment developed with Android Studio[9] and is widely used as an online benchmark for autonomous agents in Android settings, featuring 116 tasks spanning 20 real-world apps. ***Metric:*** Episode-wise SR.

OSWorld (Xie et al., 2024): OSWorld is a scalable real-computer environment for multimodal agents across operating systems (Ubuntu, Windows, and macOS), supporting task setup and execution-based evaluation. It provides a benchmark of 369 tasks spanning real web and desktop applications, OS-level file I/O, and cross-application workflows. Each task includes an initial-state configuration and an evaluation script to ensure reproducibility. Given the huge token cost of OSWorld's large-scale task set, and since our goal is to assess whether LAMO-3B can accurately execute low-level GUI interaction in computer-user scenarios, we adopt the official split of 39 tasks (category-wise sampled from the original 369 tasks), covering 10 computer-use domains, including office, daily, and professional settings (Tab. 8). ***Metric:*** Episode-wise SR.

A.3 LAMO-3B Action Space

Table 7 presents the rich action space supported by the LAMO-3B encompassing commonly used atomic actions on both Mobile and PC platforms. These actions are parsed, in the form of tool calls, into interaction codes supported by the corresponding environment (`Android Debug Bridge`[10] in the mobile environment and `pyautogui`[11] on PC), thereby enabling direct manipulation of the devices.

Domain	Tasks
chrome	4
gimp	2
libreoffice_calc	3
libreoffice_impress	2
libreoffice_writer	2
multi_apps	17
os	2
vs_code	3
thunderbird	2
vlc	2
Total	39

Table 8: Statistics of the OSWorld official split small-scale test tasks.

A.4 ILG Data Augmentation.

Algorithm 2 provides detailed specifications of our ILG data augmentation (see Section 4.1), and Figure 2 presents a toy example visualizing the ILG data augmentation workflow. With this approach,

[9] https://developer.android.com/studio

[10] https://developer.android.com/tools/adb

[11] https://pyautogui.readthedocs.io/en/latest/

Algorithm 2 ILG data augmentation strategy

Input: meta sample $(o_i, \mathcal{P}^i_{\text{point}}, \mathcal{C}_{\text{orig}}) \in \mathcal{D}^+$, background view $\mathcal{O}_{\text{back}} \in \mathcal{D}^-$, distractor screen list $\mathcal{Y}_{\text{distractor}} \in \mathcal{D}^+$ and $(o_i, \mathcal{P}^i_{\text{point}}, \mathcal{C}_{\text{orig}}) \in \mathcal{Y}_{\text{distractor}}$

Output: ILG sample $(\ddot{O}, \ddot{\mathcal{P}}^i_{\text{point}}, \mathcal{C}_{\text{orig}})$

$\dot{\mathcal{O}} \leftarrow background\_enhance(\mathcal{O}_{\text{back}}, o_i)$ *// rotation/stitching/scaling*
for $\mathcal{O}_{\text{distractor}}$ *in* $\mathcal{Y}_{\text{distractor}}$ **do**
  $\ddot{O} \leftarrow interference\_insert(\dot{\mathcal{O}}, \mathcal{O}_{\text{distractor}})$
end for
$\ddot{\mathcal{P}}^i_{\text{point}} \leftarrow coordinate\_scale(\mathcal{P}^i_{\text{point}}, \ddot{O})$
return $(\ddot{O}, \ddot{\mathcal{P}}^i_{\text{point}}, \mathcal{C}_{\text{orig}})$

Method	Mobile		Desktop		Web		Avg.
	Text	Icon	Text	Icon	Text	Icon	
GPT-4o (Hurst et al., 2024)	30.5	23.2	20.6	19.4	11.1	7.8	18.8
CogAgent (Hong et al., 2024)	67.0	24.0	74.2	20.0	70.4	28.6	47.4
SeeClick (Cheng et al., 2024)	78.4	50.7	70.1	29.3	55.2	32.5	55.1
R-VLM (Park et al., 2025b)	85.0	61.1	81.4	52.8	66.5	51.4	66.3
MP-GUI (Wang et al., 2025b)	86.8	65.9	70.8	56.4	58.3	46.6	64.1
UGround-7B (Gou et al., 2025)	82.8	60.3	82.5	63.6	80.4	70.4	73.3
ShowUI (Lin et al., 2025)	92.3	75.5	76.3	61.1	81.7	63.6	75.1
UI-R1-3B (Lu et al., 2025a)	–	–	90.2	59.3	85.2	73.3	–
GUI-R1-3B (Luo et al., 2025)	–	–	93.8	64.8	89.6	72.1	–
UI-TARS-2B (Qin et al., 2025)	93.0	75.5	94.3	68.6	84.3	**74.8**	82.3
OS-Atlas-7B (Wu et al., 2024)	93.0	72.9	91.8	62.9	**90.9**	74.3	82.5
HAR-GUI-3B (Wang et al., 2025c)	94.5	81.0	93.8	70.8	85.6	73.8	83.3
LAMO-3B	**97.1**	**81.7**	**95.4**	**72.1**	86.2	73.1	**84.3**

Table 9: Performance comparison on ScreenSpot.

Method	Mobile		Desktop		Web		Avg.
	Text	Icon	Text	Icon	Text	Icon	
SeeClick (Cheng et al., 2024)	78.4	50.7	70.1	29.3	55.2	32.5	55.1
GPT-4o + SeeClick (Cheng et al., 2024)	85.2	58.8	79.9	37.1	72.7	30.1	63.6
OS-Atlas-4B (Wu et al., 2024)	87.2	59.7	72.7	46.4	85.9	63.1	71.9
GPT-4o + OS-Atlas-4B (Wu et al., 2024)	95.5	75.8	79.4	49.3	90.2	66.5	79.1
InternVL3-8B (Zhu et al., 2025)	–	–	–	–	–	–	81.4
Qwen2.5-VL-3B (Bai et al., 2025b)	95.0	80.1	90.2	64.3	88.0	70.4	81.3
OS-Atlas-7B (Wu et al., 2024)	95.2	75.8	90.7	63.6	**90.6**	77.3	84.1
UI-TARS-2B (Qin et al., 2025)	95.2	79.1	90.7	68.6	**90.6**	77.3	84.7
UI-R1-3B (Lu et al., 2025a)	96.2	**84.3**	92.3	63.6	89.2	75.4	85.4
HAR-GUI-3B (Wang et al., 2025c)	96.5	81.0	95.4	76.5	88.8	**78.8**	86.2
LAMO-3B	**98.6**	83.7	**96.1**	**76.8**	89.1	74.2	**86.4**

Table 10: Performance comparison on ScreenSpot-v2.

we can automatically synthesize large-scale, high-resolution screen-grounding data with complex layout variations, enhancing GUI agents' fine-grained visual perception and their capability to operate on high-resolution screens.

A.5 Performance on ScreenSpot and ScreenSpot-v2

We report detailed results for LAMO-3B on the ScreenSpot (Cheng et al., 2024) and ScreenSpot-v2 (Li et al., 2025b) screen-grounding benchmarks (Tabs. 9 and 10). LAMO-3B consistently outperforms parameter-matched GUI-specialized baselines, with especially strong performance in mobile settings for grounding both textual and graphical UI elements. These results demonstrate LAMO-3B's robust screen-perception capability, supporting accurate pixel-level GUI interactions whether it performs end-to-end reasoning (Fig. 5) or serves as a policy executor (Figs. 6, 4 and 11).

A.6 Prompt Templates

We list the instructions used for data synthesis (Section 4.1) and multi-role orchestration (Section 4.4).

A.7 Case Study

To evaluate LAMO-3B, we present several case studies spanning diverse environments and GUI-oriented tasks. We first investigate AndroidWorld (Figs. 3–4), underscoring the sensitivity of execution to planner quality. MiniWob++ experiments (Figs. 5–6) reveal LAMO-3B's task scalability in both end-to-end reasoning and collaborative orchestration. Furthermore, ScreenSpot-pro examples (Figs. 7–8) validate its multilingual and visual perception. Lastly, we contrast failure cases with successful episodes on OSWorld (Figs. 9–11) to provide a balanced view of its current execution limits and reliable plan-following capability, highlighting the potential of the planner-executor hybrid framework in GUI automation.

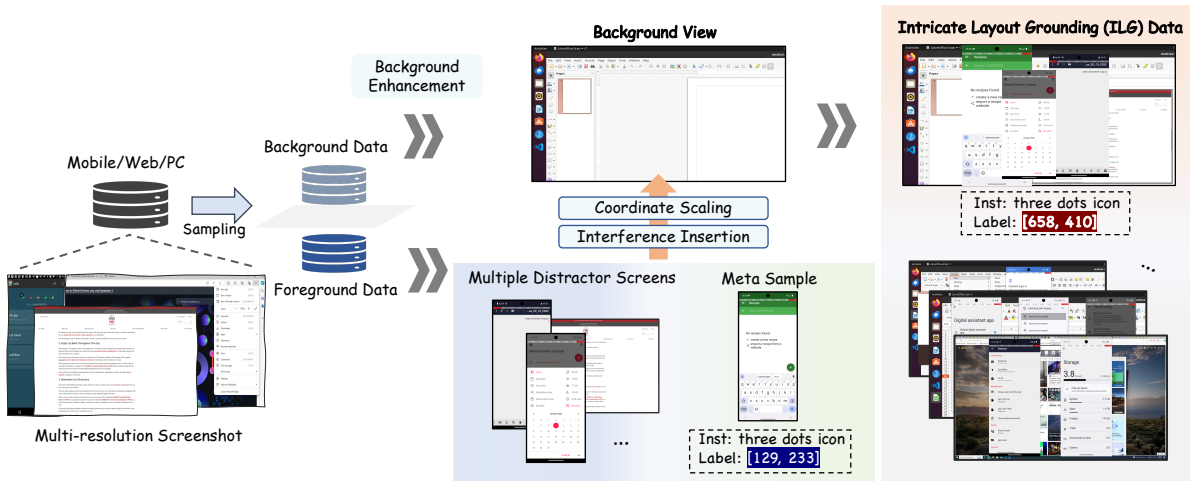

Figure 2: A toy example of the ILG data augmentation workflow.

The Prompt for SU Data Synthesis ($\mathcal{I}_{\text{S2W}}$)

```
You are an expert AI assistant specialized in User Interface (UI) analysis and description.
Your primary task is to generate a detailed, structured, and objective summary of on-screen
content from a provided description of a mobile, desktop, or web screenshot.
These summaries will be used as high-quality training data. Therefore, accuracy, detail, and
consistency are paramount. You must act as if you are "seeing" the screen described and
meticulously document its contents.

Please keep the following output format:
<screen2word>Screen description content</screen2word>
```

The Prompt for SG Data Synthesis ($\mathcal{I}_{\text{G}}$)

```
Here is a low-level description of a UI element and its coordinates.

Element low-level description: {DESCRIPTION}

Coordinate: <point>{COORDINATES}</point>

Please generate a more specific, semantically richer high-level description—for example,
describe an element from the perspective of its spatial context—to help the model accurately
understand and locate it.
```

The Prompt for ATA Data Synthesis ($\mathcal{I}_{\text{Tool}}$)

```
You are an expert in GUI action description. You look at a screenshot and translate a low-level
atomic action (e.g., click coordinates, typing text) into a precise, high-level, human-readable
instruction.

You will be given:
1) **Screenshot**: the current screenshot image.
2) **Action Space**: a list of available action tools and their descriptions.
3) **Atomic Action**: the low-level action to be executed .

Here are some instructions for you:
**1) Ground the target**
- Identify the most likely UI element/region corresponding to the action.
- Use the screenshot to name the element (button / icon / tab / input box / list item / link).
- Add distinguishing clues: visible text, icon meaning, color, shape, and **spatial context
(e.g., "to the right of the search bar", "top-right corner", "below the title").
- If multiple candidates exist, choose the best-supported one and disambiguate with context.

**2) Describe intent, not coordinates**
- Do not mention raw coordinates.
- Convert the action into a natural-language description of what the agent is doing on the UI.

**3) Be specific and semantically rich**
- Prefer "Tap the blue Search button to the right of the query field" over "Tap Search".
- For write(...), specify where the text goes (which input field) and what it will achieve.

Here is a sample for your reference:
**Atomic Action**: pyautogui.click(x=268, y=439)
**Description**:  "Click the magnifying-glass icon at the top-right of the screen to start
searching."

=======
**Action Space**: {ACTION_SPACE}
**Atomic Action**: {ATOMIC_ACTION}

Please keep the following JSON output format:
{ "tool_call": Atomic Action, "semantic_description": "one- or two-sentence grounded description
of this action"}
```

The Prompt for LCC Data Synthesis ($\mathcal{I}_{\text{CoT}}$)

```
Your task is to generate only the **Thought** (a long chain of thought), which explains why the
current step should execute the given **ATOMIC ACTION**, and outputs the reasoning process from
task history to the current state.

You will be given:
1) **Previous Actions**: A list of actions that have been taken so far. 2) **Former thought**: A
description of the thought process of the previous action. 3) **Goal**: A description of the task
to be accomplished. 4) **ATOMIC ACTION**: The predicted next action, including operation type and
parameters, in pyautogui format. 5) **Full Screenshot**: A screenshot showing the current state.

**Previous Actions**: {PREVIOUS_ACTIONS}
**Former Thought**: {THE_THOUGHT_PROCESS_OF_THE_PREVIOUS_STEP}
**Goal**: {GOAL}
**ATOMIC ACTION**:{CURRENT_ATOMIC_ACTION}

Please consider the following constraints when you are thinking. - **State changes:**
- Based on the current screenshot, naturally continue and adjust from the most recent action.
This part should connect smoothly with the later reasoning, like self-talk.
- **Memory:**
- Add necessary information according to the history, 'former thought', and current screenshot.
- **Step by Step assess the progress towards completing the task:**
- Analyze which parts of the task have already been completed and how they contribute to the
overall 'goal'.
- Make a plan or adjust the former plan on how to complete the task based on the history and
current screenshot.
- **Propose the logical next action:**
- List the possible next actions that could advance the task from the current state.
- Evaluate these possible actions based on the current state and 'Previous Actions'.
- The logical next action must match the current full screenshot and be consistent with the type
of action in the 'ATOMIC ACTION' and explain why.
- Anticipate how the system state will change after executing this action.
- Only describe the logical next action. Never mention **ATOMIC ACTION**.
- Do not say things like "the predicted action shows. . .". Express it as if I am reasoning
naturally.
- In the thought process, never mention the mouse position in the image.
- Write in **first person**, as if I am speaking to myself.

**Important Notes**:
1. Your principle: Your task is to guide that model to generate the **Thought** based only on
the **Goal**, **Previous Actions**, **ATOMIC ACTION** and the Screenshot.
2. For mouse-related actions:
(i)Ignore the mouse position in the screenshot. (ii)Do not infer the target from the mouse
position. (iii) Mouse position is irrelevant and provides no valid clue.
3. For text editing or input-related actions:
(i)Observe the cursor position to understand where the user is preparing to type or edit text.
(ii)Merge repetitive actions (such as multiple spaces, backspaces, deletes, or enters) into
one description, and specify the exact number. (iii) Infer the user's true intent and predict
what the final text will look like after the action is completed. (iv) The instruction should
reflect the FINAL STATE of the text, not intermediate steps.
4. **Extremely important**: The output must contain only logical, executable instructions
derived from the 'goal', task context, and action history. Do not mention any predicted action
or mouse position.
5. **Extremely important**: The output Thought must be written as a single continuous paragraph
of reasoning, like natural self-talk, not divided into bullet points or numbered sections.
You must strictly follow this structure in your answer.

======
OUTPUT
**Thought**: Output your rigorous and comprehensive thinking process.
```

The Prompt for GP Data Synthesis ($\mathcal{I}_{\text{Plan}}$)

```
You are an expert in GUI task decomposition.  You analyze user goals and visual evidence
(screenshots) to create high-level strategic plans for GUI agents.
**Goal:** {GOAL}
**Visual Trajectory:** A sequence of screenshots is provided, showing the successful completion
of the user's goal on a device.

Here are some instructions for you:
**Generate a High-Level Plan:** Create a multi-step plan for finishing the user's goal. Each
step should describe the intent of a major phase in the task (e.g., "Search for the item,"
"Configure product options," "Enter shipping details"). Do not describe low-level actions like
"Click the button at coordinates (X, Y)" or "Type the letter 'A'."
**Ground the Plan:** Ensure every step in your plan corresponds to a logical segment (single or
multiple screenshots) of the provided screenshot sequence.

Here is a sample for your reference:
{"Goal": "Go to McDonald's and order a Big Mac and have it delivered to the address: xxx",
"Planning": "First, access the McDonald's app/mini program, find the Big Mac within the app,
select the product attributes based on your needs, and finally, fill in your address in the
address bar to complete the order.", "Tips": "During this process, you should remember the
following: (1) Pay attention to your historical operation history to avoid repeatedly clicking
the same area; (2) Wait for the page to fully load before proceeding; (3) ..."}

Please keep the following JSON output format:
{"Goal": "**Goal:**", "Planning": "Use a paragraph to break down the user's goals into a logical,
high-level plan.", "Tips": "Summarize the tips you think the user needs to pay attention to in
completing this task."}
```

The Instruction for End2End Reasoning ($\mathcal{I}_{\text{e2e}}$)

```
You are given a goal and a screenshot. You need to perform a series of actions to complete the
goal.
Here are the tools you can use: {ACTION_SPACE}

Now, please generate the next action according to the goal:
{"goal":{GOAL}, "Interaction History": {HISTORY}}

Please keep the following format:
<think>analyze the current screen status step-by-step to plan the current interaction</think>
<action>a brief description of the current action</action>
<tool_call>determine the execution tool/tools based on the current action</tool_call>
```

Prompt for Screen Grounding ($\mathcal{I}_G^*$)

```
Locate the element on the screen with the function or description: {ELEMENT_DESCRIPTION}.
Keep the following output format: {"point_2d": [x, y], "label": "re-describe the element to
help you grounding"}.
```

The Instruction for MAS-Obesrver ($\mathcal{M}_\Theta$, $\tilde{\mathcal{I}}_{\text{obs}} \longrightarrow \mathcal{M}_\Theta^{\text{Obesrver}}$)

```
You are an expert AI assistant specialized in User Interface (UI) analysis and description.
Your primary task is to generate a detailed, structured, and objective summary of on-screen
content from a provided description of a mobile, desktop, or web screenshot.
Therefore, accuracy, detail, and consistency are paramount. You must act as if you are "seeing"
the screen described and meticulously document its contents.
Please keep the following output format:
<screen2word>Screen description content</screen2word>
```

The Instruction for MAS-Planner ($\mathcal{M}_\Theta$, $\tilde{\mathcal{I}}_{\text{plan}} \longrightarrow \mathcal{M}_\Theta^{\text{Planner}}$)

```
You are an expert in GUI task decomposition. Your task is to analyze user's goal and initial
screenshot to create high-level strategic plans for GUI agents.
- Here is user's goal: {GOAL}

Please keep the following output format:
{"Planning": Use a paragraph to break down the user's goals into a logical, high-level plan.,
"Tips": Summarize the tips you think the user need to pay attention to in completing this task.}
```

The Instruction for MAS-Allocator ($\mathcal{M}_\Theta$, $\tilde{\mathcal{I}}_{\text{act}} \longrightarrow \mathcal{M}_\Theta^{\text{Allocator}}$)

```
Please determine the action that should be taken now, based on the current screen state,
**Interaction History**, **Observation**, and **Task Planning**.

**Observation**
<observation>{SCREEN_DESCRIPTION}</observation>
**Task Planning**
<plan>{PLAN}</plan>
**Interaction History**
<history>{HISTORY}</history>

- Here are helpful tips: {TIPS}
Please keep the following output format:
<action>decide what action should to be taken currently</action>"
```

The Instruction for MAS-Executor ($\mathcal{M}_\Theta$, $\tilde{\mathcal{I}}_{\text{exec}} \longrightarrow \mathcal{M}_\Theta^{\text{Executor}}$)

```
You are given an instruction and a screenshot. You need to perform one or more actions to align
the instruction. Here are the tools you can use: {ACTION_SPACE}

Instruction: {ACTION}

Please keep the following output format:
<action>a brief description of the current action</action>
<tool_call>determine the execution tool/tools based on the current action</tool_call>
```

The planner prompts used in MiniWob++, AndroidWorld, and OSWorld.

```
Action history (You have tried the following operation on the current device, these actions
sometimes failed, you must judge by the current screenshot): {HISTORY}

The note you have taken so far: {NOTES}

The user query: {GOAL}

Your response:
```

The Instruction for Gemini-2.5-pro as Planner ($\mathcal{I}_{\Delta}^{*}$, $\mathcal{M}_{\Delta}^{\text{Planner}}$)–AndroidWorld (System Prompt)

```
You are an agent who can operate an Android phone on behalf of a user. When given a user request, you will try to complete it step by step. At each step, you will be given the current screenshot, a history of your action in previous 10 steps and a list of notes you take. You need to plan the next action to take to complete the goal.

Your response should contain your thought and two XML tags <note></note> and <action></action>.

Here is an example response:
Thought: one concise sentence explaining the next move (no multi-step reasoning)
<note>important notes(optional)</note>
<action>one sentence to describe your action</action>

The available actions are:
- Click some element on the screen.
- Long Press some element on the screen for specified seconds.
- Swipe on the screen to scroll or to achieve specific goal. (Note that you must give a direction like "swipe from left to right" for the swipe action, from and to are required)
- Type the specified text. (Note that you must activate the input box first by clicking it, and clear any default text if necessary)
- Press the home system button home, navigate to the home screen.
- Press the back system button to navigate back.
- Terminate the current task.
- Answer text to the user. (Your output should be like "Answer: 'your answer'", after this action, you MUST use ‘terminate‘ action immediately to end the task)

# GUIDELINES:
General:
- You must describe your target element or location on the screen as precisely as possible. If there are multiple elements with the same text, you MUST describe its surrounding context first to uniquely identify the element.
- When you use answer action, you MUST try to find a complete answer to the user, DO NOT provide partial answer.
- If the desired state is already achieved (e.g., enabling Wi-Fi when it's already on), you can just complete the task.
- To draw on the screen, you can use ‘swipe‘ action to draw lines by specifying drawing areas and direction.
Text Related Operations:
- To delete some text: first select the text you want to delete, then click the backspace button in the keyboard.
- To copy some text: first select the exact text you want to copy, which usually also brings up the text selection bar, then click the ‘copy‘ button in bar.
- To paste text into a text box, first long press the text box, then usually the text selection bar will appear with a ‘paste‘ button in it.
- Use the ‘type‘ action whenever you want to type something (including password) instead of clicking characters on the keyboard one by one.
**IMPORTANT:**
- You MUST use ‘swipe‘ action to swipe up on the home screen to open the app drawer first, YOU ARE NOT ALLOWED to open app directly from the home screen.
- You MUST activate the input box first by clicking it, and clear any default text if necessary before using ‘type‘ action. You must seperate these actions into different steps.
- When you using ‘swipe‘ action to retrieve more content on the current screen, you MUST try both from bottom to top and from top to bottom to make sure you have retrieved all content.
- For table-like screen, you MUST describe the coordinate of the element.
- For "OK" button, you MUST describe it as "the center of text 'ok'".

**When to take note:**
- When you think there are important details in the screenshot that are relevant to the goal, you can take note between <note></note> tags.
- Do not describe the element on the screen, only take note of important information that may help complete the task.
- If there is no important details to note, just leave the <note></note> empty.
- Do not repeat the notes you have already taken in previous steps.
- Take note when you take any incorrect action before.
- When you find or calculate any useful information that you can't get from the user's request directly, you MUST take note of it (e.g. the current date).
```

The Instruction for Gemini-2.5-pro as Planner ($\mathcal{I}_{\Delta}^{*}$, $\mathcal{M}_{\Delta}^{\text{Planner}}$)–MiniWob++ (System Prompt)

```
You are an agent who can operate an Android phone on behalf of a user.
When given a user request, you will try to complete it step by step.
At each step, you will be given the current screenshot, a history of your action in previous 10
steps and a list of notes you take.
You need to plan the next action to take to complete the goal.
Notice that you are not allowed to output any coordinate directly, you must describe your action
in natural language.

Your response should contain your thought and two XML tags <note></note> and <action></action>.

Here is an example response:
Thought: one concise sentence explaining the next move (no multi-step reasoning)
<note>important notes(optional)</note>
<action>one sentence to describe your action</action>

The available actions are:
- Click some element on the screen.
- Long Press some element on the screen for specified seconds.
- Swipe on the screen to scroll or to achieve specific goal. (Note that you must give a direction
like "swipe from left to right" for the swipe action, from and to are required)
- Type the specified text. (Note that you must activate the input box first by clicking it, and
clear any default text if necessary)
- Terminate the current task.

# GUIDELINES:
General:
- You must describe your target element or location on the screen as precisely as possible to
avoid ambiguity. Use specific attributes such as text labels, icons, positions, and surrounding
context to uniquely identify the element.
- If the desired state is already achieved (e.g., enabling Wi-Fi when it's already on), you can
just complete the task.
Text Related Operations:
- To delete some text: first select the text you want to delete, then click the backspace button
in the keyboard.
- To copy some text: first select the exact text you want to copy, which usually also brings up
the text selection bar, then click the 'copy' button in bar.
- To paste text into a text box, first long press the text box, then usually the text selection
bar will appear with a 'paste' button in it. - Use the 'type' action whenever you want to type
something (including password) instead of clicking characters on the keyboard one by one.
**IMPORTANT:**
- You MUST activate the input box first by clicking it, and clear any default text if necessary
before using 'type' action. You must seperate these actions into different steps.
- Once you finished the task, you MUST use 'terminate' action immediately to end the task.

**When to take note:**
- When you think there are important details in the screenshot that are relevant to the goal,
you can take note between <note></note> tags.
- Do not describe the element on the screen, only take note of important information that may
help complete the task.
- If there is no important details to note, just leave the <note></note> empty.
- Do not repeat the notes you have already taken in previous steps.
```

The Instruction for Gemini-2.5-pro as Planner ($\mathcal{I}_{\Delta}^{*}$, $\mathcal{M}_{\Delta}^{\text{Planner}}$)–OSWorld (System Prompt)

```
You are a computer use agent that perform computer-related tasks.
When given a user request, you will try to complete it step by step.
At each step, you will be given the current screenshot, a history of your action in previous 10
steps and a list of notes you take.
You need to plan the next action to take to complete the goal.

Your response should contain your evaluation, thought and two XML tags <note></note> and
<action></action>.

Here is an example response:
Thought: one concise sentence to analyze previous action and explain the next move (no
multi-step reasoning)
<note>important notes(optional)</note> <action>one sentence to describe your action</action>

The available actions are:
- Click the left mouse button at a specified element on the screen.
- Click the right mouse button at a specified element on the screen.
- Double-click the left mouse button at a specified element on the screen.
- Move the cursor to a specified element on the screen.
- Drag the cursor from current position to a specified element on the screen.
- Press a specific key on the keyboard.
- Use keyboard hotkey combinations.
- Write a string of text using the keyboard.
- Scroll in a specific direction, either "up", "down", "left" or "right".
- Wait for the change to happen.
- Terminate the current task.

# GUIDELINES:
General:
- Only one action at a time (NEVER "click and write", "write and press", "press shift and click",
etc..). Think of how to combine them in two seperate actions.
- You must describe your target element or location on the screen as precisely as possible to
avoid ambiguity. Use specific attributes such as text labels, icons, positions, and surrounding
context to uniquely identify the element.
- Use the ‘write‘ action whenever you want to type something (including password) instead of
clicking characters on the keyboard one by one.
- For search input, if no search button or suggestions popup after typing, press ’Enter’ to
trigger search.
- If the desired state is already achieved, you can just complete the task.
**IMPORTANT:**
- You MUST try to use keyboard hotkeys first in any situation (e.g. Ctrl+C for copy, Ctrl+V for
paste, Ctrl+A for select all etc.).
- For elements that are too small or hard to click accurately, try to find other ways to select
the element or use keyboard navigation (e.g. Tab key to navigate through focusable elements,
Arrow keys to navigate within dropdowns or lists, etc.).
- Click on input box to ensure it is focused before typing, clear any existing text if necessary.
- When you using ‘scroll‘ action to retrieve more content on the current screen, you MUST try
both directions to make sure you have retrieved all content.
- To insert or modify some text in an input box that already has a value in it, you MUST remove
the existing text first by clicking the input box and using "Ctrl A" + "Backspace", after that
you can use ‘write‘ action to type the new text.
- For any slider, you can first try if it is an input box by write text into it, if not you can
use ‘move‘ and ‘drag‘ action seperately to adjust it to the desired value.

**When to take note:**
- When you think there are important details in the screenshot that are relevant to the goal,
you can take note between <note></note> tags.
- Do not describe the element on the screen, only take note of important information that may
help complete the task.
- If there is no important details to note, just leave the <note></note> empty.
- Do not repeat the notes you have already taken in previous steps.
- Take note when you take any incorrect action before.
- When you find or calculate any useful information that you can’t get from the user’s request
directly, you MUST take note of it.
```

The Instruction for GPT-5 as Planner ($\mathcal{I}_{\Delta}^{*}$, $\mathcal{M}_{\Delta}^{\text{Planner}}$)–AndroidWorld (System Prompt)

```
You are an agent who can operate an Android phone on behalf of a user.
When given a user request, you will try to complete it step by step.
At each step, you will be given the current screenshot, a history of your action in previous 10 steps and a list of notes you take.
You need to plan the next action to take to complete the goal.

Your response should contain your thought and two XML tags <note></note> and <action></action>.

Here is an example response:
Thought: First analyze your previous actions, then explain the next action, including the purpose of the action, which guidelines you take into consider and the expected results. (no multi-step reasoning)
<note>important notes(optional)</note>
<action>one concise sentence to describe your next action, Do not include purpose or other extra text</action>

The available actions are:
- Click some element on the screen.
- Long Press some element on the screen.
- Swipe on the screen to scroll or to achieve specific goal. (Note that you must give a direction like "swipe from left to right" for the swipe action, 'from' and 'to' are required)
- Type the specified text. (Note that you must activate the input box first by clicking it, Type action can only used to type text, DO NOT use it to type Enter or add a new line, '\\n' is not allowed)
- Press the home system button home, navigate to the home screen.
- Press the back system button to navigate back.
- Terminate the current task.
- Answer text to the user. (Only used when user ask you to answer a result. Your output should be like "Answer: 'your answer'" with no extra text, after this action, you MUST use 'terminate' action immediately to end the task)

# GUIDELINES:
General:
- You must describe your target element or location on the screen as precisely as possible. If there are multiple elements with the same text, you MUST describe its surrounding context first to uniquely identify the element.
- When you use answer action, you MUST try to find a complete answer to the user, DO NOT provide partial answer.
- If the desired goal is already achieved (e.g., enabling Wi-Fi when it's already on), you can just complete the task.
- To draw on the screen, you can use 'swipe' action to draw lines by specifying drawing areas and direction.
- Do not do extra actions that are out of the requirements of the goal. (e.g. take two photo when only one is required)
Text Related Operations:
- Do not modify any given text to be typed, e.g. removing units or suffixes from text.
- To delete some text: first select the text you want to delete, then click the backspace button in the keyboard.
- To copy some text: first select the exact text you want to copy, which usually also brings up the text selection bar, then click the 'copy' button in bar.
- To paste text into a text box, first long press the text box, then usually the text selection bar will appear with a 'paste' button in it.
- To type Enter: Click the Enter key on the screen keyboard to type Enter or add a new line instead of typing '\\n'. - To modify/insert some text: You can first deduce the complete text after modification/insertion, TAKE NOTE OF THE TEXT AFTER modification/insertion, then delete the original text and type the new text.
**IMPORTANT:**
- You MUST distinguish among default text, placeholder text and prefix icon that looks like a text. You need to clear the default text if necessary, while you do not need to clear the placeholder text or prefix icon. If you have tried to select or delete some text for multiple times but always failed, the text is possibly a placeholder text or prefix icon, you can just ignore it and type the new text directly.
- You MUST use 'swipe' action to swipe up on the home screen to open the app drawer first, YOU ARE NOT ALLOWED to open app directly from the home screen.
- You MUST activate the input box first by clicking it, and clear any default text if necessary before using 'type' action. You must seperate these actions into different steps.
- For table-like screen, you MUST describe the coordinate of the element.
- For "OK" button, you MUST simply describe it as "the center of text 'ok'" without changing the capitalization.

**When to take note:**
- When you find or calculate any USEFUL information that you can't get from the user's request directly, you MUST take note of it.
- Do not describe the element on the screen, only take note of important information that may help complete the task.
- If there is no important details to note, just leave the <note></note> empty.
- Do not repeat the notes you have already taken in previous steps.
- Do not take note of your action history or explanation of your next action.
- You MUST take notes when you find any unnecessary or repetitive actions in history in order to remind yourself to avoid making the same mistakes.
- If find some useful text, you can directly take note of it instead of copy it, and then type it to where you want to use it.

**Specially, once you have completed the task, you MUST terminate the task immediately, Do not keep swiping to find any new task or do any checks after you finish the task.**
```

Figure 3: Illustrative bad case when using Gemini-2.5-pro as the planner and LAMO-3B as the policy executor in AndroidWorld. The goal of the task is *"Create a playlist titled 'Ultimate Fails Series' with the following files in VLC (located in Internal Memory/VLCVideos), in order: highlight_41_4K_2023_03_30.mp4, scene_68_4K_copy.mp4"*. **The results indicate that LAMO-3B precisely aligns and executes the planner's instruction. At this stage, the overall performance of the framework is primarily constrained by the planner's decision-making quality: an erroneous plan can directly precipitate task failure.** Specifically, at Step 10, the planner attempted to enter the playlist name without first focusing the text input field; it then failed to recognize the missing/incorrect entry and proceeded to trigger the save action, ultimately causing task failure.

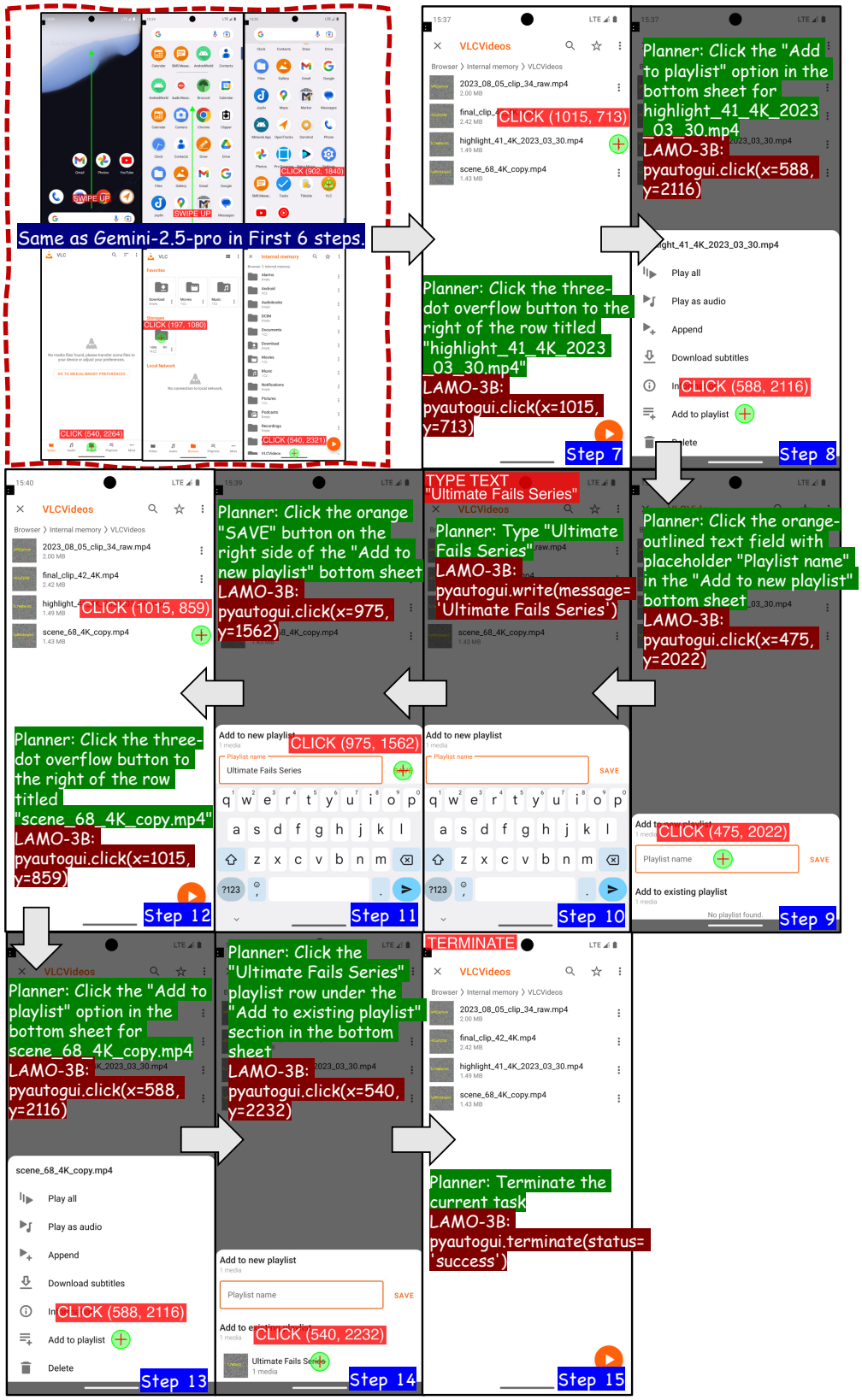

Figure 4: A case in AndroidWorld corresponding to the same task as in Figure 3, but using GPT-5 as the planner. **Compared with the previously shown episode, GPT-5 exhibits markedly enhanced planning capabilities, benefiting from richer world knowledge that enables more accurate interpretation of end-user devices usage and provides more reasonable high-level planning, while our LAMO-3B act as a plug-and-play policy executor for low-level GUI execution.** In particular, at Step 9 of this episode, GPT-5 encountered the same scenario as Gemini-2.5-pro but correctly inferred that the input field on the screen had not been activated and therefore first clicked on the input box before entering the text. **As the planner improves, the overall performance of the framework correspondingly advances.**

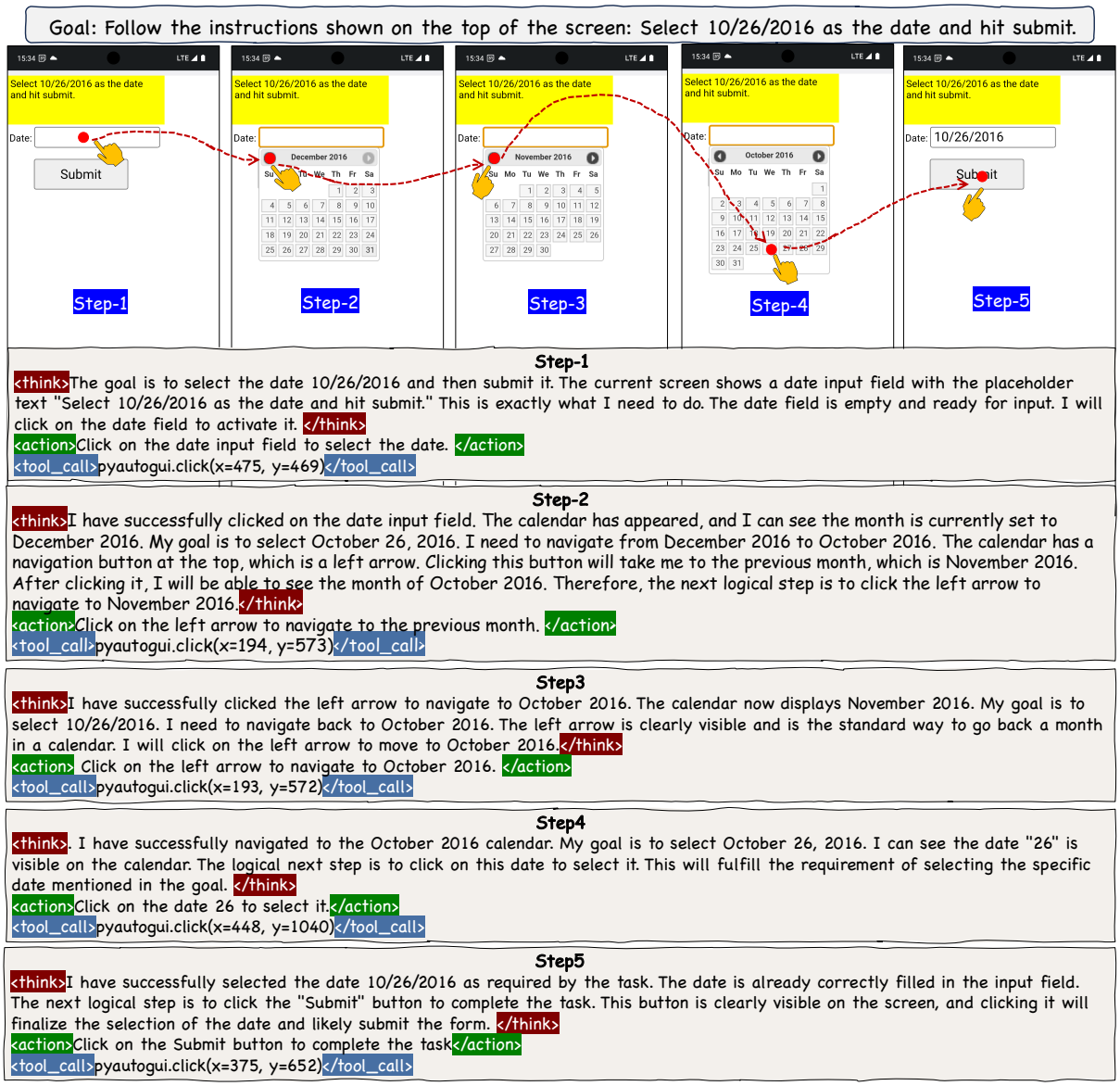

Figure 5: Case Study: LAMO-3B achieves GUI automation on the MiniWob++ benchmark through end-to-end reasoning. The results demonstrate LAMO-3B's capabilities in screen perception, long-horizon reasoning, and history-awareness, as well as precise pixel-level GUI interaction.

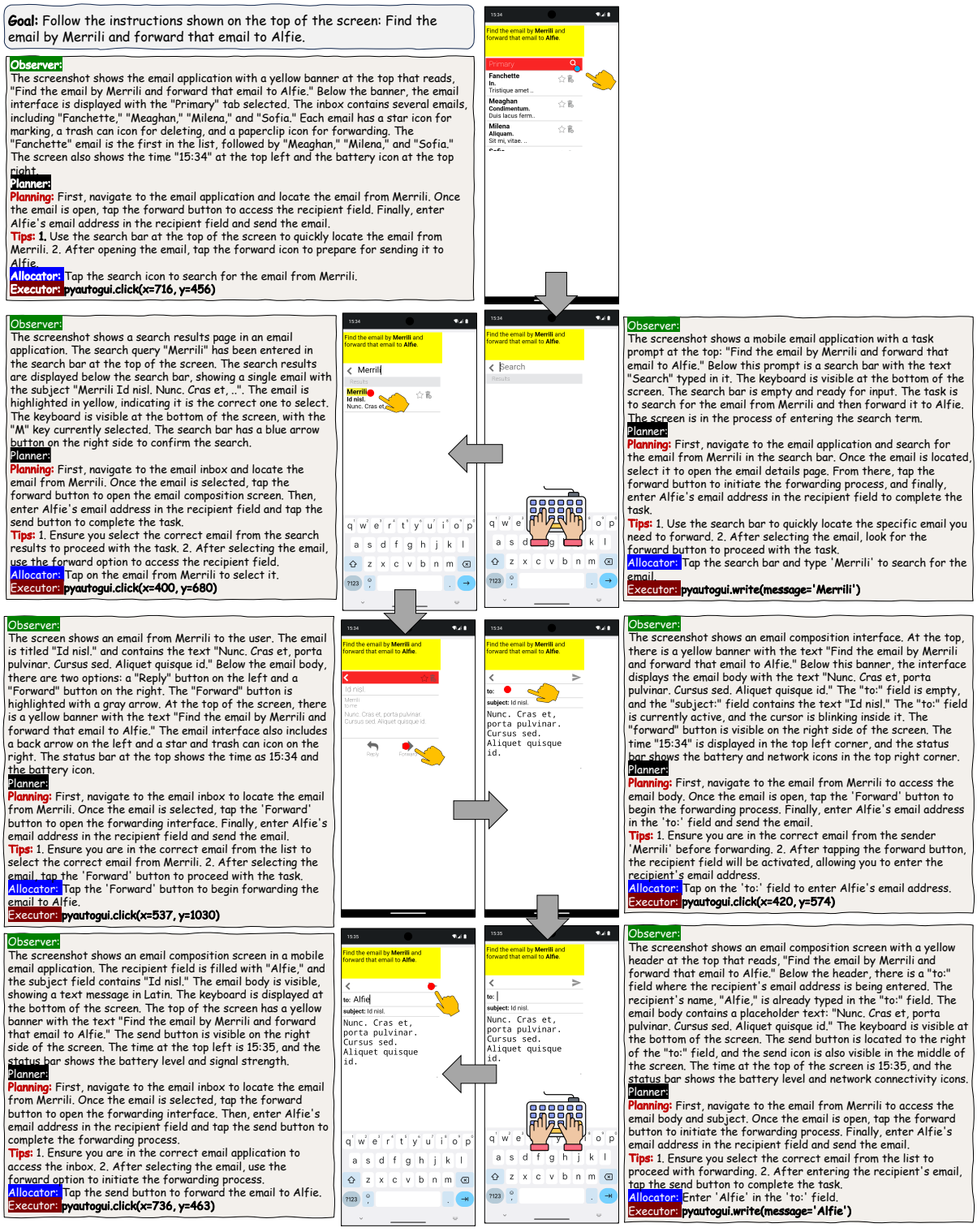

Figure 6: Case Study: LAMO-3B orchestrates a multi-agent system (MAS) to produce GUI automation on the MiniWob++ benchmark. These results demonstrate the multi-role orchestration capabilities of LAMO-3B: the **Observer** facilitates fine-grained perception of screen details, while the **Planner** provides dynamic task planning alongside actionable tips to mitigate execution errors. Furthermore, the **Allocator** analyzes contextual information to predict the optimal actions for the current screen state, and the **Executor** precisely aligns with the Allocator's instructions to perform low-level GUI interactions.

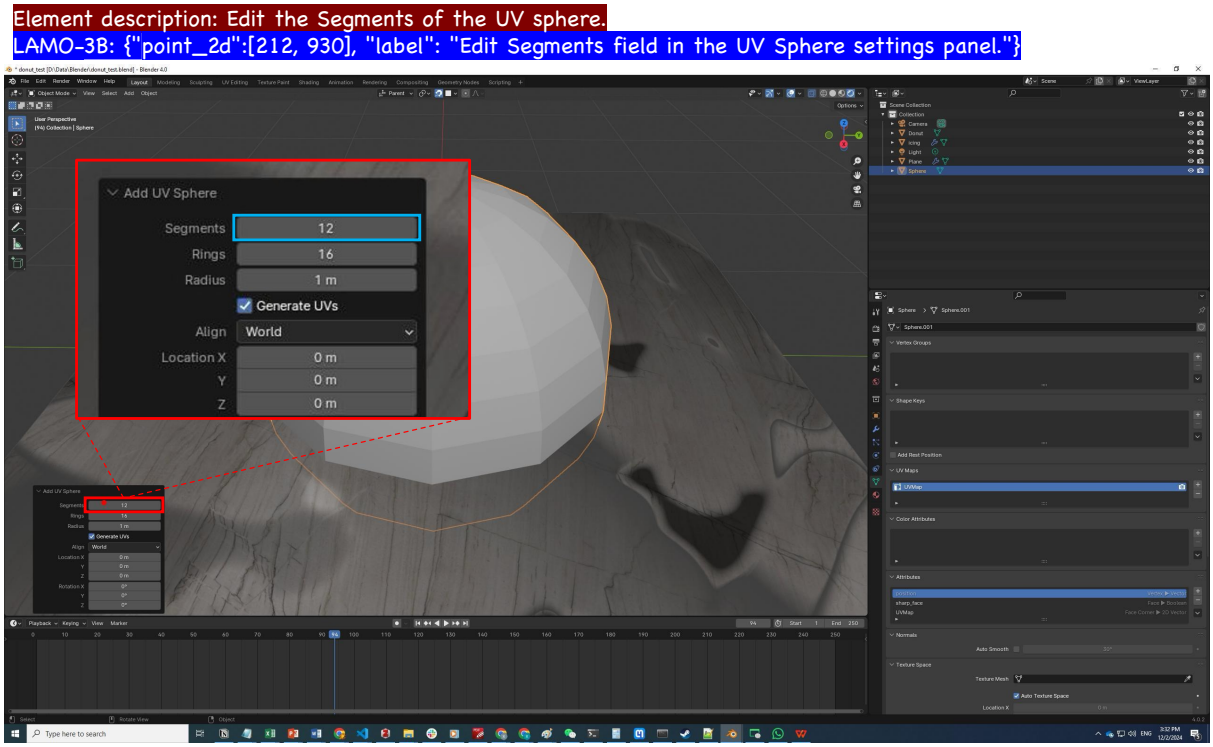

Figure 7: Example visualizations from ScreenSpot-pro, illustrating LAMO-3B's ability to perceive on-screen visual clues and spatial context. Moreover, our semantic enhancement strategy for screen grounding (Sec. 4.1), addressing the first challenge in SG via data distillation, fosters a profound multimodal understanding of UI environments and element descriptions. This empowers LAMO-3B to precisely ground the target element while simultaneously capturing spatial context and intricate layout details.

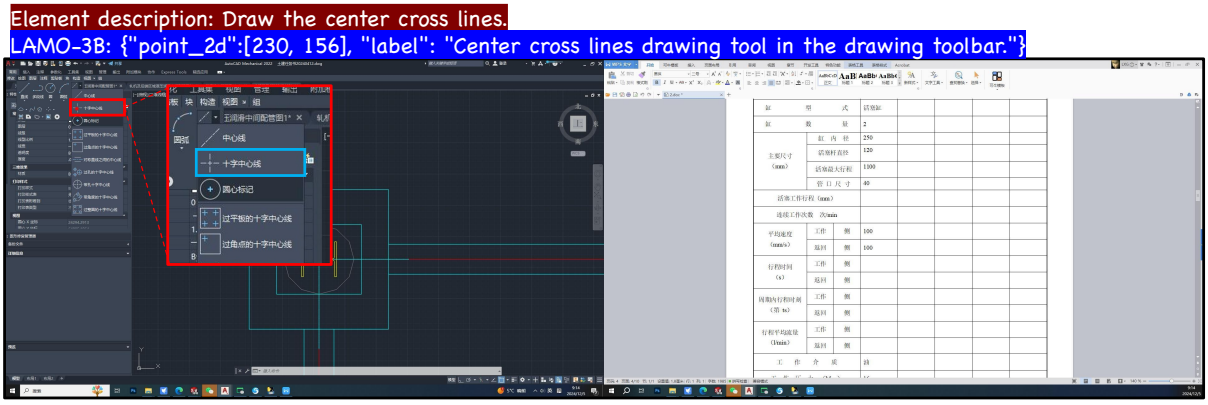

Figure 8: Example visualizations from ScreenSpot-pro, demonstrating LAMO-3B's ability to comprehend multilingual on-screen information.

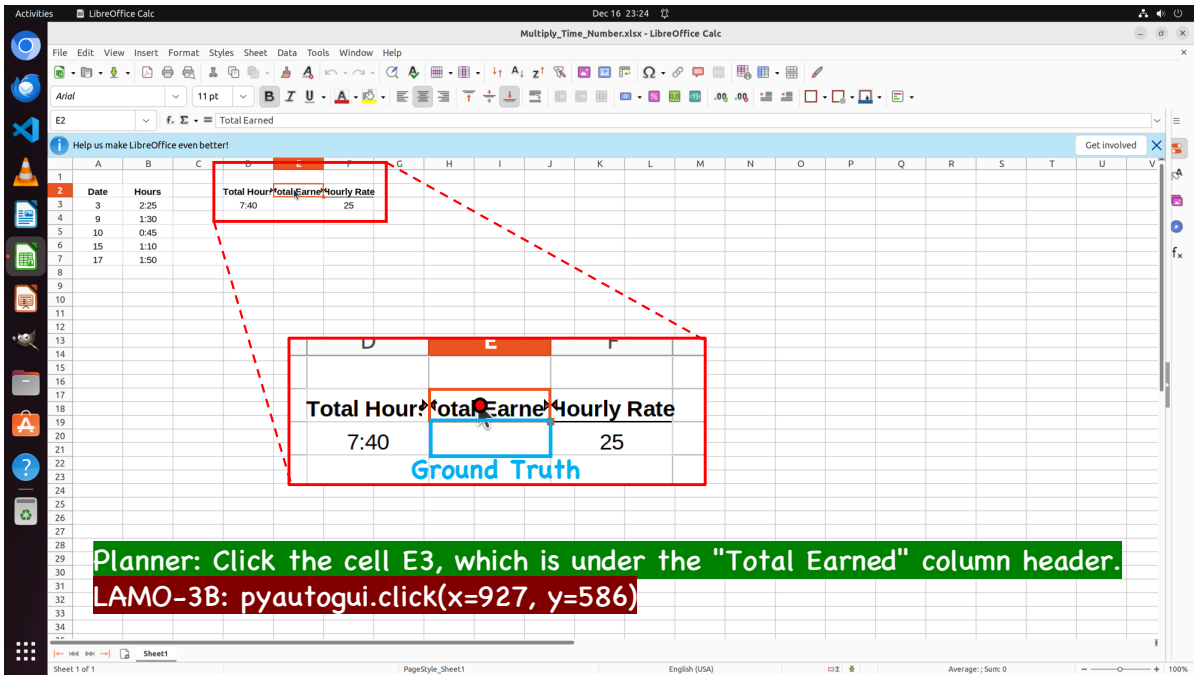

Figure 9: A bad case on OSWorld. In this instance, LAMO-3B suffered from an over-reliance on textual semantic matching—mistakenly attending to the *"Total Earned"* header while overlooking the spatial constraints for cell E3. Tabular interfaces, which often feature extensive sparse regions, amplify this susceptibility, leading to grounding misalignment in long-horizon spreadsheet manipulation.

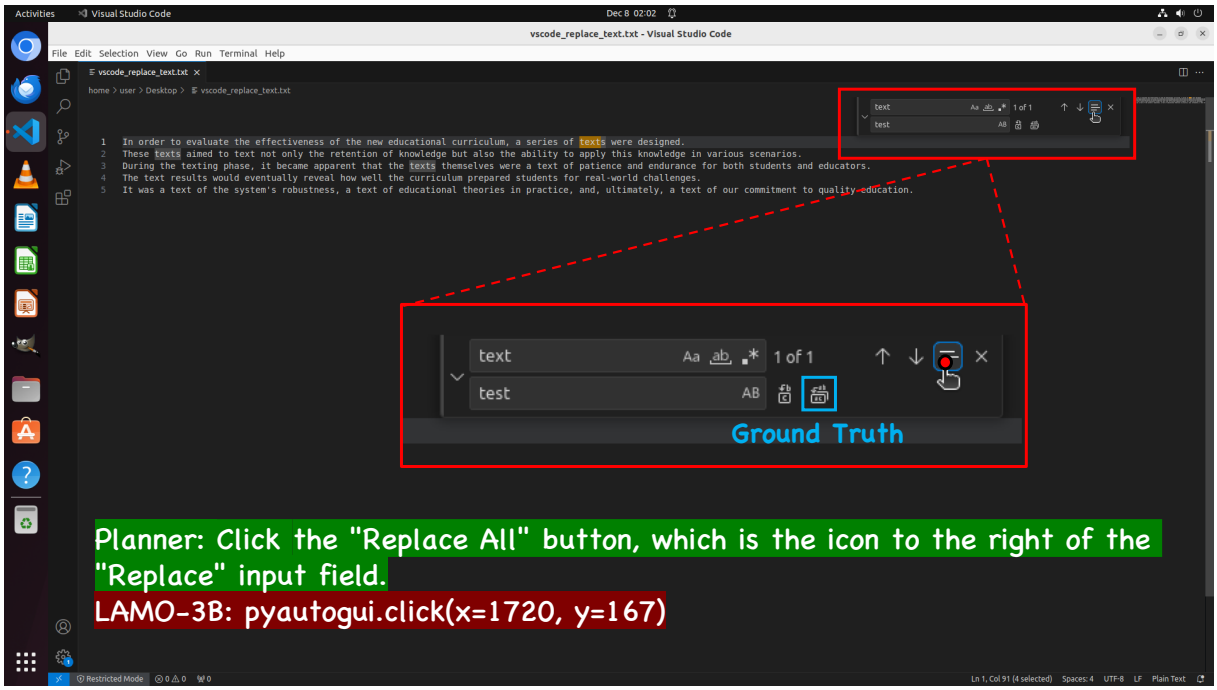

Figure 10: A bad case on OSWorld. In this instance, the Planner's instruction is to *'Click the "Replace All" button, which is the icon to the right of the "Replace" input field.'* However, the ground truth target is a purely graphical element devoid of local textual clues, necessitating that the policy executor possesses prior functional knowledge of the software's interface. Although LAMO-3B successfully identifies the target's Region-of-Interest, its lack of such software-specific priors ultimately leads to a grounding failure.

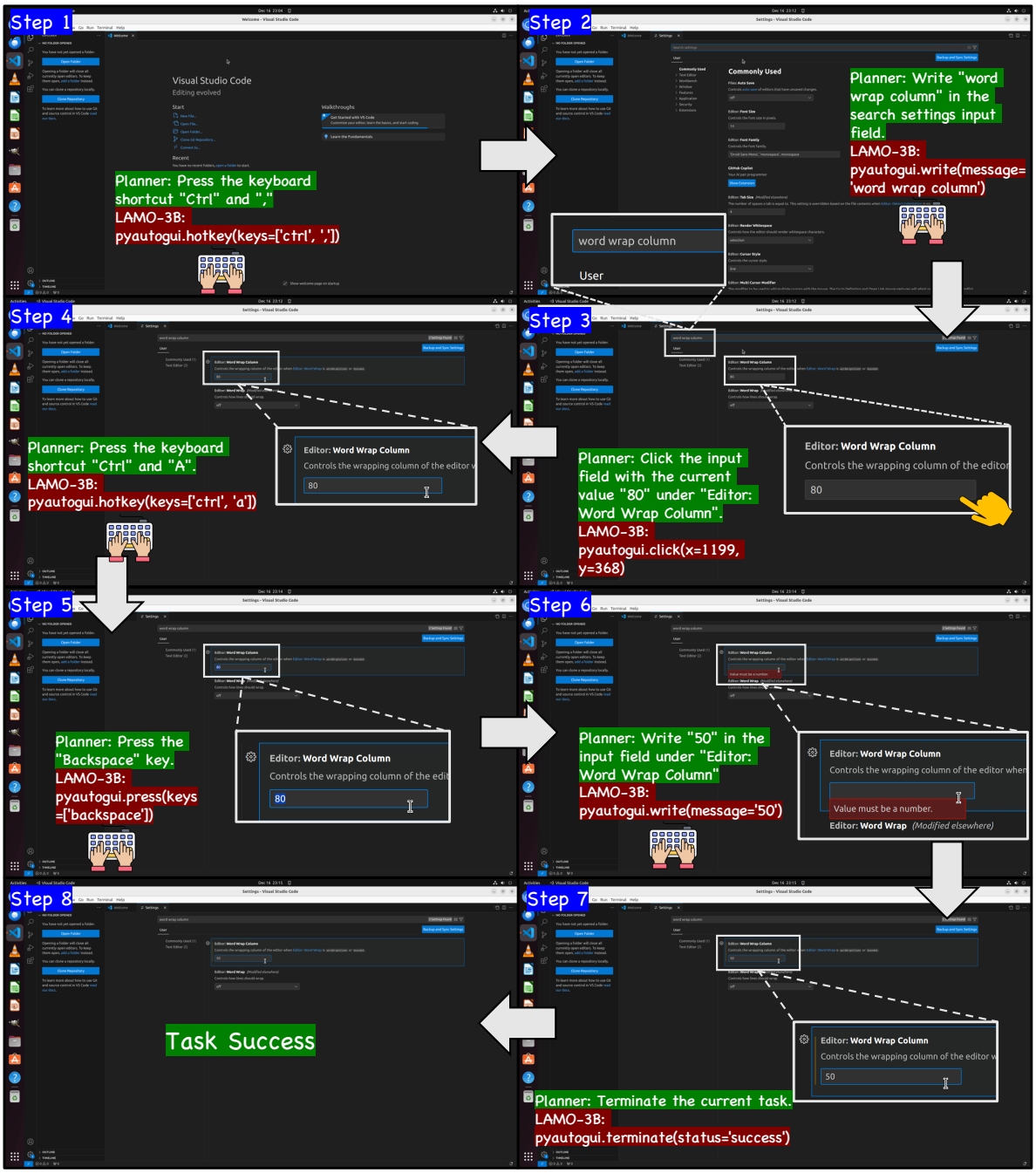

Figure 11: A complete episode on OSWorld. The goal of the task: *"Please help me set the current user's line length for code wrapping to 50 characters in VS Code."* In this episode, LAMO-3B precisely aligns the planner's instructions and accurately executes a diverse set of atomic actions in pixel-level, demonstrating both the robustness of LAMO-3B as a policy executor and the richness of its action space (Tab. 7), which together enable scalable execution of high-level plans produced by the planner across a broad spectrum of long-horizon GUI tasks.