What's Missing in Screen-to-Action? Towards a UI-in-the-Loop Paradigm for Multimodal GUI Reasoning

Anonymous ACL submission

Abstract

Existing Graphical User Interface (GUI) reasoning tasks remain challenging, particularly in UI understanding. Current methods typically rely on direct screen-based decision-making, which lacks interpretability and overlooks a comprehensive understanding of UI elements, ultimately leading to task failure. To enhance the understanding and interaction with UIs, we propose an innovative GUI reasoning paradigm called ***UI-in-the-Loop*** (UILoop). Our approach treats the GUI reasoning task as a cyclic ***Screen-UI elements-Action*** process. By enabling Multimodal Large Language Models (MLLMs) to explicitly learn the localization, semantic functions, and practical usage of key UI elements, UILoop achieves precise element discovery and performs interpretable reasoning. Furthermore, we introduce a more challenging ***UI Comprehension*** task centered on UI elements with three evaluation metrics. Correspondingly, we contribute a benchmark of 26K samples (UI Comprehension-Bench) to comprehensively evaluate existing methods' mastery of UI elements. Extensive experiments demonstrate that UILoop achieves state-of-the-art UI understanding performance while yielding superior results in GUI reasoning tasks.

1 Introduction

GUI automation leverages Artificial Intelligence to simulate user interactions with device screens, reducing human workload (Nguyen et al., 2025). Recent advances in MLLMs have significantly enhanced GUI agents (Wang et al., 2023), demonstrating substantial potential in web browsing, mobile apps, and office automation (Qin et al., 2025), while advancing Artificial General Intelligence development (Hu et al., 2025).

Existing GUI agents leverage advanced MLLMs (e.g., GPT-4o (Hurst et al., 2024) and the Qwen-VL series (Bai et al., 2025)) to interpret user instructions and perform reasoning. However, these methods struggle with the complex layouts and diverse UI elements prevalent in real-world screens (Zhang et al., 2024a). They typically follow a "***Screen-Action***" paradigm, where decisions and actions (e.g., click (123, 204), type "text", scroll down) are generated directly from screen inputs (Wang et al., 2025b; Sun et al., 2025; Pahuja et al., 2025; Qi et al., 2024). This black-box decision-making process lacks interpretability and fails to foster a comprehensive understanding of UI elements (Wang et al., 2024). Consequently, models often fail to accurately locate key elements and grasp their semantics and functions. Ultimately, this inability to effectively utilize these elements leads to task failure.

Evaluation of current GUI agents reveals significant deficiencies in UI element comprehension. As depicted in Fig. 1 Left, advanced models exhibit poor performance (average score below 0.1) across three critical dimensions: UI element localization, semantic function description, and practical-usage. Based on this, we provide these models with both beneficial and misleading UI element descriptions during user instruction execution. Fig. 1 Middle demonstrates that correct UI understanding substantially enhances reasoning across all scenarios—including zero-shot MLLMs, GUI expert, and models of varying scales. Conversely, incorrect descriptions significantly increase task failure rates. These findings underscore the critical role of UI element comprehension in GUI reasoning.

To address the "Missing in the Screen-to-Action" limitation inherent in current GUI models, we propose ***UI-in-the-Loop*** (UILoop)—an innovative paradigm that reframes GUI reasoning around the mastery of UI elements. As illustrated in Fig. 2, UILoop conceptualizes this process as a cyclic "***Screen–UI Elements–Action***" process, where UI elements serve as the critical bridge from screen to action, enabling more accurate reasoning based on correct UI elements. Leveraging reinforce-

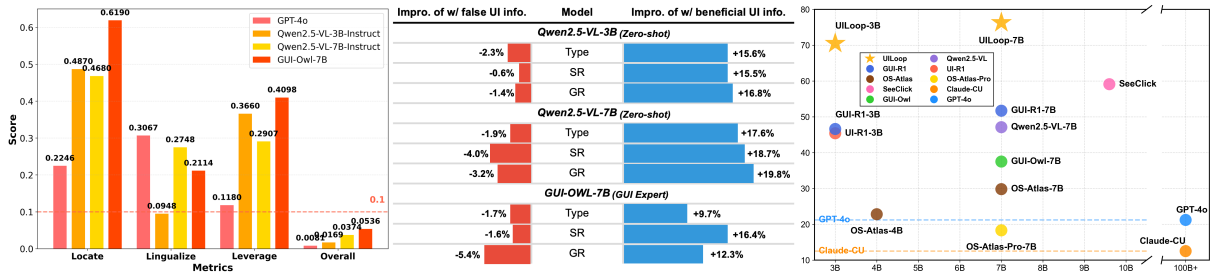

Figure 1: **Left**: Evaluation of existing methods on UI element localization, semantic function description, and practical usage. **Middle**: Performance gains with correct vs. misleading UI info compared to without UI info. **Right**: Comparison of UILoop against existing ***"Screen-to-Action"*** methods on SR metric for Android Control-High.

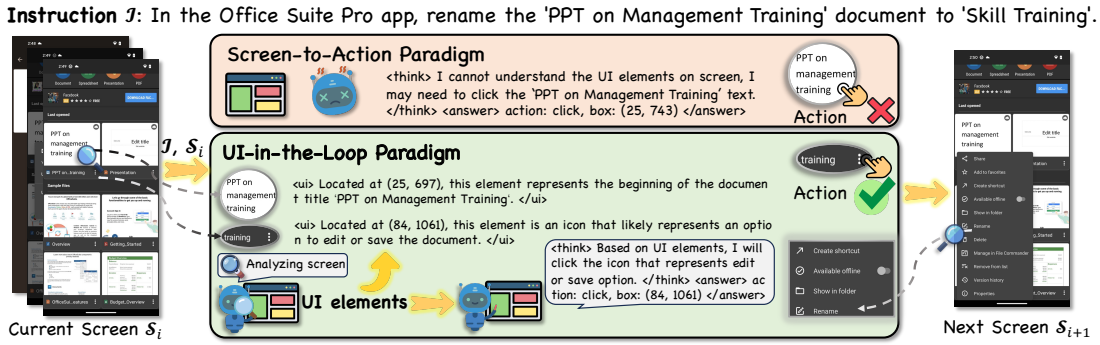

Figure 2: Compared to the existing "***Screen-to-Action***" paradigm, our UI-in-the-Loop reframes GUI reasoning as "***Screen-UI Elements-Action***".

ment learning's strength in handling complex sequential decisions (Shao et al., 2024), we design UI-Element-Driven Reinforcement Fine-Tuning, which teaches UILoop to locate key UI elements, infer their semantic functions, and master their practical usage, thereby achieving precise UI parsing and interpretable reasoning. Furthermore, recognizing the difficulty of understanding and applying UI elements, we introduce the more challenging ***UI Comprehension*** task along with three evaluation metrics, and contribute a 26K benchmark (UI Comprehension-Bench) to comprehensively evaluate the UI localization, semantic understanding, and practical-usage capabilities of existing models. Our major contributions are as follows:

- We demonstrate that comprehensive UI understanding significantly enhances reasoning in existing GUI agents. Building on this insight, we propose the innovative UILoop paradigm, which moves beyond conventional "***Screen-to-Action***" approaches by reframing GUI reasoning as cyclic "***Screen–UI Elements–Action***" loop. Through UI Element–Driven Reinforcement Fine-Tuning, UILoop improves model comprehension of interface elements, thereby advancing mutimodal GUI reasoning and interpretability.
- We introduce the more challenging ***UI Comprehension*** task with three dedicated evaluation metrics (UI Locate, Lingualize, and Leverage) to assess how existing methods master UI elements. To support this, we advance community research by contributing UI Comprehension-Bench, a 26K benchmark for comprehensive UI capability assessment.
- Extensive experiments demonstrate that UILoop achieves state-of-the-art (SOTA) performance in UI comprehension, while delivering superior results in GUI reasoning tasks.

2 Related Work

Screen-to-Action GUI Agent. Current approaches enhance GUI reasoning through large-scale pretraining (GUI-OWL (Ye et al., 2025)) and supervised fine-tuning (Aguvis (Xu et al., 2024), CoCo-Agent (Ma et al., 2024), Show-UI (Lin et al., 2025), Aria-UI (Yang et al., 2025)). Moreover, recent work (UI-R1 (Lu et al., 2025), GUI-R1 (Luo et al., 2025), InfiGUI-R1 (Liu et al., 2025b), InfiGUI-G1 (Liu et al., 2025c)) designs reinforcement learning for robust sequential decision-making. Several datasets such as Meta-GUI (Sun et al., 2022), AITW (Rawles et al., 2023), GUIAct

(Chen et al., 2025), OmniACT (Kapoor et al., 2024), Android Control (Li et al., 2024), AITZ (Zhang et al., 2024b) have been proposed to enhance SFT or RL training for the "Screen-to-Action" paradigm. However, this paradigm implicitly embeds UI comprehension within action prediction, lacking explicit UI element focus and limiting interpretability.

UI Elements-Enhanced GUI Agent. Existing methods focus on UI element localization but ignore semantic functions and practical usage. SeeClick (Cheng et al., 2024) improves localization via ScreenSpot dataset. GUI-explorer (Xie et al., 2025) retrieves UI information externally but doesn't enhance intrinsic understanding. ScreenSpot-Pro (Li et al., 2025), MMBench-GUI (Wang et al., 2025a), UI-E2I-Bench (Liu et al., 2025a), UI-Vision (Nayak et al., 2025), OS-Atlas (Wu et al., 2024), and UGround (Gou et al., 2025) improve localization but neglect their semantic and functional understanding, resulting in incorrect interactions such as clicking a scrollbar instead of dragging it. To address this, we propose UILoop, a "Screen-UI Element-Action" paradigm that explicitly teaches models to master UI elements, achieving superior GUI reasoning performance.

3 Preliminary

GUI Reasoning. Given a user instruction $\mathcal{I}$, we formulate the GUI reasoning task as a multi-turn iterative decision-making process. At each step, the GUI Agent needs to interact with the current screen $\mathcal{S}_i$ and output an action. Therefore, our objective is to train the policy model π_θ to output the correct action a_i to complete the user instruction:

$$\theta^* = \underset{\theta}{argmax} \prod_i P_{\pi_\theta}\left(a_i|\mathcal{I}, \mathcal{S}_i\right),$$

where i is the i-th iteration cycle. Meanwhile, π_θ needs to analyze the UI elements in $\mathcal{S}_i$ that are beneficial for task completion: $\mathcal{U} = \left\{u_i = \left[u_i^{loc} \in \mathcal{U}^{loc}, u_i^{lin} \in \mathcal{U}^{lin}, u_i^{lev} \in \mathcal{U}^{lev}\right]\right\}$, where $\mathcal{U}^{loc}$,$\mathcal{U}^{lin}$,$\mathcal{U}^{lev}$ represent location (e.g., (84, 1061)), semantic and functional description (e.g., "this element is an icon that likely represents an option to edit or save the document"), and usage (e.g., action: click, box: (84, 1061)), respectively. By using $\mathcal{U}$ to obtain a_i, we can therefore model the objective as a "***Screen–UI Elements–Action***" iteration loop as follows:

$$\theta^* = \underset{\theta}{argmax} \prod_i P_{\pi_\theta}\left(a_i|\mathcal{I}, u_j\right) \prod_j P_{\pi_\theta}\left(u_j|\mathcal{I}, \mathcal{S}_i\right)$$

Group Relative Policy Optimization (GRPO) (Guo et al., 2025) is a reinforcement learning algorithm for training models to improve performance on complex sequential decision-making (e.g., GUI reasoning). We employ GRPO to optimize our model. GRPO estimates the relative advantage of each response within a group of responses to the same prompt, eliminating the need for a value function. The optimization objective is:

$$\begin{aligned}\mathcal{L}(\theta) &= \mathbb{E}_{\{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(O|\mathcal{I},\mathcal{S})} \\ &= \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\Bigg\{\min\Bigg[\frac{\pi_\theta(o_{i,t}|\mathcal{I}, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|\mathcal{I}, o_{i,<t})} \\ &\quad A_{i,t}^{\mathcal{U}}, \text{clip}\left(\frac{\pi_\theta(o_{i,t}|\mathcal{I}, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|\mathcal{I}, o_{i,<t})}, 1-\epsilon, 1+\epsilon\right)A_{i,t}^{\mathcal{U}}\Bigg] \\ &\quad - \beta\,\mathbb{D}_{\text{KL}}(\pi_\theta\|\pi_{\text{ref}})\Bigg\},\end{aligned}$$

where G is the number of responses per $\mathcal{I}$, o_i is the i-th response, $\pi_{\theta_{old}}$ is the old policy, π_θ is the current policy, $A_{i,t}^{\mathcal{U}}$ is the UI advantage of the i-th response at position t, ϵ is the clipping range, and $\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right)$ denotes the KL divergence penalty.

4 UI-in-the-Loop Framework

As shown in Fig. 3, our GUI reasoning paradigm, ***UI-in-the-Loop*** (UILoop), consists of two main stages. In the first stage, we design a Scaling Data for UI Comprehension synthesis pipeline to construct the UI Comprehension-Bench, serving to enhance the model's ability to understand and utilize UI elements. In the second stage, with this benchmark, we propose UI Element-Driven Reinforcement Fine-Tuning to address the "Missing in the Screen-to-Action" limitation of existing models and strengthen the model's UI comprehension capabilities.

4.1 Scaling Data for UI Comprehension

Data Collection. Existing GUI Reasoning datasets serve the "Screen-to-Action" paradigm. Therefore, they lack fine-grained information regarding the location, semantic functionality, and practical usage of key UI elements on the screen. Consequently, we conduct a comprehensive augmentation of UI element information for existing GUI reasoning datasets.

Specifically, we collect training and testing data from Android Control (Li et al., 2024), OmniAct (Kapoor et al., 2024), GUI-Act (Chen et al., 2025), ScreenSpot (Cheng et al., 2024), ScreenSpot-Pro

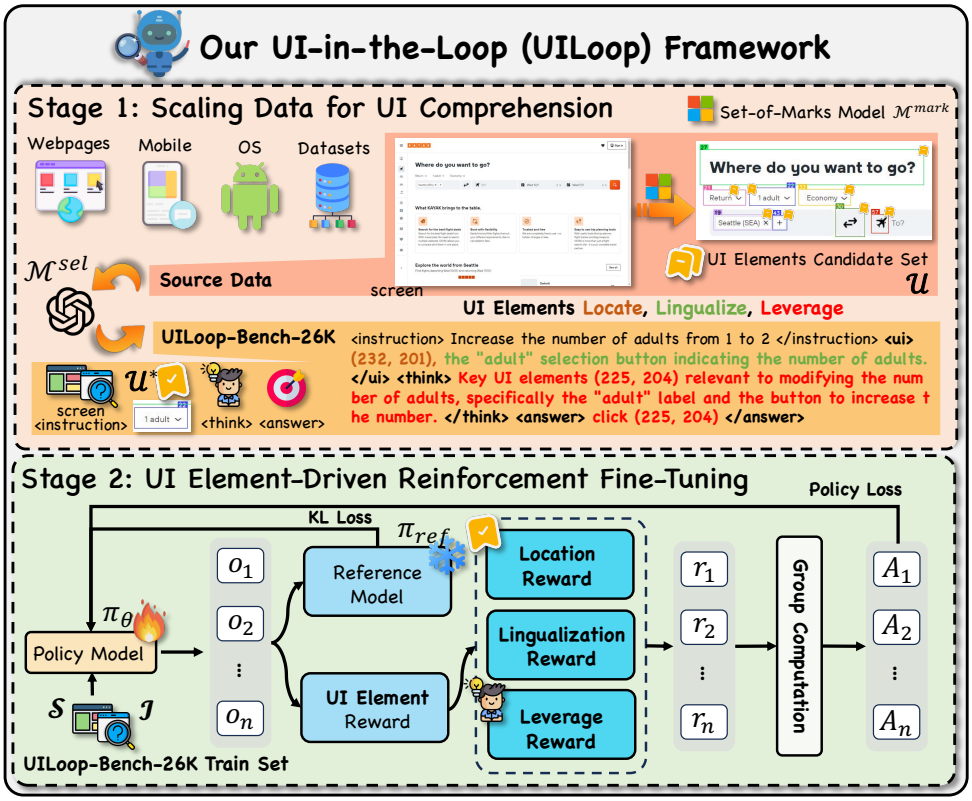

Figure 3: Overview of our ***UI-in-the-Loop*** (UILoop) framework.

(Li et al., 2025), and OS-Atlas (Wu et al., 2024) as source data, whose original data format is presented as $(\mathcal{I}, \mathcal{S}, a)$. Based on this, we apply the set-of-marks model $\mathcal{M}^{mark}$ to $\mathcal{S}$ (e.g., OmniParser V2 (Yu et al., 2025)) to mark the locations of all identifiable UI elements as follows:

$$\mathcal{M}^{mark}(\mathcal{S}) \rightarrow \mathcal{U}^{loc}$$

We employ GPT-4o as the selection model $\mathcal{M}^{sel}$ to filter out key UI elements that are beneficial for completing user instruction $\mathcal{I}$, and supplement the semantic functionality of these UI elements (as shown in Fig. 2, included in <ui> along with location information) and practical usage (in the <think> and <answer> parts) described as follows:

$$\mathcal{M}^{sel}\left(\mathcal{I}, \mathcal{S}, \mathcal{U}^{loc}, a\right) \rightarrow \mathcal{U}^{*},$$

where $\mathcal{U}^*$ represents the key UI elements. In addition, we perform fine-grained augmentation of UI element information for the dataset based on three different sources: Webpages, Mobile, and Operating System, following the same procedure as described above. Construction details are provided in the Appendix A. Finally, we augment the fine-grained UI information and construct UI Comprehension-Bench, with data format: $(\mathcal{I}, \mathcal{S}, \mathcal{U}^*, a)$. Details are in Appendix B.

More than Action Prediction: UI Comprehension. Existing GUI reasoning methods focus solely on "Screen-to-Action" prediction, leaving the reasoning process a black box. Even when models output reasoning traces, they lack explicit modeling and evaluation of intermediate steps. Current evaluations measure only final action accuracy, neglecting UI element understanding and utilization, thus lacking interpretability. To address this, we propose a novel task: ***UI Comprehension***, which provides interpretable intermediate representations based on UI elements, establishing a transparent "Screen-UI Element-Action" reasoning paradigm.

We design three evaluation metrics: Locate, Lingualize, and Leverage, assessing UI element localization, semantic function understanding, and utilization accuracy, respectively. The calculation of metrics is detailed in Sec. 4.2. We define the final score as: Overall = Locate $*$ Lingualize $*$ Leverage. Furthermore, we contribute UI Comprehension-Bench 26K for this task.

Statistics of UI Comprehension-Bench. Tab. 1 compares our large-scale 26K UI Comprehension-Bench with existing GUI reasoning datasets. We are the first to provide Ground Truth (GT) UI elements (i.e., key UI elements) for screens and offer

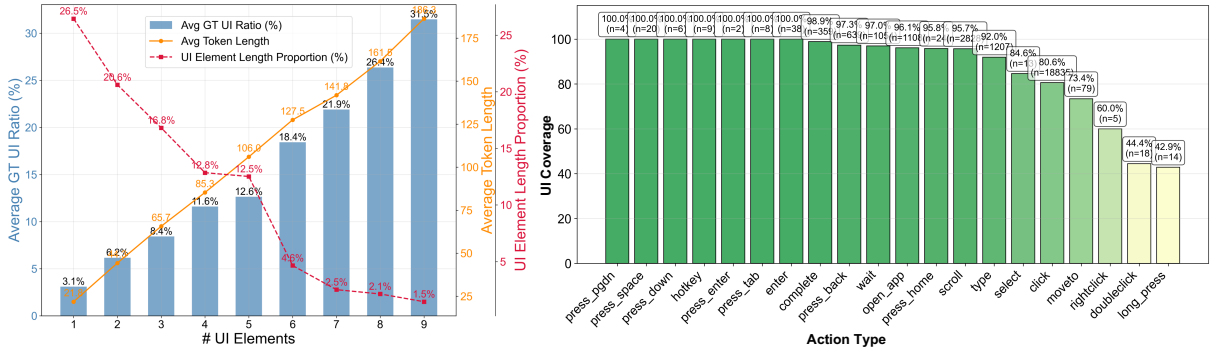

Figure 4: Statistics of Our UI Comprehension-Bench. **Left:** Proportion and distribution of GT UI elements; token length of their semantic descriptions. **Right:** Proportion of GT UI elements effectively used in action inference.

a fully interpretable "Screen-UI Elements-Action" reasoning chain: locating GT UI elements, describing their semantic functions and practical usage, and finally deriving the action.

Fig. 4 presents detailed statistics. The benchmark contains 1,576,068 UI elements, with only 57,332 GT UI elements (<4%), demonstrating identification difficulty. Fig. 4 Left visualizes the distribution of GT UI element proportions. When only 1 GT UI element exists, it comprises merely 3.1% of total elements, requiring models to identify it among numerous irrelevant layouts. Such samples constitute 26.5% of UI Comprehension-Bench, highlighting the difficulty. To verify our UI element effectiveness, we visualize text coverage rates of GT UI elements during reasoning, grouped by action type. Fig. 4 Right show coverage rates exceeding 90% for most action types, with only minimal actions below 80% (e.g., long_press with 14 samples). This demonstrates that UI Comprehension-Bench provides high-quality UI elements with logical coherence and interpretability.

4.2 UI Element-Driven Reinforcement Fine-Tuning

To address the "Missing in the Screen-to-Action" limitation, we leverage reinforcement learning's strength in handling complex sequential decisions and propose UI Element-Driven Reinforcement Fine-Tuning to enhance the model's UI Comprehension capability. Specifically, we design Location, Lingualization, and Leverage Rewards to respectively strengthen the model's ability to locate UI elements, understand their semantic functions, and utilize them effectively. Firstly, we employ Format Reward to encourage the model to output in the expected format.

Format Reward. We require the model to output in the following format.

Format

<ui> Located at [x, y], describe the UI element's semantics and function. </ui> <think> ... </think> <answer> [{'action': , 'point': , 'input_text': }] </answer>

If the output matches the expected format, the format reward is 1; otherwise, it is 0.

Location Reward. We use the Euclidean distance between the predicted UI element coordinates and the ground truth UI element coordinates as the location reward, defined as follows:

$$r^{loc} = \frac{1}{|\mathcal{U}^*|}\sum_{i=1}^{|\mathcal{U}^*|} 1_D(u_j^{pred}) * [1- \frac{\sqrt{(u_i^{loc*}[x]-u_j^{loc^{pred}}[x])^2+(u_i^{loc*}[y]-u_j^{loc^{pred}}[y])^2}}{\sqrt{w^2+h^2}}],$$

where w and h denote the width and height of the screen, respectively, and $1_D(\cdot)$ is an indicator function that equals 1 when u^{pred} is the nearest predicted UI element to u^*, and 0 otherwise.

Lingualization Reward. We calculate the semantic similarity between the text descriptions of the predicted UI elements and the ground truth UI elements as follows:

$$r^{lin} = \frac{1}{|\mathcal{U}^*|}\sum_{i=1}^{|\mathcal{U}^*|} 1_D\left(u_j^{pred}\right) * sim\left(u_i^{lin^*}, u_j^{lin^{pred}}\right)$$

Leverage Reward. We adopt different calculation methods for action types in UI element utilization as follows. When the action type is 'click':

$$r^{lev} = 1_A\left(u_j^{lev^{pred}}\right)\left(u_j^{lev^{pred}}[point] == u^{lev^*}[point]\right)$$

When the action type is one of 'scroll', 'type', 'open_app', or 'select':

$$r^{lev} = 1_A\left(u_j^{lev^{pred}}\right)\left(u_j^{lev^{pred}}[text] == u^{lev^*}[text]\right)$$

Datasets	# Episodes	# Unique Instructions	Annotation						
			Screen Desc.	Key UI Element Loc.	Key UI Element Lin.	Key UI Element Lev.	Action Coord	Action Desc.	Action Think
AITW	715142	30378	✗	✗	✗	✗	✓	✗	✗
Android Control	15283	15283	✓	✗	✗	✗	✓	✓	✗
MMBench-GUI	8123	8123	✗	✓	✗	✗	✓	✓	✓
ScreenSpot-Pro	1581	1581	✗	✓	✗	✗	✗	✗	✗
UI-E2I-Bench	1477	1477	✗	✓	✗	✗	✗	✗	✗
UI-Vision	8227	∼450	✓	✓	✗	✗	✓	✓	✗
Ours	26207	15735	✔	✔	✔	✔	✔	✔	✔

Table 1: Comparison of our UI Comprehension-Bench with existing GUI reasoning benchmarks.

For other actions, $r^{lev} = 1_A\left(u_j^{lev^{pred}}\right)$. Here, $1_A(\cdot)$ is an indicator function that equals 1 when the action type of $u_j^{lev^{pred}}$ matches that of $u_j^{lev^*}$, and 0 otherwise. **We specifically note that the Location, Lingualize, and Leverage evaluation metrics of UI Comprehension-Bench are consistent with the calculation methods of the Location, Lingualization, and Leverage Rewards described above.** We define the overall reward as follows:

$$r = r^{format} + \alpha_1 * r^{loc} * r^{lin} + \alpha_2 * 1_U\left(r^{loc} * r^{lin}\right) * r^{lev}$$

$1_U(\cdot)$ is an indicator function that equals 1 when $r^{loc} * r^{lin} > \eta$, and 0 otherwise. This design ensures that during training, the model **prioritizes locating key UI elements on the screen and understanding their semantic functions, and then learns to utilize these elements for accurate decision-making**.

Finally, we compute the advantage function using the obtained rewards as follows:

$$A_i^{\mathcal{U}} = \frac{r_i - mean\left(\{r_1, r_2, ..., r_G\}\right)}{std\left(\{r_1, r_2, ..., r_G\}\right)}$$

where $mean$ and std denote the mean and standard deviation, respectively.

5 Experiments

5.1 Experiment Setting

Datasets. We evaluate on the test splits of Android Control-High and ScreenSpot-Pro, which assess high-difficulty multi-step GUI reasoning and cross-platform grounding, respectively. For UI Comprehension, we use UI Comprehension-Bench 26K, with statistics reported in Appendix B.

Evaluation Metrics. We use action type accuracyw (Type), point accuracy (Ground Rate, GR), and step success rate (SR). Type measures action accuracy, GR assesses grounding capability, and SR evaluates overall accuracy of actions, coordinates, and text. For ScreenSpot-Pro, we use GR. For UI Comprehension, we use Locate, Lingualization, and Leverage to assess UI element grounding, semantic understanding, and utilization accuracy.

Baselines. We compare: (1) Zero-shot general MLLMs performing GUI reasoning without training; (2) Screen-to-Action models—trained on GUI datasets to directly output actions from screens.

Implementation Details. We use Qwen2.5-VL-3B and 7B as base models, trained on UI Comprehension-Bench's training set (Details in Appendix B). We perform RFT using Verl (Sheng et al., 2024) until reward convergence (3∼6 epochs) with 5 rollouts. Prompts are detailed in the Appendix C. All experiments run on 8 A100 80G GPUs. α_1 and α_2 are set to 4, 5 separately. The UI indicator threshold η is 0.5.

5.2 Main Result

As shown in Tab. 2, zero-shot MLLMs generally underperform training-based MLLMs due to lack of GUI training. Our method surpasses "Screen-to-Action" models on both datasets. On ScreenSpot-Pro, our 3B and 7B models outperform similarly-sized Qwen2.5-VL and GUI-R1 by 13.3%, 2% and 13.3%, 3.2% in overall scores, respectively. On Android Control-High, our 7B model exceeds GUI expert models OS-Atlas-7B, OS-Atlas-Pro-7B, and GUI-OWL-7B by 46.5%, 58.0%, and 38.8% in SR, respectively. These results demonstrate the superiority of the "Screen-UI Element-Action" paradigm.

5.3 Ablation Study

We conducted ablation studies to examine the impact of different UI Rewards on reasoning performance, as shown in Fig. 5. We evaluated: (1) Direct SFT; (2) Direct RFT with Leverage Reward only; (3) Locate + Leverage Rewards; (4) Full UILoop. Results show that Leverage Reward improves all metrics by teaching models to analyze and utilize UI elements. Adding Locate Reward increases GR by 7.9% and 8.6% for 3B and 7B

Methods	*ScreenSpot-Pro*															*AndroidControl-High*		
	Dev		Creative		CAD		Sci.		Office		OS		Overall			Type	SR	GR
	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Avg.			
Zero-Shot Models																		
Claude-CU	22.0	3.9	25.9	3.4	14.5	3.7	33.9	15.8	30.1	16.3	11.0	4.5	23.4	7.1	17.1	63.7	12.5	-
GPT-4o	1.3	0.0	1.0	0.0	2.0	0.0	2.1	0.0	1.1	0.0	0.0	0.0	1.3	0.0	0.8	63.1	21.2	30.9
Qwen2.5-VL-3B	16.2	1.4	23.3	1.4	10.2	4.7	38.2	6.4	24.3	3.8	15.0	1.1	21.2	3.1	12.2	47.8	38.9	46.5
Qwen2.5-VL-7B	33.1	2.1	23.7	3.5	12.2	6.3	36.8	7.3	37.8	7.5	30.8	6.9	29.1	5.6	17.4	68.7	47.1	59.7
Screen-to-Action Training Models																		
SeeClick	0.6	0.0	1.0	0.0	2.5	0.0	3.5	0.0	1.1	0.0	2.8	0.0	1.8	0.0	1.1	82.9	59.1	62.9
GUI-Owl-7B	37.0	5.5	32.8	1.4	23.9	4.7	37.5	10.0	33.9	11.3	18.7	3.4	31.0	5.5	21.3	72.9	37.5	53.7
OS-Atlas-Pro-7B	1.4	0.0	1.1	0.0	2.7	0.0	1.5	0.0	1.8	2.0	0.0	0.0	1.4	0.3	0.9	69.7	18.3	16.8
OS-Atlas-4B	7.1	0.0	3.0	1.4	2.0	0.0	9.0	5.5	5.1	3.8	5.6	0.0	5.0	1.7	3.7	49.0	22.8	49.5
OS-Atlas-7B	33.1	1.4	28.8	2.8	12.2	4.7	37.5	7.3	33.9	5.7	27.1	4.5	28.1	4.0	18.9	57.4	29.8	54.9
Qwen2.5-VL-3B*	20.3	1.8	24.6	2.8	11.2	4.7	39.5	6.4	28.6	5.7	17.8	2.2	23.8	3.9	13.9	52.1	41.2	49.5
Qwen2.5-VL-7B*	31.4	1.8	27.3	3.5	15.7	5.1	40.7	7.9	39.7	8.9	32.4	6.9	31.2	5.7	18.5	69.2	48.1	58.7
ShowUI-2B	16.9	1.4	9.1	0.0	2.5	0.0	13.2	7.3	15.3	7.5	10.3	2.2	10.8	2.6	7.7	-	-	-
Aria-UI	16.2	0.0	23.7	2.1	7.6	1.6	27.1	6.4	20.3	1.9	4.7	0.0	17.1	2.0	11.3	-	10.2	43.2
UI-R1-3B	22.7	4.1	27.3	3.5	11.2	6.3	43.4	11.8	32.2	11.3	13.1	4.5	25.0	6.9	17.8	57.9	45.4	55.7
UGround-7B	26.6	2.1	27.3	2.8	14.2	1.6	31.9	2.7	31.6	11.3	17.8	0.0	25.0	2.8	16.5	-	-	-
GUI-R1-3B	33.8	4.8	40.9	5.6	26.4	7.8	61.8	17.3	53.6	17.0	28.1	5.6	40.7	9.7	25.2	58.0	46.6	56.2
GUI-R1-7B	49.4	4.8	38.9	8.4	23.9	6.3	**55.6**	11.8	**58.7**	**26.4**	**42.1**	**16.9**	44.8	**12.4**	28.6	71.6	51.7	65.6
UILoop Training Models																		
UILoop-3B	46.1	4.8	45.6	7.8	32.5	8.5	48.2	15.0	49.3	10.8	26.4	7.7	41.3	9.1	27.2	85.3	70.5	68.9
UILoop-7B	**52.6**	**9.7**	**47.4**	**9.1**	**38.3**	**12.5**	49.6	**15.2**	51.1	12.7	34.8	8.1	**45.5**	11.2	**31.8**	**88.9**	**76.3**	**81.8**

Table 2: Performance comparison of UILoop with zero-shot and "Screen-to-Action" paradigm models on ScreenSpot-Pro and AndroidControl-High. * denotes SFT models trained on (Luo et al., 2025). Underline and **bold** indicate the best results among 3B and 7B models, respectively.

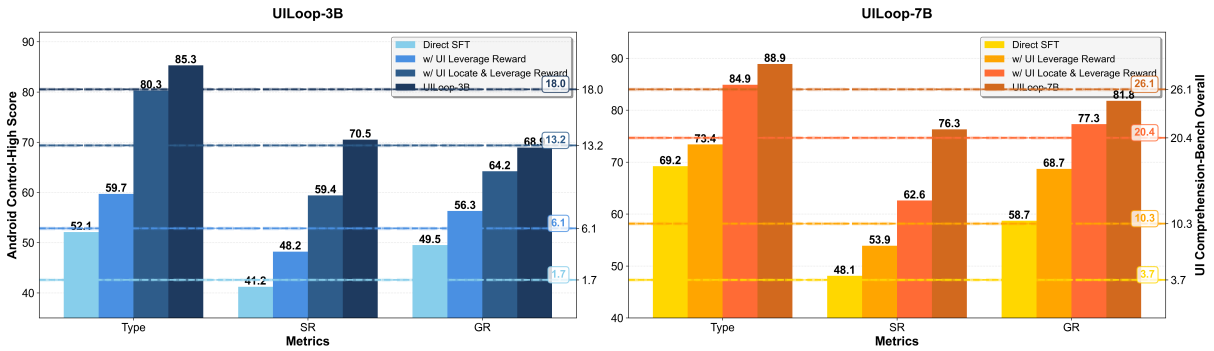

Figure 5: Ablation Study on Android Control-High and UI Comprehension-Bench. We demonstrate the individual contributions of the Locate, Lingualize, Leverage Rewards on reasoning performance and UI comprehension.

models, enhancing key UI element localization and action positioning accuracy. Further adding Lingualize Reward improves SR by 11.1% and 13.7%, strengthening semantic understanding of key UI elements and action text accuracy. These results validate that each reward effectively enhances reasoning by improving UI element mastery.

5.4 Impact of UI Elements

As shown in Tab. 3, we examined three UI intervention approaches: (1) key UI element info., (2) false UI element info., and (3) UILoop Training. Results show false UI info. impairs GUI reasoning, while key UI info. as context significantly improves accuracy, demonstrating that enhancing key UI mastery benefits GUI reasoning. Moreover, UILoop Training surpasses merely providing key UI info., achieving improvements of 31.6% and 22.8% on Qwen2.5-VL-3B and 7B (versus 16.0% and 18.7%), and 17.8% and 29.0% on GUI-Owl-7B and OS-Atlas-Pro-7B (versus 12.8% and 20.7%) for context alone, demonstrating its superiority in enhancing intrinsic UI comprehension and reasoning performance.

5.5 Experiment of UI Comprehension-Bench

We evaluated existing models on our UI Comprehension-Bench, as shown in Tab. 4. Results reveal that current "Screen-to-Action" models perform poorly across Locate, Lingualize, and Leverage tasks, all scoring below 10%. In contrast, UILoop achieves a SOTA score of 26.1 on the 7B

Methods	*Android Control-High*			Impact
	Type	SR	GR	Avg. Ratio
***GPT-4o-mini**(Zero-shot)*				
base	68.1	20.9	6.9	-
w/ UI info.	69.9	51.4	62.9	+29.4
w/ false UI info.	67.2	18.4	5.8	-1.5
***Qwen2.5-VL-3B-Instruct**(Zero-shot)*				
base	58.2	32.7	39.0	-
w/ UI info.	73.8	48.3	55.8	+16.0
w/ false UI info.	55.9	32.1	37.6	-1.4
w/ UILoop	85.3	70.5	68.9	**+31.6**
***Qwen2.5-VL-7B-Instruct**(Zero-shot)*				
base	68.3	53.6	56.7	-
w/ UI info.	86.0	72.3	76.5	+18.7
w/ false UI info.	66.4	49.6	53.5	-3.0
w/ UILoop	88.9	76.3	81.8	**+22.8**
***GUI-Owl-7B**(GUI Expert)*				
base	72.9	37.5	53.7	-
w/ UI info.	82.6	53.8	66.1	+12.8
w/ false UI info.	71.2	35.8	48.4	-2.9
w/ UILoop	84.9	64.7	68.0	**+17.8**
***OS-Atlas-Pro-7B**(GUI Expert)*				
base	69.7	18.3	16.8	-
w/ UI info.	73.3	45.1	48.5	+20.7
w/ false UI info.	54.6	16.7	15.0	-6.2
w/ UILoop	80.3	57.6	53.9	**+29.0**

Table 3: Impact of different UI element intervention methods on GUI reasoning performance.

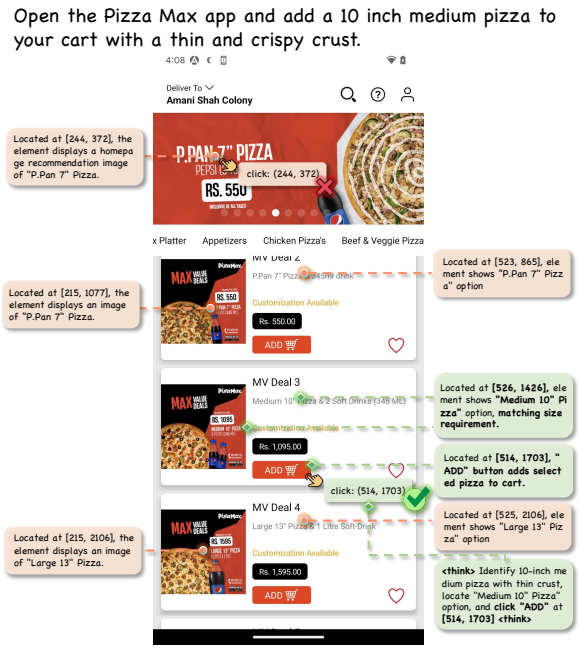

Figure 6: Comparative Case Study between UILoop and "Screen-to-Action".

model, and boosts the overall scores of GUI-Owl-7B and OS-Atlas-Pro-7B by 18.4 and 9.0 (underline parts), demonstrating its superiority in enhancing UI comprehension. Our UI Comprehension-Bench will advance GUI agents from "Screen-to-Action" toward the more superior "Screen-UI Element-Action" paradigm, providing the first robust benchmark for UI comprehension capabilities.

Methods	*UI Comprehension-Bench*			
	Loc.	Lin.	Lev.	Overall
Zero-shot Models				
GPT-4o	22.5	30.7	11.8	0.8
Qwen2.5-VL-3B-Instruct	48.7	9.5	36.6	1.7
Qwen2.5-VL-7B-Instruct	46.8	27.5	29.1	3.7
Screen-to-Action Training Models				
GUI-Owl-7B	61.9	21.1	41.0	5.4
w/ UILoop	87.4	51.1	53.4	23.8
OS-Atlas-Pro-7B	49.6	48.2	18.9	4.5
w/ UILoop	71.4	54.2	34.9	13.5
UI-R1-3B	47.1	39.7	33.7	6.3
GUI-R1-3B	47.4	37.9	35.9	6.4
GUI-R1-7B	62.6	47.6	35.3	10.5
UILoop Training Models				
UILoop-3B	80.3	44.7	50.2	18.0
UILoop-7B	86.4	49.3	61.3	**26.1**

Table 4: Overall performance of different paradigm methods on UI element Locate, Lingualize, and Leverage capabilities in our UI Comprehension-Bench.

5.6 Case Study

We conducted a case study as shown in Fig. 6. For the instruction "Open the Pizza Max app and add a 10 inch medium pizza to your cart with a crust," key UI elements (Green) and misleading ones (Red) have minimal visual differences. "Screen-to-Action" methods incorrectly click "P. PAN 7", while UILoop correctly identifies "Medium 10" by analyzing UI element semantics and the "ADD" button's function. UILoop also explicitly shows the reasoning process from Screen to key UI elements to Action, demonstrating superior interpretability.

6 Conclusion

In this paper, we highlight that comprehensive UI understanding significantly enhances GUI agent reasoning. We propose ***UI-in-the-Loop*** (UILoop), an innovative paradigm that reframes GUI reasoning from conventional ***"Screen-to-Action"*** to a cyclic ***"Screen–UI Elements–Action"*** loop. We design UI Element-Driven Reinforcement Fine-Tuning to improve interface element comprehension, advancing multimodal GUI reasoning and interpretability. To facilitate this research, we introduce the ***UI Comprehension*** task with three evaluation metrics (UI Locate, Lingualize, and Leverage) and contribute UI Comprehension-Bench, a 26K benchmark for comprehensive UI assessment. Extensive experiments show UILoop achieves state-of-the-art performance in UI comprehension and delivers superior results in GUI reasoning tasks.

Limitations

The primary limitations of our method encompass the following two aspects:

(1) UILoop primarily enhances the model's mastery of fine-grained UI elements but lacks consideration of UI layouts at different granularities within the screen, such as coarse-grained UI layouts composed of multiple fine-grained UI elements. In future work, we will further investigate the impact of UI elements at varying granularities on GUI reasoning capabilities.

(2) Current experiments predominantly focus on Qwen2.5-VL. In future work, we will explore the performance of UILoop across a broader range of MLLMs.

Ethics Statement

In this paper, we introduce UI Comprehension-Bench, which is derived from existing GUI reasoning datasets Android Control, OmniAct, GUI-Act, ScreenSpot, ScreenSpot-Pro, and OS-Atlas, combined with externally collected webpages, mobile apps, and OS data. Furthermore, we conducted manual verification and excluded low-quality or non-compliant data, ensuring that our synthesized data does not violate any ethics. All UI screenshots were carefully reviewed to exclude or anonymize any personal or sensitive information. To promote transparency and reproducibility, we commit to releasing all code, models, and datasets upon publication of this paper, enabling the research community to verify our findings and build upon our work.

References

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

Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A. Plummer. 2022. A dataset for interactive vision-language navigation with unknown command feasibility. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part VIII*, page 312–328, Berlin, Heidelberg. Springer-Verlag.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, Yuan Yao, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2025. GUICourse: From general vision language model to versatile GUI agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21936–21959, Vienna, Austria. Association for Computational Linguistics.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Annual Meeting of the Association for Computational Linguistics*.

De Chezelles, Thibault Le Sellier, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F Xu, Siva Reddy, Quentin Cappart, and 1 others. 2024. The browsergym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*.

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. In *The Thirteenth International Conference on Learning Representations*.

Sagar Gubbi Venkatesh, Partha Talukdar, and Srini Narayanan. 2024. UGIF-DataSet: A new dataset for cross-lingual, cross-modal sequential actions on the UI. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1390–1399, Mexico City, Mexico. Association for Computational Linguistics.

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

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

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

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. 2024. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part LXVIII*, page 161–178, Berlin, Heidelberg. Springer-Verlag.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng

Chua. 2025. Screenspot-pro: Gui grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 8778–8786.

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

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*.

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 *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19498–19508, Los Alamitos, CA, USA. IEEE Computer Society.

Xinyi Liu, Xiaoyi Zhang, Ziyun Zhang, and Yan Lu. 2025a. Ui-e2i-synth: Advancing gui grounding with large-scale instruction synthesis. *arXiv preprint arXiv:2504.11257*.

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

Yuhang Liu, Zeyu Liu, Shuanghe Zhu, Pengxiang Li, Congkai Xie, Jiasheng Wang, Xueyu Hu, Xiaotian Han, Jianbo Yuan, Xinyao Wang, and 1 others. 2025c. Infigui-g1: Advancing gui grounding with adaptive exploration policy optimization. *arXiv preprint arXiv:2508.05731*.

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

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

Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. 2024. Coco-agent: A comprehensive cognitive mllm agent for smartphone gui automation. In *Annual Meeting of the Association for Computational Linguistics*.

Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A Rodriguez, Montek Kalsi, Rabiul Awal, Nicolas Chapados, M Tamer Özsu, Aishwarya Agrawal, David Vazquez, and 1 others. 2025. Ui-vision: A desktop-centric gui benchmark for visual perception and interaction. *arXiv preprint arXiv:2503.15661*.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim, Ruiyi Zhang, Tong Yu, Mehrab Tanjim, and 11 others. 2025. GUI agents: A survey. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22522–22538, Vienna, Austria. Association for Computational Linguistics.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Hassan. 2025. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6300–6323.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, and 1 others. 2024. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*.

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, 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.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. 2022. Meta-gui: Towards multi-modal conversational agents on mobile gui. In *Conference on Empirical Methods in Natural Language Processing*.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. 2025. OS-genesis: Automating GUI agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5555–5579, Vienna, Austria. Association for Computational Linguistics.

Lei Wang, Chengbang Ma, Xueyang Feng, Zeyu Zhang, Hao ran Yang, Jingsen Zhang, Zhi-Yang Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji rong Wen. 2023. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, and 1 others. 2024. Gui agents with foundation models: A comprehensive survey. *arXiv preprint arXiv:2411.04890*.

Xuehui Wang, Zhenyu Wu, JingJing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, Qingyun Li, Xuan Dong, Zhe Chen, and 1 others. 2025a. Mmbench-gui: Hierarchical multi-platform evaluation framework for gui agents. *arXiv preprint arXiv:2507.19478*.

Yiqin Wang, Haoji Zhang, Jingqi Tian, and Yansong Tang. 2025b. Ponder & press: Advancing visual GUI agent towards general computer control. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 1461–1473, Vienna, Austria. Association for Computational Linguistics.

Hao Wen, Hongming Wang, Jiaxuan Liu, and Yuanchun Li. 2023. Droidbot-gpt: Gpt-powered ui automation for android. *arXiv preprint arXiv:2304.07061*.

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*.

Bin Xie, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. 2025. GUI-explorer: Autonomous exploration and mining of transition-aware knowledge for GUI agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5650–5667, Vienna, Austria. Association for Computational Linguistics.

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*.

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

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

Wenwen Yu, Zhibo Yang, Jianqiang Wan, Sibo Song, Jun Tang, Wenqing Cheng, Yuliang Liu, and Xiang Bai. 2025. Omniparser v2: Structured-points-of-thought for unified visual text parsing and its generality to multimodal large language models. *arXiv preprint arXiv:2502.16161*.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, and 1 others. 2024a. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279*.

Jiwen Zhang, Jihao Wu, Teng Yihua, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024b. Android in the zoo: Chain-of-action-thought for GUI agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12016–12031, Miami, Florida, USA. Association for Computational Linguistics.

A Details of UI Comprehension-Bench Collection

We elaborate on the data synthesis pipeline of UI Comprehension-Bench in this section. Our pipeline primarily consists of three steps: Source Data Collection, Key UI Element Identification and Parsing, and Human Verification.

Source Data Collection. Our data sources mainly include webpages, mobile applications, operating systems, and existing GUI reasoning datasets. For webpages, we capture screens from real browsers using BrowserGym (Chezelles et al., 2024) and Playwright [1], randomly simulate actions such as clicking, scrolling, and typing on the screens, and retain successfully executed actions. For mobile and OS data, we employ DroidBot[2] (Wen et al., 2023) to perform the same screen capture and action execution procedures on real Android applications and operating systems. We also incorporate training data from existing datasets—Android Control, OmniAct, GUI-Act, ScreenSpot, ScreenSpot-Pro, and OS-Atlas—as part of our source data. We normalize the format of all source data, with each sample containing the following data fields: (instruction, screen, action).

Key UI Element Identification and Parsing. We process the screens obtained from the source data by employing a set-of-marks model, specifically OmniParser V2, to annotate all identifiable UI elements on the screen. This enables us to obtain coordinate information for all candidate UI elements. We then utilize GPT-4o as a selection model to identify UI elements that are beneficial for completing the given instruction and to provide reasoning processes explaining how these UI elements contribute to task completion. Specifically, we input (instruction, screen, UI element coordinate information, action) into the selection model to identify key UI elements and generate their semantic functions and practical usage (detailed prompts are provided in Appendix C). Consequently, we expand the data format of the source data to (instruction, screen, key UI element information, action).

Human Verification. We conduct manual screening of the obtained data to exclude samples with incorrect instructions, erroneous answers, or misidentified key UI elements. Through this verification process, **we ultimately curate UI Comprehension-Bench, which comprises 26,207 samples, including a training set of 3,471 samples (selected from the training sets of Android Control, OmniAct, GUI-Act, ScreenSpot, ScreenSpot-Pro, and OS-Atlas) and a test set of 22,736 samples, ensuring complete data isolation between the two sets.**

[1] https://github.com/microsoft/playwright
[2] https://github.com/honeynet/droidbot

B Demonstrations of UI Comprehension-Bench

In this section, we compare UI Comprehension-Bench with existing GUI reasoning datasets and present UI Comprehension-Bench through detailed example instances. Existing GUI-reasoning datasets (including PixelHelp (Li et al., 2020), MoTIF (Burns et al., 2022), UGIF (Gubbi Venkatesh et al., 2024), Meta-GUI (Sun et al., 2022), AITW (Rawles et al., 2023), GUIAct (Chen et al., 2025), OmniACT (Kapoor et al., 2024), Android Control (Li et al., 2024), AITZ (Zhang et al., 2024b), MMBench-GUI (Wang et al., 2025a), ScreenSpot (Cheng et al., 2024), ScreenSpot-V2 (Cheng et al., 2024), ScreenSpot-Pro (Li et al., 2025), UI-E2I-Bench (Liu et al., 2025a), UI-Vision (Nayak et al., 2025)) follow the "Screen-to-Action" paradigm. Consequently, they lack fine-grained information about the location, semantic functionality, and practical usage of key UI elements on the screen, as shown in Tab. 5.

Meanwhile, we present UI Comprehension-Bench through detailed sample examples. We demonstrate the data fields and values for samples corresponding to common actions including "open_app", "type", and "click", as shown in Fig. 7, 8, 9. The blue parts indicate the data fields from the existing "Screen-to-Action" paradigm datasets, whereas our UI Comprehension-Bench additionally incorporates *Key UI Elements* and *Reasoning_Chains*, which represent the Locate, Lingualize, and Leverage information of UI elements, respectively.

C Prompt Details

Since different tasks have different action spaces, we specify the corresponding actions in prompts for each task.

For GUI grounding tasks (e.g., ScreenSpot-Pro dataset).

Instruction	Browse **Leonardo Da Vinci Mona lisa's painting** for me on the **Artsy app**.
image	<Image data - type: dict>
gt_action	**open_app**
gt_bbox	[-100, -100]
gt_input_text	**Artsy**
history	None
image_size	[1080, 2400]
group	android
Key UI Elements	["Located at [369, 634], this element displays the text **\"Mona Lisa, by Ambroise Dubois\"** and serves as a reference to a pa inting, **but not the one by Leonardo Da Vinci**.", "Located at [239, 1051], this element displays the text \"by Leonardo da Vinci,\" indicating the artist of the original Mona Lisa painting.", "Located at [134, 1207], this element displays the text \"Versions,\" which is part of the description related to the Mona L isa painting.", "Located at [207, 1522], this element displays the text **\"Monna Lisa is on,\" which is part of the description related to th e Mona Lisa painting**.", "Located at [518, 1155], this element displays the text \"The history of the painting can explain how rare are its,\" which is part of the description related to the Mona Lisa painting."]
Reasoning_Chains	["Analyze the user instruction which requests browsing Leonardo Da Vinci's Mona Lisa painting on the Artsy app.", "Locate key UI elements related to the Mona Lisa and Leonardo Da Vinci within the interface.", "**The UI elements mentioning \"Leonardo da Vinci\" and \"Monna Lisa\" are relevant as they directly relate to the user's r equest to browse the painting by Leonardo Da Vinci**.", "Determine that the correct action type is **\"open_app\"** with the target area being the **Artsy app**, as the user wants to b rowse within this specific application.", "**Input text \"Artsy\"** is necessary to identify the app to open for browsing the Mona Lisa painting by Leonardo Da Vinci."]

Figure 7: Case with *open_app* actions in our UI Comprehension-Bench.

Instruction	Browse **Leonardo Da Vinci Mona lisa's painting** for me on the **Artsy app**.
image	<Image data - type: dict>
gt_action	**type**
gt_bbox	[-100, -100]
gt_input_text	**Leonardo**
history	Step 1: Open the artsy app. Step 2: Click on the search icon at the bottom.
image_size	[1080, 2400]
group	android
Key UI Elements	["Located at **[508, 263], this element is the search bar** where users **can type queries to find artists**, artworks, galleries, etc. ", "Located at [327, 513], this element displays the text **\"Leonardo da Vinci,\" indicating a recent search or suggestion** relat ed to the user's query.", "Located at [229, 565], this element is labeled \"Artist,\" suggesting the category or type of the search result related to Leonardo da Vinci."]
Reasoning_Chains	["Analyze the user instruction which requires searching for Leonardo Da Vinci's Mona Lisa painting on the Artsy app.", "Locate the search bar UI element at [508, 263] as the primary area to input the search query.", "The search bar allows users to type queries, and the presence of \"Leonardo da Vinci\" at [327, 513] suggests it is a relev ant search term.", "Determine the action type as **'type'** with the target area being the search bar, and **input text \"Leonardo\"** to initiate the search.", "The **search bar is the correct UI element to interact with**, as it facilitates the input of search queries, aligning with the u ser's instruction."]

Figure 8: Case with *type* actions in our UI Comprehension-Bench.

Datasets	# Episodes	# Unique Instructions	Annotation						
			Screen Desc.	Key UI Element Loc.	Key UI Element Lin.	Key UI Element Lev.	Action Coord	Action Desc.	Action Think
PixelHelp	187	187	✗	✗	✗	✗	✓	✗	✗
MoTIF	4707	270	✗	✗	✗	✗	✓	✗	✗
UGIF	523	420	✗	✗	✗	✗	✓	✗	✗
Meta-GUI	4684	1125	✗	✗	✗	✗	✓	✓	✗
AITW	715142	30378	✗	✗	✗	✗	✓	✗	✗
GUIAct	5696	5696	✗	✗	✗	✗	✓	✓	✗
OmniACT	9802	-	✗	✗	✗	✗	✓	✓	✗
Android Control	15283	15283	✓	✗	✗	✗	✓	✓	✗
AITZ	2504	2504	✓	✗	✗	✗	✓	✓	✓
MMBench-GUI	8123	8123	✗	✓	✗	✗	✓	✓	✓
ScreenSpot	1272	1272	✗	✓	✗	✗	✗	✗	✗
ScreenSpot-V2	1272	1272	✗	✓	✗	✗	✗	✗	✗
ScreenSpot-Pro	1581	1581	✗	✓	✗	✗	✗	✗	✗
UI-E2I-Bench	1477	1477	✗	✓	✗	✗	✗	✗	✗
UI-Vision	8227	∼450	✓	✓	✗	✗	✓	✓	✗
Ours	26207	15735	✔	✔	✔	✔	✔	✔	✔

Table 5: Detailed comparison of our UI Comprehension-Bench with existing GUI reasoning benchmarks.

Instruction	There is an upcoming Amazon sale on **November 10** where I have planned to buy a music system. **Set a reminder for the date.**
image	<Image data - type: dict>
gt_action	**click**
gt_bbox	**[719, 1236]**
gt_input_text	no input text
history	Step 1: Open the to-do list app. Step 2: Open the to-do list app. Step 3: Click on the plus icon at the bottom right. Step 4: Enter the reminder, which is the Amazon Sale. Step 5: Now tap on the due date to select the date.
image_size	[1080, 2400]
group	android
Key UI Elements	["Located at [721, 1239], this element **represents the date \"10\" on the calendar, which is the target date for setting the reminder.**", "Located at [5, 911], this element displays \"November 2023,\" indicating the month and year context for the calendar.", "Located at [22, 1801], this element is the **\"OK\" button, which confirms the selection of the date for the reminder.**"]
Reasoning_Chains	["Analyze the instruction requirements to identify the need to set a reminder for the Amazon sale on November 10.", **"Locate key UI elements that correspond to the date and confirmation actions, such as the date \"10\" and the \"OK\" button.",** **"The UI element at [721, 1239] helps complete the task by allowing the user to select the correct date, November 10, on the calendar.",** "Determine the action type as **\"click,\"** with the target area being **[719, 1236],** which is slightly adjusted to ensure precise interaction with the date element.", **"Clicking the \"10\" button at [721, 1239]** confirms the date selection, completing the reminder setup process."]

Figure 9: Case with *click* actions in our UI Comprehension-Bench.

Prompt for Grounding

You are UILoop, a reasoning GUI Agent Assistant. In this UI screenshot <image>, I want you to continue executing the command 'text', with the action history being 'history'.
Please provide the action to perform (enumerate from ['click']), the point where the cursor is moved to (integer) if a click is performed, and any input text required to complete the action.

Output the location, semantics, and function of UI element(s) that you think are beneficial for reasoning within <ui> </ui> tags, reason based on the key UI element(s) and output the thinking process in <think> </think> tags, and the final answer in <answer> </answer> tags as follows: <ui> Located at [x, y], describe the UI element's semantics and function. </ui> <think> ... </think> <answer>['action': enum['click'],

’point’: [x, y], ’input_text’: ’no input text [default]’]</answer>
Note: For each UI element, you must provide its location [x, y], semantics, and functionality.
Example:
<ui> Located at [743, 724], this element represents the ’Slide Notes’ section where users can click to interact with notes related to a slide. </ui>
<ui> Located at [317, 501], this element is a text label that reads D̈eveloper Tools,ïndicating the section related to developer options. </ui>
Example of answer output:
[’action’: enum[’click’], ’point’: [123, 300], ’input_text’: ’no input text’]

For GUI reasoning tasks (e.g., Android Control-High dataset).

Prompt for Reasoning

You are UILoop, a reasoning GUI Agent Assistant. In this UI screenshot <image>, I want you to continue executing the command ’text’, with the action history being ’history’.
Please provide the action to perform (enumerate from [’wait’, ’long_press’, ’click’, ’press_back’, ’type’, ’open_app’, ’scroll’]), the point where the cursor is moved to (integer) if a click is performed, and any input text required to complete the action.
Output the location, semantics, and function of UI element(s) that you think are beneficial for reasoning within <ui> </ui> tags, reason based on the key UI element(s) and output the thinking process in <think> </think> tags, and the final answer in <answer> </answer> tags as follows:
<ui> Located at [x, y], describe the UI element’s semantics and function. </ui> <think> ... </think> <answer>[’action’: enum[’wait’, ’long_press’, ’click’, ’press_back’, ’type’, ’open_app’, ’scroll’], ’point’: [x, y], ’input_text’: ’no input text [default]’]</answer>
Note: For each UI element, you must provide its location [x, y], semantics, and functionality.
Example:
<ui> Located at [743, 724], this element represents the ’Slide Notes’ section where users can click to interact with notes related to a slide. </ui>
<ui> Located at [317, 501], this element is a text label that reads D̈eveloper Tools,ïndicating the section related to developer options. </ui>
Specific input text (no default) is necessary for actions enum[’type’, ’open_app’, ’scroll’] Example:
[’action’: enum[’wait’, ’press_back’], ’point’: [-100, -100], ’input_text’: ’no input text’]
[’action’: enum[’click’, ’long_press’], ’point’: [123, 300], ’input_text’: ’no input text’]
[’action’: enum[’type’, ’open_app’], ’point’: [-100, -100], ’input_text’: ’shanghai shopping mall’]
[’action’: enum[’scroll’], ’point’: [-100, -100], ’input_text’: enum[’up’, ’left’, ’right’, ’down’]]

When employing the selection model (e.g., GPT-4o) to perform Key UI Element Identification and Parsing, we design the prompt as follows.

Key UI Element Identification and Parsing

UI Element Analysis and Action Reasoning Task
Task Description You need to analyze the given user interface information, identify key UI elements that help complete the specified instruction, and explain how to reason about the correct action based on these elements.
Input Information **User Instruction:** instruction
Action History: history
Ground Truth - Action Type: gt_action
Ground Truth - Target Area: gt_bbox
Ground Truth - Input Text: gt_input_text
UI Element Information: ui_info
Output Requirements
1. UI Element Functional Descriptions Please provide a one-sentence description of the UI element’s position in the image and its semantic and functional description for each key UI element that helps complete the instruction, with each UI element description enclosed in <ui></ui> tags:
<ui>Located at [x1,y1], this element [semantic and functional description]</ui> <ui>Located at [x2,y2], this element [semantic and functional description]</ui> ...
2. Action Reasoning Process Based on the identified correct UI elements, please explain the reasoning process for deriving the correct action in no more than 5 sentences, with each thought enclosed in <think></think> tags:
<think> Analyze instruction requirements </think>
<think> Locate key UI elements </think>
<think> Explain why the UI element(s) help(s) complete the task </think>
<think> Determine action type, target area, input text </think>
<think> Other necessary thoughts... </think>
Important Notes 1. UI element descriptions must be concise and clear, one sentence per element 2. The reasoning process should be

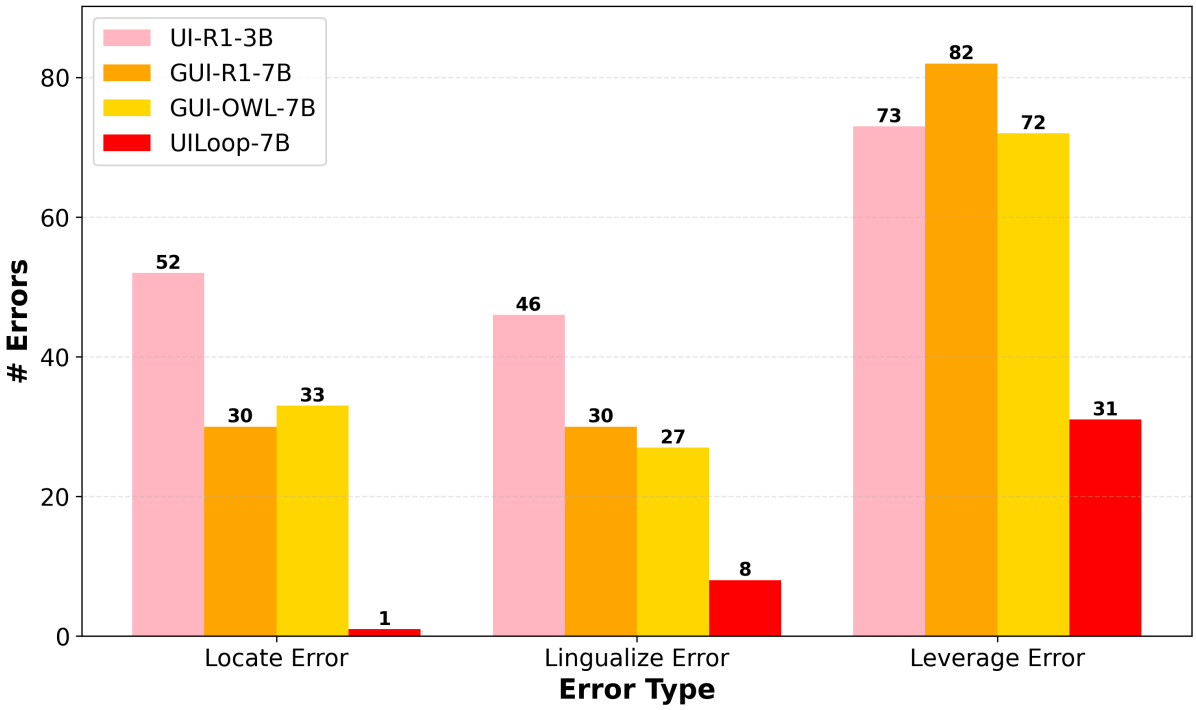

Figure 10: Error analysis of “Screen-to-Action" paradigm methods UI-R1-3B, GUI-R1-7B, GUI-OWL-7B and our method UILoop. We demonstrate that the primary error types include: (1) Locate Error, (2) Lingualize Error, and (3) Leverage Error.

logically clear, showing the complete reasoning chain from analysis to decision 3. Strictly follow the specified XML tag format for output 4. Focus on UI elements directly related to completing the instruction

D Error Analysis

In this section, we conduct a comparative error analysis between current “Screen-to-Action" paradigm methods UI-R1-3B, GUI-R1-7B, GUI-OWL-7B and our “Screen-UI Elements-Action" paradigm method. Specifically, we investigate three primary error types related to UI elements: (1) Locate Error, (2) Lingualize Error, and (3) Leverage Error. We randomly sampled 100 instances from the Android Control-High test set and performed manual statistics, as shown in Fig. 10. The results demonstrate that our method achieves error counts of 1, 8, and 31 for Locate, Lingualize, and Leverage Errors respectively, which are substantially lower than the error counts of UI-R1-3B, GUI-R1-7B, and GUI-OWL-7B. This demonstrates the advanced level of our method’s mastery over UI elements.