

## 1 Appendix

### 2 A Datasets Details

#### 3 A.1 Mind2Web

4 Mind2Web is a pioneering dataset designed to develop and evaluate generalist web agents capable  
5 of performing complex tasks on any website using natural language instructions. It features over  
6 2,000 tasks across 137 websites and 31 domains, with 7,775 training actions, offering a diverse and  
7 realistic environment for training. Unlike simulated datasets, Mind2Web uses real-world websites,  
8 providing rich data like user interaction traces and webpage snapshots. A key strength lies in its three  
9 distinct test splits—cross-task, cross-website, and cross-domain—designed to rigorously evaluate  
10 generalization performance. The dataset’s action space includes three core operations: CLICK, TYPE,  
11 and SELECT, capturing essential user interactions for navigating modern web complexities.

#### 12 A.2 AITW

13 AITW is a comprehensive Android smartphone dataset featuring 30K instructions and 715K trajec-  
14 tories, collected using the Android Emulator. We follow the experimental setup of ShowUI, dividing the  
15 data into five domains: Google Apps, Install, Web Shopping, General, and Single. The action space  
16 includes 12 actions: CLICK, TYPE, SELECT, SCROLL UP, SCROLL DOWN, SCROLL LEFT, SCROLL  
17 RIGHT, PRESS BACK, PRESS HOME, PRESS ENTER, STATUS TASK COMPLETE, and STATUS TASK  
18 IMPOSSIBLE, enabling diverse interaction analysis.

#### 19 A.3 GUIAct

20 GUIAct is a multi-scenario dataset designed to enhance GUI agents’ knowledge, covering web and  
21 smartphone environments. It includes GUI navigation tasks split into three partitions: "web-single,"  
22 "web-multi," and "smartphone," with a unified action space of 11 action types. The web dataset  
23 comprises 67K single-step and 5,696 multi-step instructions across 50 domains and 13K websites,  
24 while the smartphone dataset includes 9,157 multi-step instructions, totaling 67K training samples.  
25 Consistent with ShowUI’s experimental setup, we pretrain on GUIAct for zero-shot experiments.  
26 GUIAct uses a unified action space comprising eleven key actions: CLICK, HOVER, TAP, INPUT,  
27 SCROLL, SWIPE, SELECT TEXT, COPY, ENTER, SELECT, and ANSWER, enabling agents to interact with  
28 GUI systems across web and smartphone scenarios.

#### 29 A.4 Miniwob

30 MiniWob features 2000 open-ended tasks sourced from 137 real web environments. It includes  
31 dynamic GUI environments, allowing validation of a model’s adaptability to dynamic settings. Each  
32 task provides high-level instructions and action trajectories, enabling agents to perform low-level  
33 keyboard and mouse actions on the Internet. The action space includes 2 actions: CLICK and TYPE.

## 34 B Training Details

### 35 B.1 Pseudo-Label Generation

36 We utilize a retrospective labeling strategy to generate pseudo-labels for the agent’s intermediate  
37 reasoning and history summary, ensuring accuracy and alignment with the intended goal using  
38 known correct actions. This process creates reliable pseudo-labels for supervised learning, generated  
39 step-by-step with GPT-4o-mini. The following are the specific prompts employed in our pseudo-label  
40 generation process, which demonstrate the structured input for each step.

```
41 _PROMPT_SINGLE_WEB = """"You are an AI assistant designed to simulate the model’s reasoning process before
42 executing a given action in a gui navigation task. Given the task instruction, current screenshot,
43 the previous history summary, the current action to be executed and thought, generate a rigorous
44 chain of thought. You must strictly follow these reasoning steps:
45 (1) Progress Estimation: Interface Comprehension and Progress Estimation
46 (2) Decision Reasoning: Strategy Formulation
47 (3) History Summary: Update the history summary according the action you executed
48 """
```

```

50  ### Output format:
51  <Progress Estimation>
52  ... (one or two sentence)
53  </Progress Estimation>
54  <Decision Reasoning>
55  ... (one or two sentence)
56  </Decision Reasoning>
57  <History Summary>
58  ... (one or two sentence)
59  </History Summary>
60
61  ###Example Input & Output
62  Input:
63  Task Instruction: Find all events taking place in New York City during the month of September.
64  Current Action: {'action': CLICK, 'value': 'Apply', 'position': [0.3, 0.66]}
65  Previous History Summary: The user first changed the location to New York, then set the start date to
66  September 1, and set the end data to September 30.
67  Output:
68  <Progress Estimation>
69  The user has successfully set the location to New York and selected the date range for September 1-30, but
70  the events displayed are still for March, indicating the need to apply the date filter.
71  </Progress Estimation>
72  <Decision Reasoning>
73  Clicking the 'Apply' button will confirm the selected date range (September 1-30) and refresh the event
74  listings to show only those occurring in New York City during September.
75  </Decision Reasoning>
76  <History Summary>
77  The user changed the location to New York, set the date range to September 1-30, and applied the filters
78  to update the event listings.
79  </History Summary>
80
81  ###Input
82  Task Instruction: {_TASK}
83  Current Action: {_ACTION}
84  Thought: {_THOUGHT}
85  Previous History Summary: {_MEMO}
86  """
87

```

## 88 B.2 Action Reward Computation

89 To evaluate both the correctness and executability of predicted actions, we introduce a composite  
90 action reward function  $\mathcal{R}^a$ . This function assesses each predicted action  $\alpha_{t,i}$  based on three criteria:

91 **Action Structural Format Reward:** Whether the output adheres to the required dictionary-style  
92 format: `{‘action’: ‘ACTION_TYPE’, ‘value’: ‘element’, ‘position’: [x, y]}`.  
93 We define a function, `CheckActionF`, to check the predicted action. The function first checks if the  
94 input is a dictionary. If not, it returns False. It then verifies that the dictionary contains exactly the  
95 three required keys (“action”, “value”, and “position”) and no extra keys, the function return true:

$$r_{t,i}^{\text{af}} = \mathcal{R}^{\text{af}}(\alpha_{t,i}) = \begin{cases} 1 & \text{if CheckActionF}(\alpha_{t,i}) == \text{true} \\ 0 & \text{else} \end{cases} \quad (1)$$

96 **Action Type Reward:** We evaluate whether the predicted action type  $\alpha_{t,i}^{\text{type}}$  matches the ground-truth  
97 type. Specifically, we check if the predicted value exactly equals the annotated action type for the  
98 current step. If they match, the prediction is considered correct:

$$r_{t,i}^{\text{type}} = \mathcal{R}^{\text{type}}(\alpha_{t,i}^{\text{type}}, \alpha_{t,i}^{\text{gt type}}) = \begin{cases} 1 & \alpha_{t,i}^{\text{type}} = \alpha_{t,i}^{\text{gt type}} \\ 0 & \text{else} \end{cases} \quad (2)$$

99 **Action Position Reward:** We evaluate whether the predicted coordinates  $\mathcal{C}_{t,i}$  fall within the bounding  
100 box  $b_t = (x_1^{\text{pos}}, y_1^{\text{pos}}, x_2^{\text{pos}}, y_2^{\text{pos}}) \in \mathbb{R}^4$  of the target UI element. Specifically, we check if the predicted  
101 point lies inside the rectangular region defined by  $b_t$ :

$$r_{t,i}^{\text{pos}} = \mathcal{R}^{\text{pos}} = \begin{cases} 1 & \text{if } \mathcal{C} \text{ in } b_t \\ 0 & \text{else} \end{cases} \quad (3)$$

102 The final action reward  $r_{t,i}^a$  is computed as a weighted sum of the three components:

$$r_{t,i}^a = r_{t,i}^{\text{af}} + \lambda^{\text{type}} \cdot r_{t,i}^{\text{type}} + \lambda^{\text{pos}} \cdot r_{t,i}^{\text{pos}} \quad (4)$$

Method	Base Model	Cross-Task			Cross-Website			Cross-Domain		
		Ele.Acc	OP.F1	Step SR	Ele.Acc	OP.F1	Step SR	Ele.Acc	OP.F1	Step SR
Zero-Shot Setting										
Qwen2.5-VL-3B	Qwen2.5-VL-3B	22.2	87.1	20.1	22.5	84.3	17.0	24.8	84.3	22.8
GUI-Rise		<b>29.4</b>	<b>87.4</b>	<b>24.8</b>	<b>26.1</b>	<b>85.7</b>	<b>22.4</b>	<b>35.1</b>	<b>84.8</b>	<b>30.1</b>

Table 1: Out-of-domain evaluation results on the Mind2Web benchmark with Qwen2.5-VL-3B. The table reports performance under the zero-shot setting, where models are trained on the GUIAct training set and evaluated on the Mind2Web test set.

Method	Base Model	General	Install	G.Apps	Single	WebShop	Overall
<i>Zero-Shot Setting</i>							
Qwen2.5-VL-3B	Qwen2.5-VL-3B	35.5	43.1	41.7	35.0	39.3	38.9
GUI-Rise	VL-3B	<b>56.4</b>	<b>59.0</b>	<b>52.3</b>	<b>59.7</b>	<b>52.7</b>	<b>56.0</b>

Table 2: Evaluation results on the AITW benchmark with Qwen2.5-VL-3B. The table reports performance under the zero-shot setting, where models are trained on the GUIAct training set and evaluated on the AITW test set.

### B.3 Reinforcement Learning Objective

In the second stage of our two-phase training, we adopt GRPO. GRPO improves upon traditional Proximal Policy Optimization (PPO), surpasses traditional PPO by eliminating the need for a separate critic model. At time step  $t$ , the advantage  $\hat{A}_{i,t}$  corresponding to the  $i$ -th output  $v_i$  can be derived from Section 4.2. Then, the overall objective is:

$$b_{t,i,j}(\theta) = \frac{\pi_{\theta}(v_{t,i,j} \mid \mathbf{u}, \mathbf{o}_t, \mathbf{h}_{t-1}, v_{t,i,<j})}{\pi_{\theta_{\text{old}}}(v_{t,i,j} \mid \mathbf{u}, \mathbf{o}_t, \mathbf{h}_{t-1}, v_{t,i,<j})}. \quad (5)$$

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{(\mathbf{u}, \mathbf{o}_t, \mathbf{h}_{t-1}, \alpha_t) \sim \mathcal{D}, \{v_{t,i}\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot \mid \mathbf{u}, \mathbf{o}_t, \mathbf{h}_{t-1})} \left[ \frac{1}{G} \sum_{i=1}^G \frac{1}{|v_{t,i}|} \sum_{j=1}^{|v_{t,i}|} \left( \min \left( b_{t,i,j}(\theta) \hat{A}_{i,t}, \text{clip}(b_{t,i,j}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_{i,t} \right) - \beta D_{\text{KL}}(\pi_{\theta} \parallel \pi_{\text{ref}}) \right) \right] \quad (6)$$

where  $(\mathbf{u}, \mathbf{o}_t, \mathbf{h}_{t-1}, \alpha_t)$  is a question-answer pair from the data distribution  $\mathcal{D}$ ,  $\epsilon$  is the clipping range of importance sampling ratio. To ensure stable policy updates, GRPO also introduces a KL-divergence regularization term that penalizes deviation from the reference policy distribution, and  $\beta$  is a coefficient controlling the strength of the regularization. This formulation helps constrain policy updates, stabilizing training and encouraging consistency with previously learned behaviors.

## C Experimental Analysis

### C.1 OOD Evaluation Results

In Section 5.2, we reported zero-shot results based on Qwen2-VL-2B. To demonstrate that our framework does not rely on a specific base model, we conducted the same experiment using Qwen2.5-VL-3B. Specifically, we followed the same zero-shot setup: training on the GUIAct dataset and evaluating on the Mind2Web and AITW test sets. As shown in the table 1 and 2, GUI-Rise consistently outperforms Qwen2.5-VL across all transfer settings and metrics. It achieves a 7.3 improvement in Step SR under the Cross-Domain setting, demonstrating that our method can be effectively applied across different base models.

### C.2 Online Evaluation Results

**MiniWob.** To demonstrate the general applicability of our method across different models, we further evaluate GUI-Rise using Qwen2.5-VL-3B as the base model on the MiniWob dataset. As shown in

Online	Method	Score
MiniWob	Qwen2.5-VL-3B	23.3
	GUI-Rise-3B	<b>36.6</b>
OSWorld	UI-TARS-2B	6.5 (15 steps)
	GUI-Rise-2B	<b>8.7 (15 steps)</b>

Table 3: Zero-shot results on the online navigation benchmark MiniWob with the 35-task split and OSWorld with the chrome-task split.

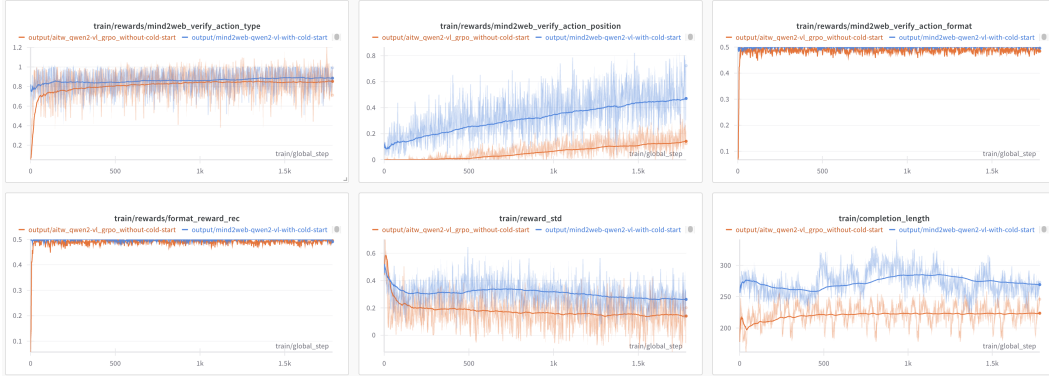


Figure 1: GUI-Rise training process on Mind2Web benchmark.

the Table 3, GUI-Rise significantly outperforms the baseline, achieving an online performance of 36.6% compared to the baseline’s 23.3%. This result further confirms the generality of our approach across different models.

**OSWorld.** To further evaluate the generalization capability of GUI-Rise, we conduct experiments on the Chrome subset of OSWorld, which includes 46 tasks involving real-world web applications. Since our training data only covers web and mobile platforms, we exclude desktop-specific tasks to ensure a fair comparison and using Qwen2-VL-2B as the base model. GUI-Rise are trained on the GUIAct training set. We compare against UI-TARS-2B, one of the current state-of-the-art models trained on large-scale data. To mitigate the effects of network instability and environmental noise, we report the average score over three runs. As shown in Table 3, GUI-Rise outperforms UI-TARS-2B by 2.1 points on the Chrome tasks. This improvement highlights the strong generalization ability of our model to previously unseen tasks.

### C.3 Impact of Cold Start

In this section, we analyze the impact of the first stage in our two-stage training framework: cold start pretraining. We compare two setups on the Mind2Web and AITW datasets: (1) Cold Start + RL and (2) RL Only. The initial model is Qwen2-VL-2B. We exclude the history summary reward in this experiment, as it depends on future actions and is unrelated to cold start effects.

Figure 1 and 2 shows the reward curves during training, revealing distinct patterns across datasets. On AITW, the action reward gap is minor at first and narrows over time, stabilizing at a difference of just 1.5 points. In contrast, on Mind2Web, cold start yields a sharp initial gain in action-type reward, while the action-position reward remains negligible without it. In fact, for the RL-only model, the position reward stays near zero throughout training, resulting in vanishing advantage estimates and ineffective gradient updates. Format reward improves rapidly on both datasets; within 30 iterations, the model consistently outputs responses in the correct format.

We attribute the differing outcomes to task complexity. Mind2Web’s visually rich web interfaces (e.g., higher resolution, dense layouts) present a steeper learning curve than AITW’s simpler mobile environments. Without a cold start, the model fails to receive meaningful reward signals early on, leading to ineffective learning.

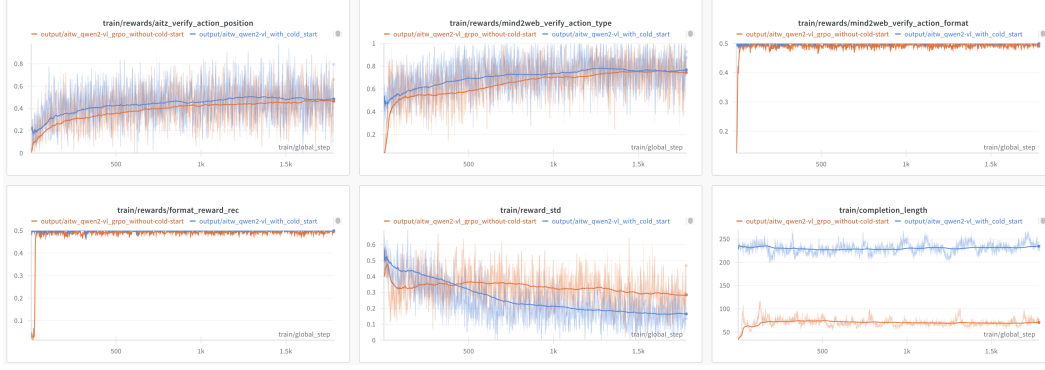
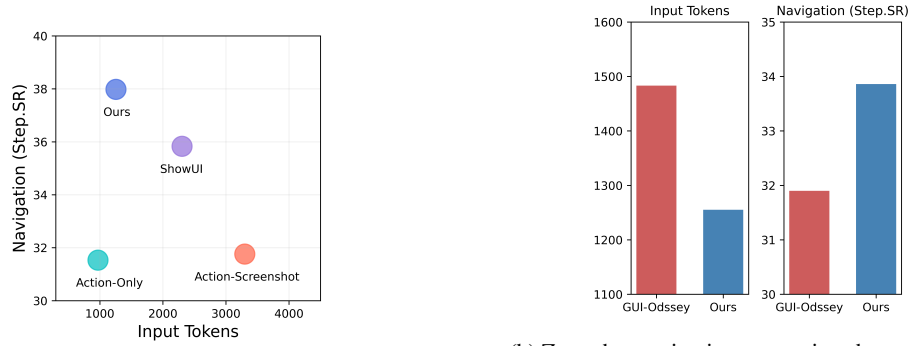


Figure 2: GUI-Rise training process on AITW benchmark.



(a) Zero-shot navigation comparison between different history methods in terms of input token cost.

(b) Zero-shot navigation comparison between GUI-Rise and GUI-Odyssey in terms of input token cost.

Figure 3: Impact by different history representation in GUI navigation in Mind2Web benchmark.

#### 154 C.4 Impact of History Representation

155 Effective history representation is essential for efficient and accurate GUI navigation. We evaluate  
 156 five strategies—Action-Only, Action+Screenshot, ShowUI, GUI-Odyssey, and our proposed history  
 157 summary representation—to assess their trade-offs between navigation success and input token  
 158 efficiency. Our experimental results shown in the Figure 3, our method achieves the highest navigation  
 159 success rate with the lowest token footprint. Compared to vision-heavy approaches like ShowUI, it  
 160 reduces visual token overhead while improving performance. Relative to GUI-Odyssey, it further  
 161 shortens input length and enhances success rates. These results highlight the importance of optimized  
 162 history representation for enabling practical, high-performance GUI interaction in MLLMs.

#### 163 C.5 Case Study

164 We select several samples from the test results of two distinct platforms to conduct a case study.  
 165 As illustrated in Figure 4, the agent demonstrates the capability to perform structured reasoning,  
 166 analyzing the progress of the current task and reasoning about decisions through a step-by-step  
 167 analytical process. Additionally, Figure 5 shows how the agent summarizes historical information  
 168 across the entire trajectory, which enables coherent reasoning in future steps.

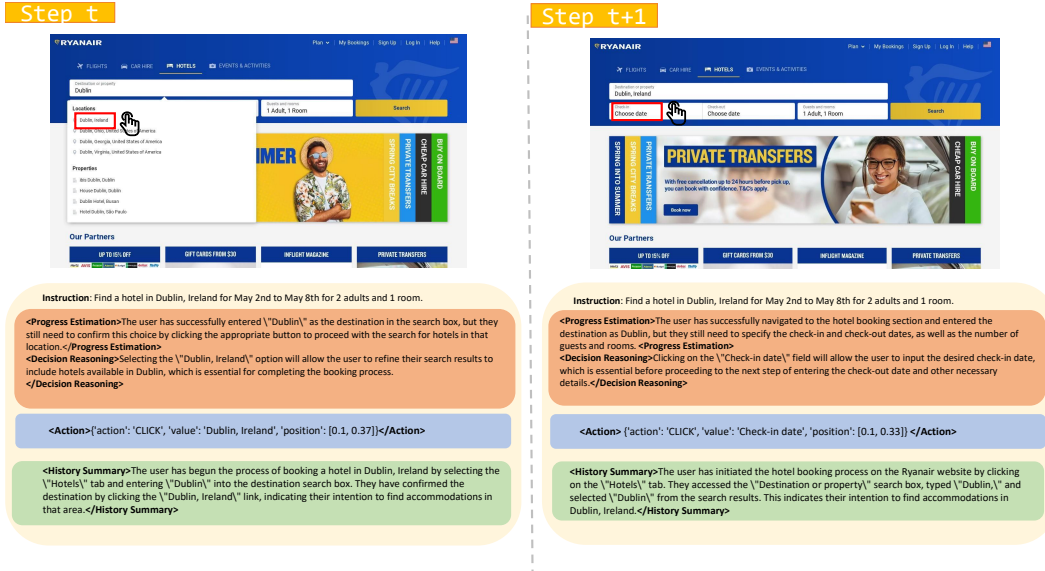


Figure 4: A case study from the Mind2Web dataset illustrates detailed reasoning across two consecutive steps.

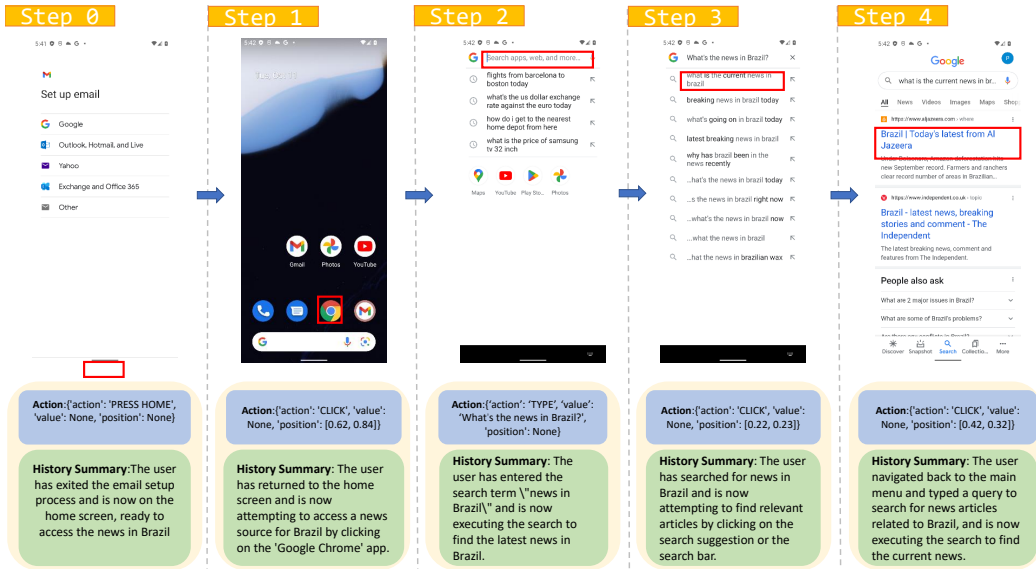


Figure 5: A case study from the AITW dataset illustrates the detailed history summarization for an entire trajectory