

# Orcust: Stepwise-Feedback Reinforcement Learning for GUI Agent

Junyu Lu, Songxin Zhang, Zejian Xie, Zhuoyang Song, Jiaxing Zhang

Lionrock AI Lab, China Merchants Research Institute of Advanced Technology

## Abstract

Recent advances in GUI agents have achieved remarkable grounding and action-prediction performance, yet existing models struggle with unreliable reward signals and limited online trajectory generation. In this paper, we introduce Orcust, a framework that integrates Principle-Constrained Reward Modeling (PCRM) and Online VM-Grounded Trajectory Construction (OVTC) to enhance reasoning reliability and data efficiency in interactive GUI tasks. We leverage environment-verifiable and LLM-derived principle to enforce interpretable reward signals that constrain long chain-of-thought reasoning and rule-based feedback. OVTC spins up instrumented virtual machines to autonomously collect structured GUI interaction trajectories with explicit procedural and structural objectives, enabling the training of a stepwise reward model that robustly captures human preferences and adheres to task-specific constraints. Extensive experiments on standard GUI benchmarks covering perceptual grounding, foundational operations, and end-to-end task execution reveal that Orcust achieves state-of-the-art performance, improving by 22.2% on ScreenSpot and 23.9% on ScreenSpot-Pro over the base model (i.e. Qwen2.5-VL-7B). The results demonstrate Orcust’s effectiveness in enhancing the reasoning, adaptability and scalability of GUI agents across various environments and task complexities.

## 1 Introduction

The rapid proliferation of graphical user interfaces (GUIs) across desktop, web, and mobile platforms has spawned a growing demand for intelligent agents capable of automating complex interaction workflows (Wu et al., 2024; Hong et al., 2024; Qin et al., 2025). Recent advances in vision-language modeling (Bai et al., 2025) have empowered agents to perceive interface elements, interpret high-level instructions (Bonatti et al.; Rawles et al., 2024), and execute atomic actions such as clicks and text entry.

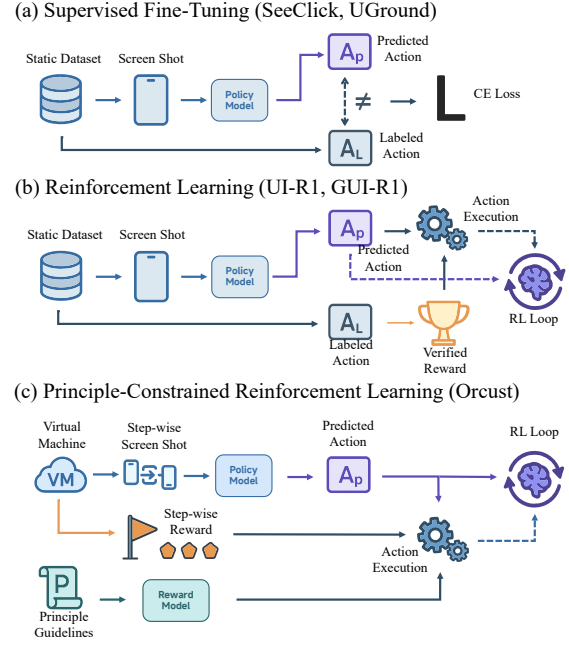


Figure 1: Comparison of GUI agent training approaches. (a) SFT on static, labeled UI interaction data. (b) RFT with trial-and-error in a GUI environment (e.g., rule-based R1 reward shaping). (c) The proposed PCRM that introduces explicit principles and stepwise, verifiable rewards into the RL training loop.

However, bridging perception with robust, long-horizon decision-making remains an open challenge, as agents must reason over dynamic layouts and ensure safety and correctness.

Prior work on GUI Agents has largely focused on visual grounding and supervised fine-tuning (Gou et al., 2024; Xu et al., 2024; Wang et al., 2024b). As shown in Figure 1 (a), vision-only grounding approaches achieve precise pixel localization of UI components but often lack deep planning capabilities and rely on static, manually annotated datasets. Conversely, supervised action prediction methods deliver strong performance on seen interfaces but struggle to generalize to unseen layouts and require large volumes of labeled data.

Recently, rule-based R1-style reinforcement fine-tuning (RFT) (Guo et al., 2025; Shao et al., 2024) and end-to-end models with iterative reflection presented in Figure 1 (b) have been introduced to enhance decision making in GUI tasks (Lu et al., 2025; Xia and Luo, 2025). These frameworks integrate structured reward signals and chain-of-thought (CoT) reasoning to improve generalization, but key limitations persist. (i) Rewards are often coarse-grained—success/failure signals delayed until the final screen—hindering credit assignment for long chains of thought. (ii) Existing trajectories are static; they neither evolve with interface updates nor capture emergent failure modes, forcing costly recollection cycles.

In this paper, we propose Orcust, a comprehensive RL framework that delivers robust, sample-efficient learning and stepwise feedback into GUI Agents as Figure 1 (c). First, Principle-Constrained Reward Modeling (PCRM) injects a dual-source principle set comprising explicit domain and implicit learned principles into a generative reward model that scores every CoT step and GUI action. Since rule compliance is checked deterministically (e.g., cursor-in-bounds, widget-state change), rewards are verifiable, preventing silent exploitation. Second, Online VM-Grounded Trajectory Construction (OVTC) spins up lightweight instances seeded with procedural task graphs; milestones declared by the agent trigger dense, step-wise rewards, enabling rapid collection of millions of high-fidelity traces without manual labeling. For example, when automating “Export chart as PDF” in LibreOffice, OVTC grants intermediate reward as soon as the agent reaches File → Export → Save action, offering fine-grained feedback unattainable in prior pipelines.

- We introduce Orcust, the first GUI-agent framework to integrate principle-guided reward mechanisms with generative critiques for reliable and interpretable feedback.
- We design an automated VM-based trajectory generator that produces millions of subgoal-annotated interactions for dense, stepwise training supervision.
- Extensive evaluations demonstrate that Orcust achieves state-of-the-art (SOTA) performance across a broad spectrum of GUI-agent tasks, including perceptual, foundational and complex multi-step scenario.

## 2 Related Work

### 2.1 GUI Agent

Early approaches (Zheng et al., 2024; Gur et al.) convert GUI structures into text or DOM representations for transformer models to select elements or generate API calls. Recently, GUI agents have progressed from models depending on HTML or accessibility trees toward fully vision-based systems that perform pixel-level operations (Qin et al., 2025; Wu et al., 2024; Wan et al., 2024; Hu et al., 2024). UGround (Gou et al., 2024) trains on large-scale synthetic web data to map natural-language region descriptions to screen coordinates, and Ponder & Press (Wang et al., 2024b) decouples high-level instruction interpretation from element localization, improving both modularity and accuracy. UI-R1 (Lu et al., 2025) first demonstrates the DeepSeek-R1 style reinforcement learning can improve the generalization of models across diverse platforms with minimal data, while GUI-R1 (Xia and Luo, 2025) further generalizes this paradigm by defining a unified action space across platforms with verifiable rewards for type, location, and text.

### 2.2 Reinforcement Fine-Tuning

Reward-centric post-training approaches such as DeepSeek-R1 (Guo et al., 2025) demonstrate that rule-based rewards can incentivize reasoning in LLMs without human preference labels. subsequent studies (Peng et al., 2025; Shen et al., 2025; Wang et al., 2025) extended RFT to multimodal Scenario by designing task-specific rewards for vision tasks. Visual-RFT (Liu et al., 2025b) and Vision-R1 (Huang et al., 2025) use environment-verifiable signals (e.g., IoU for image grounding and object detection) and a staged GRPO schedule to refine CoT reasoning, and R1-v (Chen et al., 2025) reinforce code and reasoning generation through symbolic feedback. Nevertheless, applying RFT to high-level GUI agent scenarios remains challenging due to diverse UI layouts, implicit task semantics and long-horizon action dependencies, prior RFT methods lacked advanced guidance in the reward function and did not incorporate dynamic trajectory generation. Notably, we first introduce a PCRM strategy within an RFT framework, which enables the agent to learn under explicit domain principles, and to leverage online VM-grounded trajectory construction for more scalable GUI policy learning.

### 3 Method

This section introduces two core components of Orcust: Principle-Constrained Reward Modeling (PCRM) and Online VM-Grounded Trajectory Construction (OVTC). Figure 2 illustrates the framework of Orcust.

#### 3.1 Principle-Constrained Reward Modeling

Inspired by the rule-based signals of multi-modal reinforcement framework (Peng et al., 2025; Chen et al., 2025), PCRM integrates deterministic validators with generative critiques to yield verifiable, interpretable rewards at each interaction step.

**Dual-Source Principle Generation.** We derive the set of guiding principles  $\mathcal{P}$  from two complementary sources: (1) Explicit domain principles, drawn from developer-provided rules and established user interface guidelines; and (2) Implicit learned principles, induced from data and language model insights. The explicit principles encapsulate known requirements and prohibitions in GUI tasks. For example, “A deletion action must be confirmed via a prompt” or “Fill all mandatory fields before submitting a form.”

In parallel, the implicit principles are extracted in a data-driven manner. We leverage large language models (LLMs) to analyze task descriptions and expert demonstrations, prompting the LLM to articulate additional plausible rules or strategies for accomplishing the tasks. For instance, given a high-level instruction and a successful demonstration, an LLM might infer a principle like “If an operation involves multiple steps (e.g., open a menu then click an option), maintain the correct sequence without skipping steps.” The outputs from both sources are consolidated into a unified principle set  $\mathcal{P} = p_1, p_2, \dots, p_K$ , where each  $p_i$  is represented as a check that can be evaluated against the agent’s behavior. We implement each principle  $p_i$  as a deterministic function or predicate that inspects the relevant state-action context. This function returns a boolean value  $p_i(s_t, a_t)$  indicating whether the action  $a_t$  taken in state  $s_t$  satisfies the principle. (Some principles apply only to specific actions or situations—e.g. a confirmation rule applies only to “delete” actions—so  $p_i(s_t, a_t)$  may be defined to always return True when the principle is not applicable.) By combining human knowledge with LLM-induced rules, this dual-source approach yields a robust and diverse principle set, covering

both obvious constraints and more subtle high-level strategies. The principle  $\mathcal{P}$  can be denoted as:

$$\mathcal{P} = \{p_j^{\text{human}}\}_{j=1}^m \cup \{p_i^{\text{GPT}}\}_{i=1}^n.$$

**Stepwise Verifiable Rewards.** Formally, we represent an interaction trajectory as  $\tau = \{(s_t, a_t, c_t)\}_{t=1}^T$ , where  $s_t$  is the state of the GUI environment at time step  $t$  (e.g. the current screen or DOM state),  $a_t$  is the action taken by the agent (e.g. a GUI operation like clicking or typing), and  $c_t$  is the agent’s reasoning trace (the chain-of-thought textual context) at that step.

Given the principle set  $\mathcal{P}$ , we define a reward function that evaluates the agent’s behavior at each time step by verifying principle compliance. Inspired by the process supervision methods that judge each reasoning step (Lightman et al.; Wang et al., 2024a), our reward modeling provides stepwise feedback, guiding the agent through long tasks, rather than use standard sparse rewards that only signal final success or failure. At time step  $t$ , the total reward for a trajectory  $R(\tau)$  is computed as the sum of two components: (1) Environment-verifiable principle (EVP) reward  $r^{\text{env}}(s_t, a_t \mid \mathcal{P})$  is determined by deterministic checks that verify the correctness of the agent’s actions. Examples include ensuring that the cursor remains within bounds, the clicked widget is visible and interactive, and no crashes or unintended behaviors occur during interaction. These checks offer rule-based verification, making this reward component both transparent and reliable. (2) LLM-Derived Principle (LDP) reward  $r^{\text{critique}}(c_t \mid \mathcal{P})$  provides a complementary, dense feedback signal by evaluating the quality of the agent’s reasoning and action against the principle set  $\mathcal{P}$ . We deploy a generative reward model  $r_\theta$ , which takes as input the current principles and the agent’s thought process for the step, and outputs a critique feedback and a numerical score (ranging from 1 to 10) for each reasoning step  $c_t$ . The model scores the agent’s adherence to the pre-defined principles, such as whether the agent is following the semantic search principle or any other relevant guideline. This mechanism resembles constitutional AI-style self-critique, where an LLM judges outputs by ethical or logical principles (Bai et al., 2022; Sharma et al., 2025; Liu et al., 2025a). For instance, if the agent reasons “Proceeding to delete the file without checking for save”, violating the principle “ensure no unsaved changes before deletion”, the critique model flags

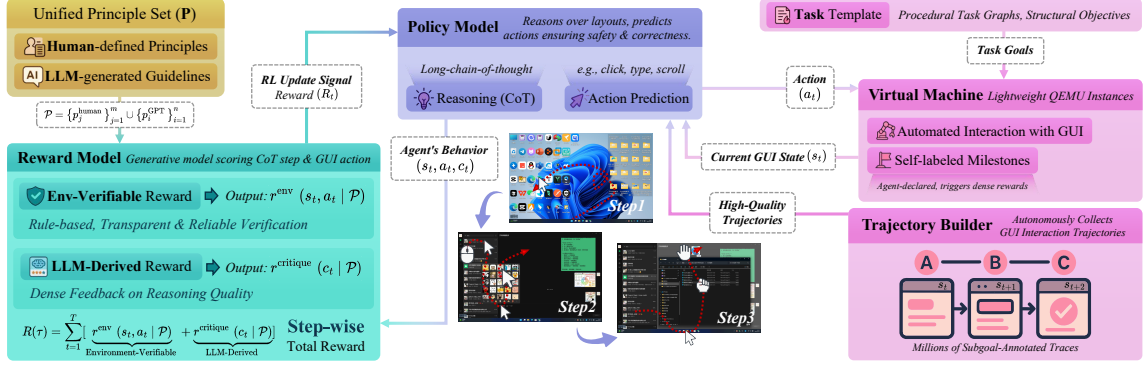


Figure 2: Architectural overview of the Orcust framework, which integrates Principle-Constrained Reward Modeling (PCRM) for stepwise, principle-aligned reward signals and Online VM-Grounded Trajectory Construction (OVTC) for automatic generation of high-quality, multi-step GUI interaction trajectories using a virtual machine environment.

this and assigns a negative reward. Conversely, if the reasoning is thorough and compliant with all principles, the critique model assigns a positive reward. The result function can be defined as:

$$R(\tau) = \sum_{t=1}^T \left[ \underbrace{r^{\text{env}}(s_t, a_t | \mathcal{P})}_{\text{Environment-Verifiable}} + \underbrace{r^{\text{critique}}(c_t | \mathcal{P})}_{\text{LLM-Derived}} \right].$$

We optimize the policy via GRPO, which maximizes the advantage of each trajectory:

$$A(\tau) = R(\tau) - \hat{b}(\text{group}),$$

where  $\hat{b}(\text{group})$  represents the group-specific baseline for stability. This two-term reward ensures that every action is both verifiable (via  $r^{\text{env}}$ ) and interpretable (via  $r^{\text{critique}}$ ), and that reward-hacking is constrained by immutable rule checks.

### 3.2 Online VM-Grounded Trajectory Construction

Collecting high-fidelity trajectories through simulation is critical for data-efficient policy learning, especially in interactive GUI and web domains where real-world labeled sequences are scarce (Qin et al., 2025; Bai et al.). OVTC is the mechanism for automatically generating diverse, high-quality trajectories complete with intermediate reward annotations, without relying on manual labeling.

**Virtual-Machine Harness.** As illustrated in Figure 2, we build a custom VM environment using lightweight QEMU/KVM instances to simulate various GUI applications and websites in a controlled setting, which are instrumented with an event bus to record every interaction with the GUI, including:

(1) **Screen RGB** captures the visual state of the GUI at every time step, providing the agent with the most up-to-date image of the interface. (2) **Input event** logs every user input (e.g., mouse clicks, text entries) so the agent can correlate actions with their effects on the GUI. (3) **DOM snapshot** records the structure of the webpage or application and underlying elements of the interface, such as buttons, fields and containers, ensuring precise verification of agent actions against the corresponding DOM information.

Each task in the VM is defined by a task template which specifies a high-level goal and a sequence of requisite sub-tasks. Each task within the VM is defined by a task template that specifies a high-level objective along with its corresponding subtasks and success criteria. For example, the task "export document as PDF in a text editor" decomposes into: (1) open the "File" menu (menu DOM node becomes open and visible); (2) click "Export" and select PDF (file selection dialog appears or PDF file is created in the file system); (3) confirm the save dialog (confirmation message is displayed). These criteria are akin to environment-based checks that can be done automatically by monitoring the DOM or system events. By using a range of task templates, we cover a variety of applications (file editors, email clients, web forms, etc.) and user goals.

### 3.3 Self-Labeled Sub-Goals.

GUI tasks, especially high-level ones, often require a sequence of several low-level actions to complete. To tackle the challenge of long-horizon credit assignment, we incorporate self-labeled sub-goals into the training process. For example, "Book a flight ticket" might involve sub-tasks like logging



in, entering passenger info, selecting a flight, and confirming payment. Our agent auto-generates its own sub-goal markers within its reasoning process. We implement this by allowing the agent’s chain-of-thought (CoT) to emit special milestone tokens whenever it believes it has completed a meaningful intermediate objective. Concretely, we augment the agent’s output format so that it can produce a token like [MILESTONE: ...] (or a similar notation) in the <think> reasoning stream, describing the accomplishment. For instance, after the agent fills out a form in a GUI, it might insert [MILESTONE: FormFilled] in its reasoning trace before proceeding to the next step. These self-labeled sub-goals serve two purposes: (1) they let the agent reflect on its progress and (2) they enable the training system to automatically detect sub-goal completion and assign appropriate rewards. Upon detecting a milestone, the environment flags it in the agent’s observation and the reward model assigns a positive reward, or applies a penalty if a step was missed or incorrect.

## 4 Experiments

### 4.1 Experimental Setup

**Training and Inference Procedures.** We instantiate Orcust in 3B and 7B parameters using a Qwen2.5-VL (Bai et al., 2025) backbone. To initialize the policy and avoid instability, we first perform a brief SFT on a small curated dataset of high-quality GUI interaction trajectories, and then conduct RFT using our PCRM reward. The OVTC procedure continually generates 15K interaction trajectories by deploying the agent in a virtual machine environment, so the model can practice tasks in a realistic GUI setting.

**Evaluation Benchmarks.** We evaluate Orcust on eight standard GUI agent benchmarks, covering mobile, desktop and web environments. Low-level benchmarks include AndroidControl-Low (Li et al., 2024) (mobile UI step-by-step tasks), GUI-Act-Web (Chen et al., 2024), OmniAct-Web and OmniAct-Desktop (Kapoor et al., 2024) (web platform and desktop environment actions). High-level, multi-step benchmarks include AndroidControl-High (Li et al., 2024) (mobile long-horizon tasks) and GUI-Odyssey (Lu et al., 2024) (cross-application navigation scenarios). For assessing visual grounding capabilities, we use the ScreenSpot (Cheng et al., 2024) benchmark that

spans GUI understanding tasks on mobile/desktop/web, and the ScreenSpot-Pro (Li et al.), featuring high-resolution professional UIs with expert-annotated targets, covering 23 apps across 5 industries and 3 operating systems.

**Evaluation Metrics.** Following Os-Atlas (Wu et al., 2024), we use the standard metrics for GUI agent evaluation, including Type accuracy (exact match of the predicted action type, e.g. Click vs Scroll), Grounding accuracy (correctness of the predicted click position or target UI element), and Stepwise Success Rate (a step is successful only if both the action type and all associated arguments, e.g. click coordinates or input text are correct).

### 4.2 Main Result

We compare Orcust-3B/7B with prior SOTA models, including the reinforcement-learned UI-R1 (Lu et al., 2025) and GUI-R1 (Xia and Luo, 2025), as well as strong supervised models UGround (Gou et al., 2024), OS-Atlas (Wu et al., 2024).

**Grounding Capability.** We summarize the GUI grounding accuracy results on the ScreenSpot and ScreenSpot-Pro benchmarks in Table 1. Orcust exhibits SOTA grounding performance, significantly outperforming all baselines in both text and icon grounding. Orcust-3B achieves higher average accuracy than the previous SOTA model OS-Atlas-7B, underscoring the data-efficiency of our principle-aligned RL approach (15K trajectories versus 14M SFT samples). Notably, Orcust-7B attains an average of 88.8% accuracy on the ScreenSpot (Web + Desktop) tasks and 39.0% on text and icon categories across six ScreenSpot-Pro domains, about 5.7% and 13.8% higher than the GUI-R1-7B. We attribute these gains to the principle-based rewards guiding the model to focus on correct high-level grounding, allowing Orcust to leverage small but diverse data for generalizable GUI understanding. We also observe that simply fine-tuning the base model on high-quality data can improve grounding, for instance, Qwen2.5-VL-3B/7B\* supervised on the OVTC-15K dataset outperforms its zero-shot counterparts by roughly 4.8% and 6.4% points on ScreenSpot. Nonetheless, the principle-constrained reward modeling leads to far greater gains, guiding the agent to align UI elements with instructions more effectively.

**Low-level Task Capability.** We evaluate low-level action execution on the four step-by-

Table 1: GUI-grounding accuracy (%) on SCREENSPOT-PRO and SCREENSPOT. All models share the same zero-shot prompt for fairness. \* indicates supervised fine-tuning on OVTC-15K.

Model	SCREENSPOT-PRO												SCREENSPOT			
	Dev		Creative		CAD		Scientific		Office		OS		Web		Desktop	
	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Text	Icon	Text	Icon
<i>Zero-shot</i>																
Qwen-VL-7B	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.0	0.0	0.0	0.0	–	–	–	–
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	–	–	–	–
Qwen2.5-VL-3B	14.9	2.1	20.2	1.4	4.1	4.7	34.0	7.3	22.0	3.8	6.5	2.2	60.0	43.2	80.9	40.0
Qwen2.5-VL-7B	28.6	2.1	22.3	2.8	11.2	4.7	35.9	7.3	32.4	5.7	25.6	5.6	70.9	51.7	84.6	48.2
<i>Supervised fine-tuning</i>																
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	55.7	32.5	72.2	30.0
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	82.6	63.1	72.1	45.7
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	90.8	74.2	91.7	62.8
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	–	–	–	–
CogAgent-18B	14.9	0.7	9.6	0.0	7.1	3.1	22.2	1.8	13.0	0.0	5.6	0.0	70.4	28.6	74.2	20.0
Aria-GUI	16.2	0.0	23.7	2.1	7.6	1.6	27.1	6.4	20.3	1.9	4.7	0.0	–	–	–	–
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	80.4	70.4	82.5	63.6
Qwen2.5-VL-3B*	18.7	2.1	25.1	2.8	12.3	5.1	38.6	6.8	29.2	5.7	18.3	2.2	64.4	46.3	84.1	48.5
Qwen2.5-VL-7B*	33.0	2.8	28.8	3.5	14.9	4.7	42.6	7.3	38.2	8.9	31.9	5.6	77.9	57.4	87.0	58.5
<i>Reinforcement fine-tuning</i>																
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	85.2	73.3	90.2	59.3
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	89.6	72.1	93.8	64.8
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	91.3	75.7	91.8	73.6
<b>Orcust-3B</b>	48.6	7.0	49.2	9.1	<b>37.2</b>	9.4	65.2	19.1	62.7	26.4	40.4	12.4	93.6	77.2	<b>97.2</b>	72.2
<b>Orcust-7B</b>	<b>60.4</b>	<b>12.4</b>	<b>58.3</b>	<b>12.2</b>	35.8	<b>14.1</b>	<b>69.5</b>	<b>24.9</b>	<b>73.3</b>	<b>32.3</b>	<b>56.6</b>	<b>18.2</b>	<b>95.8</b>	<b>83.7</b>	95.8	<b>79.8</b>

step benchmarks (GUI-Act-Web, OmniAct-Web, OmniAct-Desktop, AndroidControl-Low). As shown in Table 2, Orcust-7B delivers the best performance on every metric, achieving 96.4% action type accuracy, 92.4% grounding accuracy, 82.8% step-wise accuracy, and an overall success rate of 88.1%, substantially outperforming the previous best 7B model (GUI-R1-7B at 83.3%) and the strongest supervised baseline (OS-Atlas at 70.1%). Even the smaller Orcust-3B model (85.9% success) surpasses the GUI-R1-7B, underscoring the efficiency of our principle-constrained RL approach. We observe that Orcust mitigates the overfitting issues seen in small-scale SFT, which even degrade certain low-level metrics (e.g. grounding accuracy on AndroidControl-Low), whereas Orcust’s policy remains robust on single-step action execution across mobile, web, and desktop tasks.

**High-level Task Capability.** As illustrated in Table 3, Orcust pronounce significant advantages across high-level task execution on AndroidControl-High and GUI-Odyssey, which involve multi-step tasks and cross-application navigation. Orcust-3B outperforms the previously proposed RL methods UI-R1-3B and GUI-R1-3B by

approximately 14.39% and 10.99% respectively, while Orcust-7B achieves an average score of 67.80%, significantly surpassing all baseline models. Furthermore, we observe that Qwen2.5-VL exhibits marginal performance improvement after supervised fine-tuning. It demonstrate the critical impact of PCRM in long-horizon reasoning. By encoding high-level GUI principles into the reward function, PCRM guides the agent to plan more coherently and avoid faulty action sequences, leading to markedly higher goal-completion rates. We see that Orcust not only selects correct action types but also executes them with greater accuracy. Compared to earlier methods, Orcust’s trajectories have fewer failed steps and less deviation from the task goal, which boosts the overall success in complex multi-step scenarios. In short, PCRM enables more reliable action planning and proper GUI usage principles, thus maintains high success even as task length grows, a key differentiator on tasks like cross-app navigation and form-filled automation.

### 4.3 Ablation Study

To understand the impact of our reward type, data curation and reward granularity in Orcust, we con-

Table 2: GUI low-level task accuracy on ANDROIDCONTROL-LOW, GUI-ACT-WEB, OMNI-ACT-WEB and OMNI-ACT-DESKTOP. All experiments are conducted under the same zero-shot prompt for fair comparison.

Models	GUI-ACT-WEB			OMNI-ACT-WEB			OMNI-ACT-DESKTOP			ANDROIDCONTROL-LOW			Overall
	Type	GR	SR	Type	GR	SR	Type	GR	SR	Type	GR	SR	
Supervised Fine-Tuning													
Os-Atlas-4B	79.22	58.57	42.62	46.74	49.24	22.99	63.30	42.55	26.94	64.58	71.19	40.62	50.71
Os-Atlas-7B	86.95	75.61	57.02	85.63	69.35	59.15	90.24	62.87	56.73	73.00	73.37	50.94	70.07
Reinforcement Fine-Tuning													
UI-R1-3B	75.89	79.43	67.31	75.42	61.35	61.33	73.41	64.12	63.98	79.15	82.41	66.44	70.85
GUI-R1-3B	89.86	87.42	76.31	88.58	75.10	75.08	91.86	78.37	78.31	83.68	81.59	64.41	80.88
GUI-R1-7B	90.85	88.06	80.31	91.16	77.29	77.35	92.20	83.36	83.33	85.17	84.02	66.52	83.30
Orcust-3B	94.23	93.14	83.22	94.77	79.15	81.63	95.88	84.55	84.73	86.32	84.94	68.19	85.90
Orcust-7B	96.41	92.44	87.76	94.24	81.83	83.25	95.15	87.94	88.02	89.75	87.85	72.32	88.08

Table 3: GUI high-level task accuracy on GUI-ODYSSEY and ANDROIDCONTROL-HIGH. \* denotes supervised fine-tuned on OVTC-15K.

Models	ANDROIDCONTROL-HIGH			GUI-ODYSSEY		
	Type	GR	SR	Type	GR	SR
<i>Zero Shot</i>						
GPT-4o	63.06	30.90	21.17	37.50	14.17	5.36
Qwen2.5-VL-3B	45.66	47.23	36.18	35.10	27.15	28.06
Qwen2.5-VL-7B	59.70	55.35	45.13	49.20	38.29	33.85
<i>Supervised Fine-Tuning</i>						
OS-Atlas-4B	49.01	49.51	22.77	49.63	34.63	20.25
OS-Atlas-7B	57.44	54.90	29.83	60.42	39.74	26.96
QwenVL2.5-3B*	50.17	47.55	37.29	37.18	28.33	27.25
QwenVL2.5-7B*	58.32	56.17	44.18	50.54	37.72	35.10
<i>Reinforcement Fine-Tuning</i>						
UI-R1-3B	57.85	55.70	45.44	52.16	34.46	32.49
GUI-R1-3B	58.04	56.24	46.55	54.84	41.52	41.33
GUI-R1-7B	71.63	65.56	51.67	65.49	43.64	38.79
<b>Orcust-3B</b>	72.55	66.18	57.29	69.12	<b>54.15</b>	51.15
<b>Orcust-7B</b>	<b>79.12</b>	<b>79.42</b>	<b>62.93</b>	<b>75.25</b>	53.88	<b>56.34</b>

duct extensive ablation experiments on Orcust-7B.

**Reward Function Types.** We ablate the reward function to compare the contributions of Environment-Verifiable Principles (EVP) versus LLM-Derived Principles (LDP) in Table 4. The hybrid EVP&LDP consistently achieves the average score of 78.57% on the AndroidControl-Low and AndroidControl-High tasks, outperforming the EVP-only variant by 2.73% and LDP-only variant by 2.30%. Specifically, EVP covers immediate, factual correctness to prevent outright mistakes, while LDP encourages strategic coherence and adherence to the instruction’s intent. Together, the hybrid approach inherits the strengths and offers a more comprehensive feedback. In qualitative terms, the EVP-only agent tends to “click correctly” but sometimes

Table 4: Ablation Study in the reward function.

Reward Type	AndroidControl-Low			AndroidControl-High		
	Type	GR	SR	Type	GR	SR
EVP-only	87.17	86.25	69.12	77.09	77.87	57.52
LDP-only	86.82	85.42	70.31	77.17	77.65	60.23
<b>EVP &amp; LDP</b>	<b>89.75</b>	<b>87.85</b>	<b>72.32</b>	<b>79.12</b>	<b>79.42</b>	<b>62.93</b>

Table 5: Ablation Study in the average reward step.

Reward Step	AndroidControl-Low			AndroidControl-High		
	Type	GR	SR	Type	GR	SR
1-step	87.33	86.45	69.33	76.49	77.11	58.19
2-step	<b>90.85</b>	86.41	71.61	78.36	<b>80.55</b>	61.32
4-step	89.75	<b>87.85</b>	<b>72.32</b>	<b>79.12</b>	79.42	<b>62.93</b>
8-step	87.42	86.13	71.89	77.60	77.95	61.66

lacks context and follows instructions too literally, whereas the LDP-only agent is more context-aware but can occasionally drift since the LLM feedback isn’t tied to guaranteed ground-truth.

**Stepwise Reward Depth.** We investigate how the number of interaction steps over which rewards are averaged or assigned impacts learning. As illustrated in Table 5, we observe that an intermediate feedback horizon yields the best performance, whereas too short or too long intervals underperform. In AndroidControl-Low, using a 4-step reward produces the highest SR of 72.32%, versus 69.33% with 1-step feedback. A similar improvement appears in the harder AndroidControl-High tasks, where the SR rises from 58.19% at 1-step to 62.93% at 4-step feedback. It confirms that short-horizon rewards may fail to credit longer-term actions, while overly delayed reward make credit assignment difficult, slightly reducing performance.

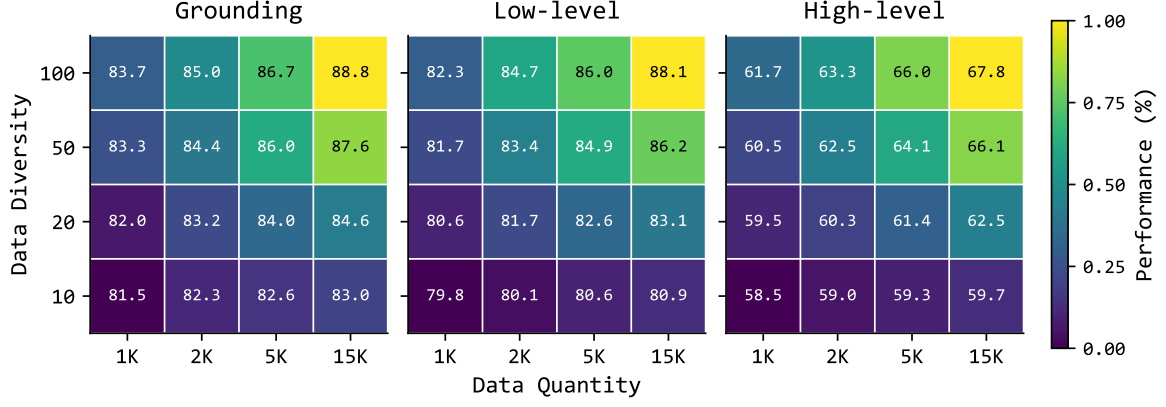


Figure 3: Performance heatmap (%) vs. data diversity (vertical) and data quantity (horizontal).



Figure 4: Reward score convergence with varying data quality and image resolution.

Thus, a 4-step reward interval provides the most informative feedback for policy learning.

**Trajectory Diversity and Quantity.** In Figure 3, we analyze the impact of training dataset diversity (number of distinct tasks/UI contexts) and quantity (number of trajectories) on the agent’s performance at the average score of grounding (ScreenSpot), low-level and high-level tasks. We observe that under the lowest data diversity setting, increasing the data quantity from 1K to 15K results in performance improvements of only 1.5%, 1.1% and 1.8% across the three tasks, respectively, whereas the improvements are 5.1%, 5.8% and 6.1% under the highest data diversity setting. Moreover, the results demonstrate that diversity provides strong gains in any data regime, while insufficient data limits the performance ceiling. Therefore, both dimensions matter that an abundance of training trajectories alone cannot cover all cases if they lack diversity, and a diverse set of tasks will still leave many gaps if it’s too small. It justifies that OVTC gathers a reasonably large number of trajectories spanning many different GUI contexts, enabling Orcust to learn skills that transfer broadly.

**Data Quality and Image Resolution.** As shown in the reward curves of Figure 4, using high-quality VM-grounded trajectories and high-resolution screenshots significantly accelerates learning and improves the final reward attainment. The High Quality + High Resolution setting achieves a high reward score within the first 20 training steps and eventually plateaus around 90%, whereas the Low Quality + Low Resolution baseline climbs much more slowly and only reaches 60% even after 100 training steps. It indicates higher image resolution clearly enables the agent to perceive fine UI details such as small text, tiny icons, which would be blurred or lost in low-res inputs, thus improving its grounding accuracy and confidence. Likewise, filtering out low-quality trajectories (e.g. those with wrong labels or irrelevant steps) accelerates learning.

## 5 Conclusion

In this work, we present Orcust, a novel framework for GUI agents that seamlessly integrates Principle-Constrained Reward Modeling with Online VM-Grounded Trajectory Construction. PCRM combines verifiable, rule-based rewards with LLM-derived critiques to provide stepwise, interpretable feedback, while OVTC harnesses lightweight VM to autonomously generate millions of high-fidelity interaction traces. Experiments demonstrate that Orcust achieves SOTA performance across eight GUI benchmarks, including ScreenSpot and ScreenSpot-Pro, highlighting that the integration principle-based reward constraints with scalable data collection significantly enhances robustness and data efficiency in interactive GUI tasks.



## Limitations

Despite these promising results, our approach has several limitations. First, Orcust relies on simulated VM environments for training and evaluation. While this ensures controlled and repeatable experiments, it may not capture the full variability of real-world GUIs. Then, the computational overhead of our method is significant: running VM-based simulations and computing dense stepwise rewards (including querying an LLM for critique feedback) incurs a high cost. This complexity could impede scalability or real-time use, as training and inference require substantial resources and careful optimization.

Another concern is deploying an autonomous GUI agent in practical settings raises ethical and security considerations. Its advanced automation capabilities could be misapplied to sensitive or critical GUI tasks (for example, manipulating financial or personal data through the interface) without proper authorization or oversight. There is also a privacy risk when scaling to real-world interfaces, as the agent may handle or observe user data on-screen. These issues highlight the need for careful safeguards, transparency, and possibly policy constraints before applying Orcust to real-world applications. Ensuring that the system’s actions remain ethical, user-approved, and within intended bounds will be crucial as we move from simulation to deployment.

## References

- Hao Bai, Yifei Zhou, Jiayi Pan, Mert Cemri, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibao Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, and 1 others. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.
- Rogério Bonatti, Dan Zhao, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Buckner, Lawrence Keunho Jang, and 1 others. Windows agent arena: Evaluating multi-modal os agents at scale. In *NeurIPS 2024 Workshop on Open-World Agents*.
- Liang Chen, Lei Li, Haozhe Zhao, Yifan Song, and Vinci. 2025. R1-v: Reinforcing super generalization ability in vision-language models with less than \$3. <https://github.com/Deep-Agent/R1-V>. Accessed: 2025-02-02.
- Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, and 1 others. 2024. Guicourse: From general vision language models to versatile gui agents. *arXiv preprint arXiv:2406.11317*.
- 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.
- Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2024. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shitong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Izzeddin Gur, Hiroki Furuta, Austin V Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis. In *The Twelfth International Conference on Learning Representations*.
- Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.
- Siyuan Hu, Mingyu Ouyang, Difei Gao, and Mike Zheng Shou. 2024. The dawn of gui agent: A preliminary case study with claude 3.5 computer use. *arXiv preprint arXiv:2411.10323*.
- Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. 2025. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint arXiv:2503.06749*.
- Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. 2024. Omniaact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *European Conference on Computer Vision*, pages 161–178.

- Kaixin Li, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, Tat-Seng Chua, and 1 others. Screenspot-pro: Gui grounding for professional high-resolution computer use. In *Workshop on Reasoning and Planning for Large Language Models*.
- Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on computer control agents. *arXiv e-prints*, pages arXiv-2406.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*.
- Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. 2025a. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*.
- Ziyu Liu, Zeyi Sun, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Haodong Duan, Dahua Lin, and Jiaqi Wang. 2025b. Visual-rft: Visual reinforcement fine-tuning. *CoRR*.
- 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. *CoRR*.
- Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanqing Xiong, and Hongsheng Li. 2025. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*.
- Yingzhe Peng, Gongrui Zhang, Miaosen Zhang, Zhiyuan You, Jie Liu, Qipeng Zhu, Kai Yang, Xingzhong Xu, Xin Geng, and Xu Yang. 2025. Lmm-r1: Empowering 3b llms with strong reasoning abilities through two-stage rule-based rl. *arXiv preprint arXiv:2503.07536*.
- Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Uitars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.
- Christopher Rawles, Sarah Clinckemaulle, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, and 1 others. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseek-math: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Mrinank Sharma, Meg Tong, Jesse Mu, Jerry Wei, Jorrit Kruthoff, Scott Goodfriend, Euan Ong, Alwin Peng, Raj Agarwal, Cem Anil, and 1 others. 2025. Constitutional classifiers: Defending against universal jailbreaks across thousands of hours of red teaming. *arXiv preprint arXiv:2501.18837*.
- Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, and 1 others. 2025. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*.
- Jianqiang Wan, Sibong Song, Wenwen Yu, Yuliang Liu, Wenqing Cheng, Fei Huang, Xiang Bai, Cong Yao, and Zhibo Yang. 2024. Omniparser: A unified framework for text spotting key information extraction and table recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15641–15653.
- Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024a. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439.
- Weiyun Wang, Zhangwei Gao, Lianjie Chen, Zhe Chen, Jinguo Zhu, Xiangyu Zhao, Yangzhou Liu, Yue Cao, Shenglong Ye, Xizhou Zhu, and 1 others. 2025. Visualprm: An effective process reward model for multimodal reasoning. *arXiv preprint arXiv:2503.10291*.
- Yiqin Wang, Haoji Zhang, Jingqi Tian, and Yansong Tang. 2024b. Ponder & press: Advancing visual gui agent towards general computer control. *arXiv preprint arXiv:2412.01268*.
- 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*.
- Xiaobo Xia and Run Luo. 2025. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*.
- Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2024. Aguis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*.
- Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v (ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning*, pages 61349–61385. PMLR.

## A Appendix

### A.1 Prompt for Training and Inference

During training, the Orcust agent is guided by Principle-Constrained Reward Modeling (PCRM) prompts. These prompts integrate explicit domain principles and implicit LLM-derived rules directly into the feedback loop. After each action, the prompt includes annotated checks for each principle/rule and computes a stepwise reward. This yields a dense reward signal at every step, with contributions from deterministic rule checks and LLM-based critiques. Listing 1 are examples for training and inference prompt in raw text form, showing how rules are enforced and how rewards are scored at each step.

During reinforcement fine-tuning, each turn in an episode is formatted as a single concatenated prompt. This prompt is designed to provide the agent with all necessary context and to structure its output for effective learning. The prompt includes: (i) The high-level task description. (ii) The current GUI screenshot, representing the state  $s_t$ . (iii) An instruction stub that enforces the `<think>-<answer>` structure.

This `<think>-<answer>` structure encourages the agent to first output its chain-of-thought (CoT) reasoning within the `<think>` tags. This reasoning can include its step-by-step plan, assessment of the current GUI state, and importantly, self-identified sub-goals denoted by `[MILESTONE: ...]` tokens. Following the reasoning, the agent provides the GUI action in JSON format within the `<answer>` tags. This structured output is crucial for PCRM, which evaluates both the reasoning and the action to provide stepwise, interpretable rewards.

#### Explanation of Key Elements.

1. **Stepwise Input:** The prompt provides the current state ( $s_t$  via screenshot) and task context at each step  $t$ .

2. **CoT Generation (<think> block):**

- The agent generates its reasoning  $c_t$  here. This is crucial for the **LLM-Derived Principle (LDP)** reward component of PCRM. The generative reward model ( $r_c$ ) evaluates this  $c_t$  against the principle set  $\mathcal{P}$ .
- **Self-Labeled Sub-Goals:** The agent can emit `[MILESTONE: FormFilled]` or `[MILESTONE: NavigatedToExportMenu]` tokens within

this block. The OVTC environment detects these, and the reward model assigns a positive reward (or penalty if incorrect/missed), contributing to dense, step-wise rewards.

3. **Action Generation (<answer> block):**

- The agent predicts an atomic action  $a_t$ . This action is primarily evaluated by the **Environment-Verifiable Principle (EVP)** reward component of PCRM. Checks include cursor-in-bounds, widget visibility/interactivity, etc.
- The JSON structure ensures the action is parseable and directly executable in the VM environment.

4. **Online VM-Grounded Trajectory Construction (OVTC):** This prompt is used by the agent deployed within the VM. The agent's `<answer>` is executed in the VM, leading to a new state  $s_{t+1}$ , and the cycle continues, generating trajectories for RFT.

#### Placeholders.

- `{SCREENSHOT_BASE64}` - PNG of state  $s_t$  encoded as a data URI.
- `type`  $\in$  {click, type, scroll, key}.
- `target` - absolute "x", "y" coordinates or a UI-tree element ID.
- `text` - non-empty only when `type == "type"`.
- `t`: The current step number within the episode.

### A.2 Illustrative Input for Reward Model Critique

The Principle-Constrained Reward Modeling (PCRM) framework evaluates the agent's behavior at each step. A core part of this is the LDP component, which uses a generative reward model ( $r_c$ ) to critique the agent's reasoning ( $c_t$ ) and action ( $a_t$ ) against the defined principle set ( $\mathcal{P}$ ). To perform this critique, the reward model  $r_c$  receives a structured input containing all relevant information for the current step. While the paper does not specify the exact prompt tokenization provided to  $r_c$ , we can construct an illustrative example (Listing 2) of the information it would process, designed to enable a comprehensive evaluation.

#### Illustrative Input for Critique Model.

1. **The High-Level Task Context:** This provides the overall goal the agent is trying to achieve, en-

sure the critique model understands the broader objective.

2. **The Current GUI State ( $s_t$ ):** The visual screenshot of the interface at time  $t$ , allowing the critique model to ground the agent’s reasoning and action in the actual UI.

3. **The Agent’s Behavior at Step  $t$ :** This is a crucial part, encompassing:

- **Agent’s Reasoning ( $c_t$ ):** The chain-of-thought output from the agent, including any self-labeled sub-goals ([MILESTONE] tokens).
- **Agent’s Predicted Action ( $a_t$ ):** The GUI action (e.g., click, type) in JSON format that the agent proposes to take.

4. **Applicable Principles for Evaluation:** A relevant subset of the guiding principles (P), both human-defined (EVP-related) and LLM-derived (LDP-related), against which the agent’s behavior is to be judged.

5. **The Reward Model’s Task Instructions:** Specific instructions to the critique model on how to evaluate the agent’s behavior against the principles and what format the output critique and score should take.

### Explanation of Key Elements.

1. **Task Context and GUI State:** These elements provide the necessary grounding. The critique model needs to understand what the agent is *supposed* to do and what the visual environment *looks like* to assess the appropriateness of the agent’s reasoning and action.

2. **Agent’s Reasoning ( $c_t$ ):** This is the primary input for LDP evaluation. The critique model assesses:

- **Coherence:** Does the reasoning logically lead to the proposed action?
- **Completeness:** Has the agent considered all necessary preconditions or safety checks (e.g., as per LDP\_Completeness)?
- **Efficiency:** Is the agent avoiding redundant steps (e.g., as per LDP\_Efficiency)?
- **Adherence to Sub-goals:** If [MILESTONE] tokens are used, does the reasoning reflect progress towards them?

3. **Agent’s Action ( $a_t$ ):** While EVPs directly check action verifiability, the LDP critique also considers the action in light of the reasoning. For instance, a correctly formatted action might still be flagged by LDP if the reasoning leading to it was flawed or violated a strategic principle.

4. **Applicable Principles:** Supplying the specific principles focuses the critique model’s evaluation. This allows for targeted feedback related to safety, UI state consistency, coherence, completeness, and efficiency.

5. **Reward Model Task Instructions:** These guide the critique model to produce a structured output, typically a numerical score (e.g., 1-10 for LDPs) and a textual critique explaining the score, referencing specific principles that were met or violated. This output then contributes to the  $r^{\text{critique}}$  component of the total reward.

### A.3 Hyperparameter Setting

We configure the hyperparameters as listed in Table 6 and train the base model using 32 NVIDIA A100 GPUs. The Orcust model is instantiated using the Qwen2.5-VL backbone. Our training regimen comprises two main phases: an initial Supervised Fine-Tuning (SFT) phase and a subsequent Reinforcement Fine-Tuning (RFT) phase.

For the Initial SFT Phase, the model is fine-tuned for 1-3 epochs on a curated dataset of approximately 1K-5K high-quality GUI interaction trajectories. We employ the AdamW optimizer with a learning rate ranging from  $2e-5$  to  $5e-6$  and a batch size of 16-32. The maximum prompt length is set between 1024 and 2048 tokens.

In the RFT Phase, policy refinement is conducted using PCRM. The Online VM-Grounded Trajectory Construction (OVTC) module continually generates approximately 15K interaction trajectories. The learning rate for RFT is set between  $1e-6$  and  $5e-7$ . The generative reward model ( $r_{\text{critique}}$ ) component of PCRM utilizes a Qwen2.5-72B. During RFT, the training batch size is 128-256. To manage policy divergence, a KL divergence constraint between 0.01 and 0.05 is applied, typical for GRPO-style algorithms. For online trajectory generation within OVTC, the self-labeled milestone tokens are assigned illustrative reward/penalty values of  $+0.5/-0.5$ .



### Listing 1: The training prompt for the Orcust

```
### SYSTEM
You are a GUI agent.
First write your reasoning between <think> ... </think>.
Then output ONE GUI action inside <answer> ... </answer> in JSON.

### Task
Export the active chart as a PDF file and close the dialog.

<principles>
P1 Cursor must remain inside the visible viewport.
P2 Destructive operations require explicit confirmation.
P3 Maintain the action sequence specified by the menu hierarchy.
</principles>

<screen>
![screenshot]({SCREENSHOT_BASE64}) @ step {t}
</screen>

<think>
[MILESTONE: MenuOpened]
I have opened **File → Export** (P3 satisfied).
Next, according to P2 I must choose "PDF" before confirming.
Cursor (412, 265) is within bounds (P1).
</think>

<answer>
{"type": "click", "target": {"x": 412, "y": 265}, "text": ""}
</answer>
```

```
### SYSTEM
You are Orcust, a vision-language GUI agent.
You must follow the **Principles** below at every step.
After thinking, output a JSON dict with keys "thought" and "action".

### TASK
Create a new folder named `Work` on the desktop.

<principles>
1. Act only on visible interactive elements.
2. Decompose long tasks into explicit sub-goals.
3. Mark a **MILESTONE** token immediately after a sub-goal is really achieved.
4. Never guess-verify on screen before proceeding.
5. Minimise unnecessary clicks / keystrokes.
</principles>

<screen>
![screenshot]({SCREENSHOT_BASE64}) @ step {t}
</screen>

FORMAT RULES:
- Enclose chain-of-thought in **<think> ... </think>**.
- End with **<answer> ... </answer>**. containing a single JSON action,
  e.g. {"type": "click", "x": 512, "y": 164} or {"type": "type", "text": "Work"}.

<think>
Goal: create folder "Demo".
Step1: Right-click empty area → [MILESTONE: sub-menu open]
Step2: Select "New → Folder" → [MILESTONE: folder shell]
Step3: Type name and press Enter → [MILESTONE: folder created]
</think>

<answer>
{"type": "click", "x": 732, "y": 645}
</answer>
```

Listing 2: The critique template for the generative reward model

```

### Task Context
Instruction: "Delete the draft proposal document, ensuring no unsaved changes are
lost."
Current Screen State ( $s_t$ ): [Image data or rich representation of the GUI where a '
Delete' button is visible, and an 'Unsaved Changes' warning is NOT visible]

### Agent's Behavior at Step t
Agent's Reasoning ( $c_t$ ):
<think>
The task is to delete 'draft_proposal.doc'.
First, I need to ensure there are no unsaved changes. The UI does not show any '
unsaved changes' indicator.
Next, I need to confirm the deletion. I will click the 'Delete' button, then look
for a confirmation prompt.
[MILESTONE: Pre-deletionChecksComplete]
The 'Delete' button is clearly visible.
</think>

Agent's Action ( $a_t$ ):
<answer>
{"type": "click", "target": "[coordinates_of_delete_button]", "text": ""}
</answer>

### Applicable Principles for Evaluation (Subset of P)
Explicit Domain Principles (Human-defined):
1. EVP_SafetyGuard: "A deletion action must be confirmed via a prompt if one appears
. If no prompt appears for a minor deletion, it's acceptable."
2. EVP_UIState: "Action should lead to an expected UI state change (e.g.,
confirmation dialog appears, or item is removed)."
```

Implicit Learned Principles (LLM-derived):

```

3. LDP_Coherence: "Reasoning should logically lead to the action and cite relevant
UI cues."
4. LDP_Completeness: "Ensure all pre-conditions mentioned in the task or implied by
safety are checked in reasoning before critical actions." (Relates to Goal-
aligned planning)
5. LDP_Efficiency: "Avoid redundant steps if a direct action is possible."

### Reward Model Task:
Based on the agent's reasoning ( $c_t$ ) and action ( $a_t$ ) in the given screen state ( $s_t$ 
), evaluate compliance with the provided principles.
- For LDPs: Provide a numerical score (e.g., 1-10) and a brief textual critique.
- For EVPs: Determine if the action itself is verifiable against these rules (e.g.,
True/False, or a specific reward value).
```

Table 6: Orcust: Hyperparameter Configuration

Initial SFT Phase		RFT Phase	
Hyperparameter	Value	Hyperparameter	Value
SFT Dataset Size	~1K-5K trajectories	GRM ('r_critique')	Qwen2.5-72B
Learning Rate	2e-5 to 5e-6	Trajectory Generation	15K (continuous)
Batch Size	16 - 32	Learning Rate	1e-6 to 5e-7
Number of Epochs	1 - 3	KL Divergence Constraint	0.01 - 0.05
Max Prompt Length	1024 - 2048	Batch Size	128 - 256
Optimizer	AdamW	Milestone Token Reward	-0.5 / +0.5

Table 7: Illustrative principle–reward channels employed by PCRM. “EVP” denotes *Environment-Verifiable Principles*; “LDP” denotes *LLM-Derived Principles*. Typical ranges refer to per–step reward contributions before weighting.

Category	Reward Type	Formal Check & Rationale	Typical Range
<b>EVP</b>	Action–type correctness	$r = 1$ if $\hat{a}_t.type = a_t^*$ , else 0; enforces the correct primitive operation.	$\{0, 1\}$
	Target–bounding compliance	$r = 1 - \delta$ , where $\delta$ is the normalised Manhattan distance between predicted and gold coordinates (clipped at 0); rewards spatial precision.	$[0, 1]$
	UI–state transition success	Binary flag from the VM harness when post–action DOM/screenshot matches an expected template (e.g. <i>Settings</i> page visible).	$\{0, 1\}$
	Output–format validity	$r = -1$ if the agent’s output violates JSON/XML/span tags; prevents malformed actions.	$\{-1, 0\}$
	Safety guard	Immediate $r = -1$ if a destructive command is issued without prior confirmation; hard-coded safety constraint.	$\{-1, 0\}$
<b>LDP</b>	Coherent chain-of-thought	Generative reward model $f_{GRM}$ scores whether $c_t$ sets sub-goals, cites UI cues, avoids hallucinations.	0–10
	Goal-aligned planning	Positive score when $c_t$ explicitly references the top-level instruction and maintains it through the episode.	0–5
	Efficiency heuristic	$-0.1$ for each redundant navigation; $+0.5$ bonus for finishing in $\leq \tau^*$ steps; encourages brevity.	$\approx -1 \dots +1$
	User-experience etiquette	$+1$ if a risky action is preceded by confirmation in $c_t$ ; $-1$ if blocking pop-ups remain uncleared.	$\{-1, 0, +1\}$
	Cross-widget consistency	$+1$ when visual style (e.g. title-case) is preserved across form fields, judged by LLM critique.	$\{0, 1\}$

*Note.* Scaling factors  $(w_i, v_j)$  are annealed so that EVP feedback dominates early training while LDP feedback becomes prominent after roughly 20k steps.

#### A.4 Illustrative Principle-Reward Channels

Table 7 below details some of the illustrative principle-reward channels employed by PCRM. It outlines the category of principle (EVP or LDP), the specific aspect of agent behavior being rewarded (Reward Type), the formal check or rationale behind the reward, and the typical range of the reward contribution per step (before weighting). This dual-source reward mechanism guides the agent to produce actions that are not only functionally correct but also coherent, efficient, and aligned with human preferences and task constraints.

The Orcust framework’s PCRM is central to its ability to learn robust GUI interaction policies. PCRM provides stepwise and interpretable rewards by evaluating the agent’s actions and reasoning

against a set of predefined principles. These principles are categorized into **EVP**, which are deterministic checks against the GUI environment, and **LDP**, which leverage an LLM to critique the agent’s reasoning and adherence to more nuanced, high-level strategies.

#### A.5 Dataset Distribution

Table 8 presents statistical information on 15,000 high-quality multi-step GUI interaction trajectories autonomously generated via OVTC, covering various task types across different platforms. The “average steps per trajectory” refers to the number of self-labeled sub-goals or critical interaction points within each trajectory. Moreover, these trajectories consist of sequences of fundamental atomic GUI actions, with specific types detailed in Table 9.

Table 8: Illustrative Task Type Distribution and Complexity in 15K OVTC Trajectories.

Platform	Task Category / Example	Count	Trajectory Avg Steps
<b>Desktop Applications</b>	Document Editing & Export (e.g., LibreOffice, Text Editors - save, export to PDF, change font)	2,500	5 – 8
	File System Operations (e.g., create folder, rename file, copy/paste)	1,500	3 – 5
	Email Client Usage (e.g., compose email, send, check inbox, delete email)	1,000	6 – 10
	Application Settings Configuration (e.g., changing preferences in a software)	500	4 – 7
<b>Web Applications</b>	Online Form Filling (e.g., registration, login, contact forms)	2,000	5 – 9
	E-commerce Tasks (e.g., search product, add to cart, initiate checkout)	1,500	6 – 12
	Web Navigation & Information Search (e.g., navigating menus, finding specific info)	2,000	4 – 8
	Social Media Interactions (e.g., create post, browse feed)	500	4 – 7
<b>Mobile Applications</b>	System Settings Adjustment (e.g., toggle Wi-Fi, change display brightness)	1,000	2 – 4
	Contact List Management (e.g., add contact, edit contact)	500	4 – 6
	Messaging App Usage (e.g., send message, read message)	1,000	3 – 6
	General App Navigation & Task Completion (e.g., launching app, using core features)	1,000	5 – 10
<b>Total</b>		<b>15,000</b>	

Table 9: Illustrative Fundamental Atomic Actions in OVTC Trajectories.

Atomic Action Type	Description
click	Clicking on a UI element (button, link, menu item, coordinates).
type / text_entry	Entering text into a text field, search bar, or form input.
scroll	Scrolling the view (vertically or horizontally) to reveal more content.
key_press	Simulating a keyboard key press (e.g., Enter, Esc, Tab, specific characters not part of a larger text entry).
swipe	Swiping gestures, primarily for mobile interfaces (e.g., swipe left/right/up/down).
drag	Clicking and holding an element, then moving it to another location (e.g., drag-and-drop).
select	Choosing an option from a dropdown menu or list box.
long_press	Pressing and holding an element, typically to open a context menu (mobile).