

Fully Decoupled Reinforcement Learning with Adaptive Data Curation for GUI Agents

Pengxiang Li^{1,2*}, Zechen Hu^{3*}, Zirui Shang^{1,2*}, Jingrong Wu^{3*},
Yang Liu², Hui Liu³, Zhi Gao^{1,2,5}✉, Chenrui Shi^{1,2}, Bofei Zhang²,
Zihao Zhang³, Xiaochuan Shi³, Zedong Yu^{2,4}, Yuwei Wu^{1,5}, Xinxiao Wu^{1,5},
Yunde Jia⁵✉, Liuyu Xiang⁴, Zhaofeng He⁴, Qing Li²✉

¹Beijing Key Laboratory of Intelligent Information Technology,
School of Computer Science & Technology, Beijing Institute of Technology,
²State Key Laboratory of General Artificial Intelligence, BIGAI, ³DataCanvas,

⁴Beijing University of Posts and Telecommunications,

⁵Guangdong Laboratory of Machine Perception and Intelligent Computing,
Shenzhen MSU-BIT University

* Equal Contribution. ✉ Correspondence: gaozhibit@bit.edu.cn, jiayunde@smbu.edu.cn, dylan.liqing@gmail.com.

Abstract

Vision-Language Models (VLMs) have catalyzed the development of Graphical User Interface (GUI) agents for desktop automation. However, reinforcement learning (RL) for GUI agents remains bottlenecked by two challenges: existing RL frameworks still leave substantial idle time inside the rollout stage, and the learning signal from long-horizon, sparse-reward interactions is highly imbalanced. In this paper, we introduce **DART**, a **fully Decoupled Agentic RL Training** framework for GUI agents. Whereas prior RL designs only separate the trainer from the rollout, DART splits the pipeline into four independent modules: environment cluster, rollout service, data manager, and trainer, which brings two extra efficiency benefits: rollout-wise trajectory scheduling and per-worker model synchronization. This yields a $1.6\times$ increase in GPU utilization, a $1.9\times$ boost in training throughput, and a $5.5\times$ increase in environment utilization over a non-decoupled baseline. We further propose a multi-level adaptive data curation scheme operating at the task, trajectory, step, and token levels: it reallocates rollout budget by task difficulty, bootstraps hard tasks with a pre-collected experience pool, restricts policy updates to high-entropy critical steps, and applies truncated importance sampling to absorb the distribution gap between the rollout and training engines. On the OSWorld benchmark, our *DART-GUI-7B* achieves a 43.28% task success rate, a 15.76% absolute improvement over its base model and a new state-of-the-art among open-source 7B-scale agents. *Code, data, and model checkpoints are open-sourced at <https://computer-use-agents.github.io/dart-gui/>.*

1 Introduction

The rapid advancement of multimodal foundation models, particularly large language models (LLMs) (OpenAI, 2025b; DeepSeek-AI, 2025) and vision-language models (VLMs) (Bai et al., 2025; Wang et al., 2024; Shen et al., 2025), has catalyzed the development of autonomous agents capable of perceiving and interacting with graphical user interfaces (GUIs) (Liu et al., 2024; Hong et al., 2024). Operating directly on raw visual observations and natural language instructions, these GUI agents (OpenAI, 2025a; Bai et al., 2025; Guo et al., 2025) demonstrate immense potential for automating workflows across both desktop and mobile environments. While reinforcement learning (RL) has proven effective for enhancing the reasoning and exploration capabilities of LLMs/VLMs in various domains (Zhang et al., 2025; Wang et al., 2025c; Ye et al., 2025), its application to GUI agents (especially desktop agents) remains particularly challenging. GUI tasks typically involve long-horizon, multi-turn interactions that maintain multimodal context across dozens of states and actions, making RL training processes prohibitively inefficient.

While recent efforts have attempted to adapt RL for GUI agents (Lu et al., 2025; Yang et al., 2025), their performance remains modest, lagging behind closed-source, proprietary systems (Anthropic, 2025b; OpenAI, 2025a; Wang et al., 2025a). We identify two primary bottlenecks. First, existing RL frameworks leave substantial idle time inside the rollout stage. Recent asynchronous designs (Fu et al., 2025b; Cao et al., 2025; Wang et al., 2025c) decouple the trainer from rollout, but rollout itself remains synchronous in two ways: environments

in a batch wait for the slowest trajectory before the next round, and all workers pause whenever weights are refreshed. This is less severe in shorter or more uniform interaction settings, such as many mobile GUI or text-based tool-use tasks. Desktop GUI tasks, however, span dozens of steps and tens of minutes per trajectory with high variance, so each barrier and each weight update leaves both GPUs and environments idle for substantial periods. Second, current RL frameworks extract a highly imbalanced learning signal from desktop GUI interactions at both the task and step levels. Across tasks, easy ones consume most of the rollout budget while hard ones rarely yield a successful trajectory. Within a trajectory, only a few steps determine the outcome, while the rest have little effect on it. Standard policy updates weight all steps equally, so the signal on these few decisive steps is diluted by the many unimportant ones. Such imbalances at the task and step levels destabilize training and slow convergence.

In this paper, we address these bottlenecks by introducing **DART**, a fully Decoupled Agentic RL Training framework that decouples the RL process into four independent, asynchronous modules, namely the environment cluster, rollout service, data manager, and trainer. This decoupled design lets every module run in parallel, so training and rollout no longer block each other. It addresses the idle periods within the rollout through two schemes. With *rollout-wise scheduling*, each environment moves on to its next trajectory the moment it finishes the current one, so it no longer waits for the slowest peer in the batch. With *per-worker model synchronization*, rollout workers are refreshed one at a time when new weights arrive, so the rollout service keeps serving requests throughout the update. Together they deliver $1.6\times$ higher GPU utilization, $1.9\times$ higher training throughput, and $5.5\times$ higher environment utilization over a non-decoupled baseline.

We further introduce a multi-level adaptive data curation scheme that rebalances and stabilizes learning across the task, trajectory, step, and token levels. At the task and trajectory levels, we dynamically adjust each task’s number of rollouts and maximum trajectory length according to its real-time success rate, and maintain an experience pool of pre-collected successful trajectories to bootstrap hard tasks that yield no positive samples online. At the step level, we restrict policy updates to high-entropy steps, which represent the critical decision

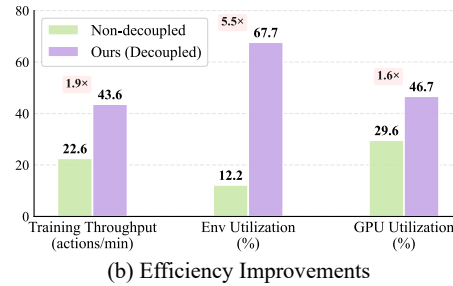
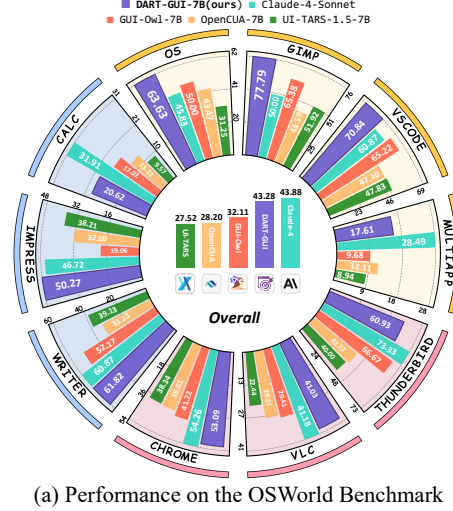


Figure 1: Overview of the performance and efficiency of Decoupled Agentic RL Training (**DART**) framework.

points within long trajectories, allowing us to concentrate the gradient signal where it matters most. At the token level, a truncated importance sampling weight absorbs the distribution gap between the rollout and training engines, keeping updates stable under asynchronous execution.

We evaluate our approach by training DART-GUI-7B, initialized from the UI-TARS-1.5-7B (Qin et al., 2025). On the challenging OSWorld benchmark, our framework provides steady performance improvements, achieving a 43.28% task success rate. This represents a 15.76% absolute gain over the base model. More importantly, this success stems from robust policy learning rather than memorizing the training environments. DART-GUI-7B shows strong zero-shot generalization on 171 unseen human-annotated tasks on the Ubuntu OS, beating the baseline by 8.87% (36.06% vs. 27.19%). It also transfers well to completely different software interfaces in the UI-Vision benchmark (Nayak et al., 2025).

Our main contributions are as follows: (1) We propose DART, a novel decoupled RL framework that significantly enhances training efficiency for GUI agents, achieving $1.6\times$ higher GPU utilization

for rollout, $5.5\times$ better environment utilization, and $1.9\times$ higher training throughput. (2) We introduce an adaptive data curation scheme that optimizes learning at the task, trajectory, step, and token levels, leading to more effective policy updates. (3) We introduce DART-GUI-7B, an open-source GUI agent model that achieves state-of-the-art performance on OSWorld among models at the 7B scale. To promote reproducibility and advance research in the agentic RL community, we *have fully open-sourced our code, model checkpoints, and data* at <https://computer-use-agents.github.io/dart-gui/>.

2 Related Work

GUI Agents The training paradigms for GUI agents have shifted from supervised fine-tuning (SFT) (Lin et al., 2024; Qin et al., 2025; Wang et al., 2025e), which is often constrained by the fixed patterns of demonstration data, to reinforcement learning approaches that learn from environmental feedback. Early RL methods like GUI-R1 (Luo et al., 2025) and InfiGUI-R1 (Liu et al., 2025) adopted offline training without real-time interaction, struggling with distribution shift and multi-turn reasoning. Recent online RL frameworks address these limitations through various strategies: ARPO (Lu et al., 2025) extends GRPO for multi-turn interactions, ZeroGUI (Yang et al., 2025) automates task and reward generation via VLMs, while ComputerRL (Lai et al., 2025) designs asynchronous architectures for training API-equipped GUI Agents. Similarly, DistRL (Wang et al., 2025c) also utilizes an asynchronous framework, specifically designed for mobile agents with off-policy learning. In contrast, our framework targets the challenges of long-horizon, multimodal GUI interactions, where high-latency desktop environments and sparse rewards demand both finer-grained module decoupling and difficulty-aware data curation.

RL for LLMs, VLMs and Agents Reinforcement learning has emerged as a powerful paradigm for enhancing the reasoning and decision-making capabilities of large language models, evolving from preference-based to outcome-based training approaches. The computational demands of RL have driven the development of asynchronous architectures that decouple different training components for improved efficiency. Building on successful applications in game AI (Vinyals et al., 2019; Berner et al., 2019), recent frameworks have

adapted asynchronous training for language models: AREAL (Fu et al., 2025b) separates rollout generation from model training to maximize GPU utilization while employing staleness-aware PPO to maintain training stability, ROLL (Wang et al., 2025d) provides a comprehensive RL library supporting multi-model pipelines and flexible resource scheduling for large-scale LLM training. Building on this line of work, our framework extends asynchronous RL to the visually-grounded, sparse-reward regime of desktop GUI agents through rollout-wise scheduling and per-worker model synchronization.

Data Curation and Trajectory Optimization for GUI Agents

Beyond architectural designs, recent works have highlighted the importance of data selection and trajectory optimization for training GUI agents. OS-Genesis (Sun et al., 2025) constructs interaction trajectories through reverse task synthesis and reward-based filtering. Other studies explore step-wise Q-estimation (Putta et al., 2024), multi-stage data selection (Lu et al., 2026), and dynamic context sampling (Zhou et al., 2026). Related research has also investigated trajectory-based agent tuning, feedback refinement, and step-wise preference learning (Gao et al., 2025; Li et al., 2025b, 2024). In particular, TongUI (Zhang et al., 2026) collects internet-scale trajectories from multimodal web tutorials to improve the generalization of GUI agents. GUIDE (Xie et al., 2026) further addresses domain bias through real-time web-video retrieval and plug-and-play annotation. Our multi-level adaptive data-curation scheme complements these efforts by addressing task difficulty imbalance and step-level redundancy in long-horizon desktop interactions.

3 DART: Decoupled Agentic RL Training Framework

3.1 Formulation

We formulate GUI tasks as a sequential decision-making process. At time step t , given the current visual state s_t (a screenshot of GUI) and the interaction history $h_t = \{(s_{\max(1,t-m)}, r_{\max(1,t-m)}, a_{\max(1,t-m)}), \dots, (s_{t-1}, r_{t-1}, a_{t-1})\}$ of previous m steps, where r denotes the reasoning thought and a denotes the action (such as clicking on a specific UI element or entering text), along with the task τ , the agent generates a new thought r_t and an executable action a_t . Executing the action

a_t leads to a new visual state s_{t+1} (an updated screenshot). This interaction loop continues, with the agent repeatedly observing the environment, producing thoughts and actions, and receiving updated observations until either a termination condition is met (e.g., the task is completed or fails) or the maximum number of steps is reached. We parameterize the GUI agent using a policy model π_θ (i.e., a VLM) that generates thoughts and actions based on the current state and historical context: $r_t^*, a_t^* = \arg \max_{r_t, a_t} \pi_\theta(r_t, a_t | \tau, h_t, s_t)$, where r_t^* and a_t^* represent the optimal thought and action selected by the policy model.

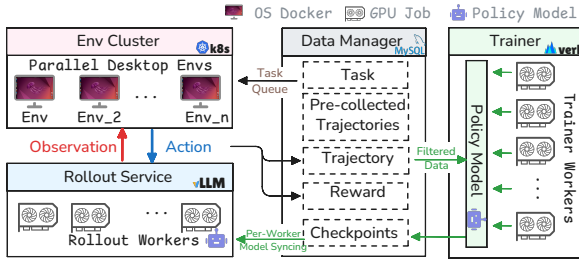


Figure 2: Overall architecture of our framework. The *Rollout Service* interacts with multiple environments in parallel to generate trajectories, which are managed and delivered to the *Trainer* for policy updates. Updated actors are synchronized back to the *Rollout Service*, enabling scalable and efficient asynchronous learning.

3.2 Architecture

Our RL framework contains four decoupled modules: *Trainer*, *Data Manager*, *Env Cluster*, and *Rollout Service*, forming a fully non-blocking pipeline to maximise training throughput, as shown in Figure 2. We set up hundreds of real desktop environments in *Env Cluster*, and design a *Rollout Service* to load multiple policy models. The *Env Cluster* receives a sequence of tasks from the *Data Manager*, and samples N trajectories for each task. The *Rollout Service* dynamically assigns idle workers to produce thoughts and actions for different rollouts in parallel. We store sampled trajectories and corresponding rewards in the *Data Manager*. When N trajectories of one task are finished, the *Data Manager* filters them according to predefined rules, and passes the selected trajectories to the *Trainer* for policy updates. Finally, the updated policy model is synchronized back to the *Rollout Service*, enabling scalable and highly efficient asynchronous learning. Key interactions among these modules are as follows.

3.3 Asynchronous Trainer

To improve GPU and environment utilization, we decouple the *Trainer* from the trajectory rollout process, which avoids the blocking between training and rollout. The *Trainer* operates asynchronously, receiving filtered trajectories from the *Data Manager* and performing step-wise GRPO updates. The updated model weights are synchronized to the *Rollout Service*, enabling continuous training while new trajectories are sampled simultaneously.

Step-wise GRPO. We adopt step-wise Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to train our GUI agent. For each task τ , we sample N trajectories $T_1, T_2, \dots, T_N$ using the current policy $\pi_{\theta_{\text{old}}}^{\text{Rollout}}$ of the *Rollout Service*, where the i -th trajectory has length L_i and consists of state-thought-action triplets: $T_i = \{(s_{i,j}, r_{i,j}, a_{i,j})\}_{j=1}^{L_i}$. Each trajectory receives a reward R_i , and we decompose each trajectory into individual steps and group all steps from the same task for advantage computation. Specifically, we create a step group $\mathcal{D} = \{(h_{i,j}, s_{i,j}, r_{i,j}, a_{i,j}, R_i) \mid i \in [1, N], j \in [1, L_i]\}$, where each step $(s_{i,j}, r_{i,j}, a_{i,j})$ is combined with its history $h_{i,j}$ and the trajectory-level reward R_i . We denote “ $h_{i,j}, s_{i,j}, r_{i,j}, a_{i,j}$ ” as “ h, s, r, a ” for simplicity. The step-wise GRPO objective is formulated as

$$\begin{aligned} \mathcal{J}(\theta) = \mathbb{E}_{(h,s,r,a,R) \sim \mathcal{D}} \Big[& \min(\rho_\theta(r, a|h, s) A, \\ & \text{clip}(\rho_\theta(r, a|h, s), 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}) A) \\ & - \beta D_{\text{KL}}(\pi_\theta^{\text{Train}}(\cdot|h, s) \parallel \pi^{\text{Ref}}(\cdot|h, s)) \Big], \end{aligned} \quad (1)$$

where $\rho_\theta(r, a|h, s) = \frac{\pi_\theta^{\text{Train}}(r, a|h, s)}{\pi_{\theta_{\text{old}}}^{\text{Train}}(r, a|h, s)}$ is the policy ratio, and $A_{i,j} = (R_i - \mu_R) / \sigma_R$ denotes the group-normalized advantage, with μ_R and σ_R being the mean and standard deviation of $\{R_i\}_{i=1}^N$ within group $\mathcal{D}$.

3.4 Rollout-wise Sampling

Solving practical GUI tasks (such as OSWorld) typically involves dozens of steps and lasts for tens of minutes, making sampling efficiency a major bottleneck. Tasks within the same batch vary significantly in difficulty, and even repeated executions of the same task may produce trajectories of vastly different lengths (Wang et al., 2025e). Under conventional batch-wise sampling (Figure 3(a)), this heterogeneity causes substantial resource underutilization: environments that complete early remain idle while GPUs stay underused until all

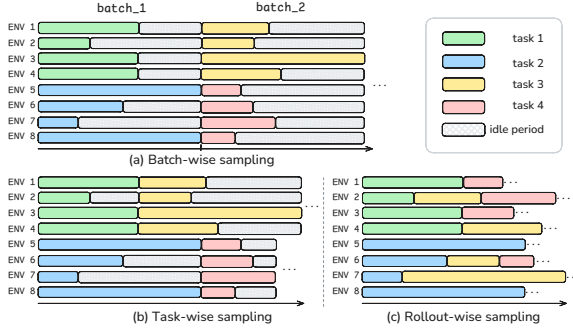


Figure 3: Visualization of sampling timelines for 4 tasks in two batches (denoted by different colors) with rollout number $N = 4$, batch size = 2, and a total of 8 environments. Each bar represents the timeline of rollout execution of a task on one environment.

trajectories in the batch finish. Task-wise sampling (Figure 3(b)) partially addresses this but still waits for entire tasks to complete before resuming. We instead implement rollout-wise sampling (Figure 3(c)), where an individual trajectory serves as the minimal scheduling unit: once an environment completes a rollout, it immediately launches the next sampling request without waiting for others, significantly improving environment utilization and GPU throughput.

Furthermore, rather than statically assigning environments to fixed rollout workers, our decoupled training framework introduces a dynamic model service pool for load balancing. All GPUs are pooled into the shared *Rollout Service*, with incoming requests being distributed to workers based on current device utilization. This design provides two engineering advantages: (1) all environments communicate with the *Rollout Service* through a unified interface, simplifying system architecture and coordination, and (2) balanced GPU workloads minimize idle time and bottlenecks, resulting in faster inference and higher overall throughput.

3.5 Per-Worker Model Synchronization

Model synchronization presents a critical bottleneck in asynchronous GUI agent RL. Traditional approaches rely on global synchronization: when the *Trainer* completes a training iteration, all rollout workers halt operations and wait for every GPU worker to receive the updated weights before resuming sampling. As shown in Figure 4 (a), this creates system-wide downtime where environments sit idle, and GPUs remain underutilized during each update.

We introduce per-worker model updates to elim-

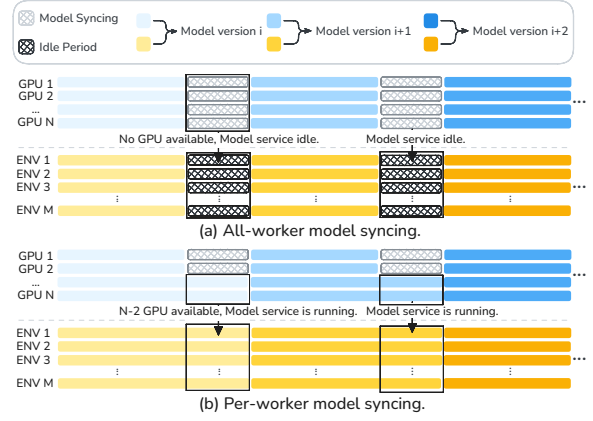


Figure 4: Timeline comparison between all-worker and per-worker model updating with an illustrative example using N GPUs and M environments. The timelines depict idle periods for the GPUs and environments across two model updates, with different model versions represented by varying shades of color. Each worker is served by 2 GPUs.

inate this bottleneck through staggered parameter distribution. Instead of synchronized model updates across all rollout workers simultaneously, we gradually refresh model weights on rollout workers. That is, when one worker is updating model weights, the others continue serving inference requests with their current model version. Figure 4 (b) illustrates how this maintains continuous service availability: environments never experience complete blocking. This approach delivers two key advantages: dramatically improved sampling throughput through reduced idle time, and seamless model version transitions that preserve ongoing rollout stability.

4 Multi-Level Adaptive Data Curation for GUI Tasks

4.1 Performance-Aware Task Rollout

Fixed sampling strategies in reinforcement learning waste computational resources by treating all tasks equally. Easy tasks receive excessive sampling while difficult tasks lack sufficient exploration. We propose a performance-aware task rollout strategy that dynamically adjusts both sampling frequency and trajectory length based on each task’s learning progress.

Dynamic Rollout Number. We continuously monitor each task’s success rate and adjust its sampling frequency accordingly. When a task achieves high success rates (above 0.6), we reduce its rollout frequency from 8 to lower values, preventing over-

fitting on already-solved tasks. Tasks with low success rates maintain maximum sampling to ensure adequate learning opportunities. This strategy real-locates computational resources from well-learned tasks to challenging ones, improving overall training efficiency.

Dynamic Trajectory Length. We maintain a per-task budget that adapts to how many steps each task actually needs. For each task τ , let $\mathcal{S}_\tau$ denote the set of successful trajectories observed so far. We set the per-task budget as $L_\tau = \min(L_{\max}, \max_{T \in \mathcal{S}_\tau} |T| + \delta)$, where $L_{\max}$ is a global upper bound and δ is a small exploration buffer. Tasks with $\mathcal{S}_\tau = \emptyset$ default to $L_{\max}$. The budget is refreshed at the end of each epoch, and any rollout exceeding L_τ is truncated and recorded as a failure. Consequently, simple tasks such as a single click terminate within a few steps, while complex multi-application workflows are allowed the dozens of steps they historically need.

4.2 High-Entropy-Driven Step Optimization

Inspired by Wang et al. (2025b), who showed that training exclusively on high-entropy tokens can drive effective RL, we extend this insight to multi-turn GUI agent training. Since low-entropy steps typically correspond to non-critical decisions and destabilize training, we filter out the bottom 20% lowest-entropy steps within each sample group and update the policy only on the remaining 80%.

For the t -th step in a trajectory, let $x = [r_t; a_t]$ denote the concatenated sequence of thoughts and actions. We calculate the step-level entropy H_t as the average entropy across all generated tokens: $H_t = \frac{1}{|r_t| + |a_t|} \sum_{i=1}^{|r_t| + |a_t|} H_{t,i}$, where $H_{t,i} = -\sum_{v=1}^V p_{t,i,v} \log p_{t,i,v}$ represents the token-level entropy, and $p_{t,i,v} = \pi_\theta(v|\tau, h_t, s_t, x_{<i})$ is the probability of generating token v given the prefix. During training, we compute the gradient updates exclusively for steps whose entropy H_t is greater than or equal to the 20th percentile of step entropies within the group ($H_t \geq P_{20}$).

4.3 Distribution Alignment for Decoupled Training

During training, the *Rollout Service* typically employs low-precision quantization (e.g., FP8) to maximize sampling throughput, whereas the *Trainer* operates in higher precision (e.g., BF16) to ensure gradient stability. This precision discrepancy introduces mismatches between their respective policy

distributions for the exact same model checkpoint. In addition, the pre-collected offline trajectories also exhibit a distinct distribution from the current rollout policy. To mitigate this distribution gap, we adopt the truncated importance sampling technique from Yao et al. (2025) and incorporate a weight $w(r, a|h, s) = \min\left(\frac{\pi_{\theta_{\text{old}}}^{\text{Train}}(r, a|h, s)}{\pi_{\theta_{\text{old}}}^{\text{Rollout}}(r, a|h, s)}, C\right)$ into the training objective. By weighting each sample by the importance ratio between the trainer and rollout policies, w approximates an expectation under the trainer’s distribution from samples drawn under the rollout’s distribution, thereby correcting the off-policy bias and stabilizing training. The final objective is defined as:

$$\begin{aligned} \mathcal{J}_{\text{HE}}(\theta) = & \mathbb{E}_{(h,s,r,a,R) \sim \mathcal{D}} \left[\mathbb{I}[H_t \geq \tau_D^{0.2}] \cdot w(r, a|h, s) \cdot \right. \\ & \min(\rho_\theta(r, a|h, s) A, \\ & \quad \left. \text{clip}(\rho_\theta(r, a|h, s), 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}) A) \right. \\ & \left. - \beta D_{\text{KL}}(\pi_\theta^{\text{Train}}(\cdot|h, s) \parallel \pi^{\text{Ref}}(\cdot|h, s)) \right], \end{aligned} \quad (2)$$

where $\mathbb{I}[\cdot]$ is the indicator function, $\tau_D^{0.2}$ denotes the 20th percentile of step entropies within group $\mathcal{D}$.

5 Experiment

5.1 Settings

Experimental Setup We evaluate our approach on OSWorld-Verified (Xie et al., 2024), a comprehensive benchmark for assessing multimodal autonomous agents in realistic computer environments. For our training corpus, we adopt the sampling methodology proposed by Lu et al. (2025), selecting a representative subset of 203 tasks from the OSWorld benchmark. We use the evaluation scripts from the OSWorld for the RL reward.

Evaluation Protocol We follow the OSWorld evaluation (Xie et al., 2024), which uses execution-based validation scripts to assess task completion. Each trajectory receives a reward score in $[0, 1]$ based on programmatic verification of the final system state against predefined success criteria.

Implementation We adopt UI-TARS-1.5-7B (Qin et al., 2025) as the baseline for our policy model. Unlike approaches that rely on multi-agent systems, agent workflows, or agents equipped with additional APIs/tools, we focus on enhancing the capabilities of a single VLM agent through reinforcement learning, aiming to improve the model’s inherent decision-making abilities without external scaffolding. Based on decoupled agentic

Table 1: Results on the OSWorld benchmark. Max Steps indicates the maximum number of agent-environment interactions allowed. **Bold** values denote the best performance among *open-source* models. For brevity, LibreOffice Calc, Impress, and Writer are abbreviated as *calc*, *impress*, and *writer*, respectively. Our results are obtained through evaluation on self-deployed devices using the official codebase and Docker environment. * means results reported in the original papers.

Model	Max Steps	Task Success Rate (%)										
		chrome	gimp	calc	impress	writer	multi_apps	os	thunderbird	vlc	vs_code	Overall
Closed-source Model												
OpenAI CUA o3 (OpenAI, 2025b)	50	21.74	38.46	8.51	4.26	21.74	11.83	37.50	20.00	17.59	21.74	17.17
OpenAI CUA o3 (OpenAI, 2025b)	100	13.04	38.46	10.64	10.64	30.43	16.53	62.50	26.67	39.18	39.13	23.00
OpenAI CUA (OpenAI, 2025a)	100	33.61	48.08	18.09	24.47	23.91	15.61	54.17	56.67	38.21	60.87	30.50
OpenAI CUA (OpenAI, 2025a)	50	36.87	34.62	14.89	29.70	26.09	15.81	70.83	66.67	11.76	69.57	31.30
Claude-3-7-Sonnet (Anthropic, 2025a)	100	54.26	42.31	21.28	31.83	47.83	19.62	45.83	66.67	24.88	56.52	35.57
Claude-3-7-Sonnet (Anthropic, 2025a)	50	52.09	38.46	31.91	36.09	43.48	17.66	50.00	53.33	23.53	56.52	35.83
DeepMiner-Mano-7B (Fu et al., 2025a)	100	39.13	69.23	27.66	42.47	56.52	17.20	50.00	73.33	35.29	78.26	40.16
DeepMiner-Mano-72B (Fu et al., 2025a)	100	54.26	88.46	57.45	61.62	78.26	24.41	66.67	66.67	35.29	78.26	53.88
Seed1.5-VL-250717 (Guo et al., 2025)	100	56.52	50.00	34.78	48.91	56.52	15.35	39.13	73.33	35.29	56.52	40.18
Claude-4-Sonnet-0514 (Anthropic, 2025b)	100	47.74	50.00	29.79	42.47	52.17	27.86	45.83	66.67	32.82	69.57	41.39
UI-TARS-250705 (Wang et al., 2025a)	100	56.43	50.00	40.43	55.30	60.87	14.66	41.67	66.67	44.00	52.17	41.84
UI-TARS-2-2509 (Wang et al., 2025a)	100	62.96	50.00	65.96	56.38	60.87	34.13	41.67	73.33	49.94	73.91	53.11
Claude-4-Sonnet-0514 (Anthropic, 2025b)	50	54.26	50.00	31.91	46.72	60.87	28.49	45.83	73.33	41.18	60.87	43.88
Claude-4-Sonnet-0929 (Anthropic, 2025b)	100	62.96	53.85	72.34	68.00	82.48	49.54	70.83	60.00	58.18	73.91	62.88
Open-Source Model												
Qwen2.5-VL-32B (Bai et al., 2025)	100	8.70	3.85	0.00	0.00	8.70	2.15	8.33	6.67	0.00	8.70	3.88
Qwen2.5-VL-72B (Bai et al., 2025)	100	4.35	0.00	6.38	0.00	8.70	3.23	16.67	13.33	5.88	4.35	4.99
ZeroGUI* (Yang et al., 2025)	15	-	-	-	-	-	-	-	-	-	-	20.2
UI-TARS-72B-dpo (Qin et al., 2025)	50	33.24	61.54	12.77	25.45	43.48	6.71	33.33	33.33	23.53	47.83	25.88
OpenCUA-7B (Wang et al., 2025e)	100	36.14	47.44	10.64	31.90	27.54	9.87	42.87	40.00	29.41	44.93	26.60
UI-TARS-72B-dpo (Qin et al., 2025)	100	32.61	73.08	6.38	23.81	34.78	8.29	37.50	60.00	17.65	52.17	26.84
UI-TARS-1.5-7B (Qin et al., 2025)	50	32.79	53.85	8.51	39.31	41.30	8.60	25.54	46.67	24.41	52.17	27.25
UI-TARS-1.5-7B (Qin et al., 2025)	100	38.34	51.92	9.57	38.21	39.13	8.94	31.25	40.00	22.44	47.83	27.52
OpenCUA-7B (Wang et al., 2025e)	50	38.61	43.59	13.22	32.60	33.33	12.11	43.47	42.22	28.31	47.10	28.20
ARPO* (Lu et al., 2025)	15	-	-	-	-	-	-	-	-	-	-	29.9
GUI-Owl-7B (Ye et al., 2025)	15	41.22	65.38	17.02	19.06	52.17	9.68	50.00	66.67	29.41	65.22	32.11
GUI-Owl-7B (Ye et al., 2025)	30	41.22	73.08	12.77	25.39	60.86	10.75	50.00	66.67	47.06	69.57	34.79
OpenCUA-32B (Wang et al., 2025e)	50	43.39	69.23	13.48	38.28	40.58	14.93	53.62	53.33	25.49	55.07	34.14
OpenCUA-32B (Wang et al., 2025e)	100	39.77	66.67	18.44	37.60	36.23	16.21	55.07	46.67	33.33	63.31	34.79
Baseline												
UI-TARS-1.5-7B	30	23.91	50.00	6.38	27.45	34.77	9.54	43.48	53.33	17.65	40.91	24.17
UI-TARS-1.5-7B	100	38.34	51.92	9.57	38.21	39.13	8.94	31.25	40.00	22.44	47.83	27.52
Ours												
DART-GUI-7B	30	51.99	76.99	19.12	48.67	60.82	16.53	62.43	60.03	39.23	69.54	42.05
Δ (Ours vs. Baseline)	30	↑28.08	↑26.99	↑12.74	↑21.22	↑26.05	↑6.99	↑18.95	↑6.70	↑21.58	↑28.63	↑17.88
DART-GUI-7B	100	53.09	77.79	20.62	50.27	61.82	17.61	63.63	60.93	41.03	70.84	43.28
Δ (Ours vs. Baseline)	100	↑14.75	↑25.87	↑11.05	↑12.06	↑22.69	↑8.67	↑32.38	↑20.93	↑18.59	↑23.01	↑15.76

RL training (DART) and the data curation scheme, we obtain the DART-GUI-7B model. More details about training and RL framework can be found in Appendix C.

5.2 Main Results

Table 1 presents the results on OSWorld across ten applications. With a maximum of 30 interaction steps, DART-GUI-7B achieves an average task score of $42.05 \pm 1.50\%$, outperforming the UI-TARS-1.5-7B initial checkpoint at 24.17% by 17.88 percentage points. When the interaction budget is increased to 100 steps, DART-GUI-7B reaches $43.28 \pm 0.65\%$, compared with 27.52% for UI-TARS-1.5-7B, yielding an absolute gain of 15.76 percentage points. These results establish DART-GUI-7B as the best-performing open-source 7B-scale agent evaluated under the corresponding OSWorld protocols.

The improvements span a diverse set of applications, including web browsers, office software, system utilities, and multi-application tasks. This

broad improvement demonstrates the effectiveness of combining online RL with our fully decoupled architecture and adaptive data-curation strategies, particularly for long-horizon tasks with sparse rewards and diverse action spaces.

Table 2: Efficiency comparison between the non-decoupled baseline and DART under identical hardware and environment configurations, measured by Training Throughput ($T_{\text{throughput}}$) (action/mins), GPU Utilization (GPU Util.) (%) and Env Utilization (Env Util.) (%).

Metric	Non-Decoupled	Decoupled (Ours)	Improvement
$T_{\text{throughput}}$	22.6	43.6	1.9$\times$
Env Util.	12.2	67.7	5.5$\times$
GPU Util.	29.6	46.7	1.6$\times$

5.3 Efficiency Analysis

We evaluate our decoupled framework’s efficiency gains across three key metrics:

Env Util. (U_{env}): The ratio of time environments spend actively executing actions versus the total available time. $U_{\text{env}} = \frac{\sum_{i=1}^{N_{\text{env}}} t_{\text{active}}^{(i)}}{T_{\text{total}} \times N_{\text{env}}} \times 100\%$, where

$t_{\text{active}}^{(i)}$ denotes the active execution time of the i -th environment, N_{env} is the total number of environment instances (180), and T_{total} is the total wall-clock time of the experiment. **Training Throughput** ($T_{\text{throughput}}$): The average number of action steps collected and used in training per minute. $T_{\text{throughput}} = \frac{N_{\text{actions}}}{T_{\text{total}} (\text{min})}$, where N_{actions} represents the total number of valid action steps used in the training period. **GPU Util.** (U_{gpu}): The ratio of time the GPU(s) spend actively executing computation (e.g., inference and gradient updates) versus the total available time. $U_{\text{gpu}} = \frac{\sum_{j=1}^{N_{\text{gpu}}} t_{\text{active}}^{(j)}}{T_{\text{total}} \times N_{\text{gpu}}} \times 100\%$, where $t_{\text{active}}^{(j)}$ denotes the active computing time of the j -th GPU, and N_{gpu} is the number of GPUs used.

As shown in Table 2, under identical model, hardware, rollout configuration, and 180 environment instances, DART improves training throughput from 22.6 to 43.6 actions per minute ($1.9\times$), environment utilization from 12.2% to 67.7%, and GPU utilization from 29.6% to 46.7%. These gains arise from rollout-wise scheduling and per-worker synchronization, which reduce environment idle time and model-update blocking. They reflect improved wall-clock system efficiency rather than higher sample efficiency per policy update.

Table 3: Incremental ablation of the data-curation components. Each configuration retains all components introduced in the preceding row. DR denotes dynamic rollout, DTL denotes dynamic trajectory length, HE denotes high-entropy step filtering, and DA denotes distribution alignment.

Setting	Components Enabled				Pass@1 (%)
	DR	DTL	HE	DA	
Baseline					28.67
+ DR	✓				50.90
+ DTL	✓	✓			66.11
+ HE	✓	✓	✓		68.33
+ DA	✓	✓		✓	70.55
DART-GUI (Ours)	✓	✓	✓	✓	72.28

5.4 Ablation

We evaluate the contribution of each module in our data curation scheme on a subset of 45 training tasks. Table 3 details the incremental improvements, and Figure 5 visualizes the training dynamics. More ablations can be found in Appendix E. **Efficiency via Dynamic Allocation (DR & DTL).** The most significant gains come from adaptive resource allocation. Dynamic Rollout (DR) improves

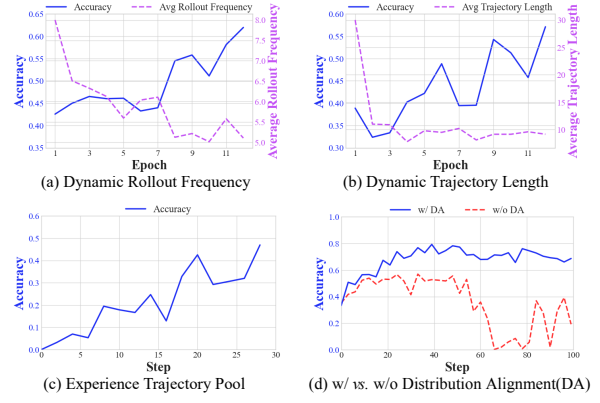


Figure 5: (a) Dynamic rollout vs. model accuracy across epochs. (b) Dynamic trajectory length vs. model accuracy over training. (c) Impact of experience trajectory pool on accuracy. (d) Performance comparison with and without distribution alignment.

Pass@1 from 28.67% to 50.90%. As shown in Figure 5(a), the system reduces rollout frequency (purple line) as model accuracy increases, effectively reallocating computational budget to harder tasks. Dynamic Trajectory Length (DTL) further boosts performance to 66.11%. Figure 5(b) illustrates that the average trajectory length decreases significantly during training, indicating the agent learns to solve tasks more efficiently with fewer steps.

Stability and Quality (HE & DA). High-Entropy (HE) step selection refines the policy by focusing on critical decision points, raising accuracy to 68.33%. Crucially, Distribution Alignment (DA) ensures training stability in our decoupled framework. Figure 5(d) reveals that without DA (red dashed line), the model suffers from performance collapse due to off-policy lag. Incorporating DA (blue line) corrects this distribution shift, stabilizing the score at 70.55%.

5.5 Generalization Analysis

We first evaluate transfer under broader distribution shifts using four external benchmarks. UI-Vision (Nayak et al., 2025) evaluates GUI understanding across diverse application domains and interface layouts; MobileWorld (Kong et al., 2026) focuses on mobile interaction; ScreenSpot-Pro (Li et al., 2025a) evaluates grounding in professional, high-resolution interfaces; and Mind2Web (Deng et al., 2023) covers web interaction across task, website, and domain splits. Table 4 reports category-wise results on UI-Vision and aggregate results on the other benchmarks.

DART-GUI-7B achieves comparable or improved performance across all four benchmarks.

Table 4: Generalization performance on external GUI benchmarks. UI-Vision reports category-wise and overall accuracy, MobileWorld reports the GUI-only score, ScreenSpot-Pro reports average grounding accuracy, and Mind2Web reports average Step SR.

Method	UI-Vision				MobileWorld	ScreenSpot-Pro	Mind2Web
	Basic	Func.	Spatial	Overall	GUI-only	Avg.	Avg. Step SR
UI-TARS-1.5-7B	20.82	21.84	7.49	16.44	6.0 ± 0.9	44.28	25.07
DART-GUI-7B (Ours)	21.67	22.40	8.48	17.24	8.5 ± 0.9	45.10	25.69

Table 5: Comparison with the SFT baseline on OSWorld with a maximum of 30 interaction steps.

Method	Success Rate (%)
UI-TARS-1.5-7B	24.17
UI-TARS-1.5-7B-SFT	28.14
DART-GUI-7B (Ours)	42.05

Table 6: Performance on 171 human-annotated unseen tasks.

Method	Success Rate (%)
UI-TARS-1.5-7B	27.19 ± 2.14
DART-GUI-7B (Ours)	36.06 ± 1.38

It improves the overall UI-Vision score from 16.44 to 17.24, the average ScreenSpot-Pro grounding accuracy from 44.28 to 45.10, and the average Mind2Web Step SR from 25.07 to 25.69. A larger gain is observed on MobileWorld, where the score increases from 6.0 ± 0.9 to 8.5 ± 0.9 . These results show that the policy transfers beyond OSWorld while retaining competitive GUI understanding, grounding, and web interaction capabilities. The larger gain on MobileWorld suggests that DART is particularly effective for long-horizon tasks requiring sustained interaction with the environment.

We next distinguish policy learning from trajectory imitation by comparing DART-GUI-7B with a Supervised Fine-Tuning (SFT) baseline. The SFT model is trained on 34k successful interaction steps collected from the base model via Pass@32, slightly more than the 30k steps used for RL training. As shown in Table 5, SFT improves the OSWorld score from 24.17% to 28.14%, whereas DART-GUI-7B reaches 42.05%. This result indicates that the gain cannot be explained by exposure to successful trajectories alone; online RL learns a more effective closed-loop policy for handling execution errors and environment stochasticity.

Finally, we evaluate DART-GUI-7B on 171 human-annotated tasks that are excluded from training. As shown in Table 6, DART-GUI-7B improves the score from 27.19% to 36.06%, an absolute gain

of 8.87 percentage points. Together with the external benchmark results, this improvement confirms that the learned policy generalizes to unseen task instances rather than relying on memorized training trajectories.

6 Conclusion

In this paper, we presented DART, a fully decoupled RL framework with adaptive data curation for training VLM-based GUI agents. DART focuses on two practical bottlenecks in GUI-agent RL: inefficient resource utilization during rollout and imbalanced learning signals from long-horizon interactions. Its four-module architecture enables asynchronous execution and worker-wise synchronization, while its multi-level data-curation strategy adaptively improves the efficiency and quality of online interactions. On OSWorld, DART-GUI-7B achieves an average task score of $43.28 \pm 0.65\%$ with 100 steps, demonstrating strong performance among open-source 7B-scale GUI agents. Controlled system comparisons and component-wise ablations validate the contributions of the decoupled architecture and adaptive data curation. Evaluations on held-out OSWorld tasks and four external benchmarks further demonstrate transfer beyond the online RL training tasks. By open-sourcing our framework, data, and model, we hope DART can provide a practical foundation for future research on RL-based GUI agents.

Acknowledgments. This work was supported by the Opening Project of the State Key Laboratory of General Artificial Intelligence No. SKLAGI2025OP07, the Natural Science Foundation of China (NSFC) under Grant No.62406009, the Shenzhen Science and Technology Program under Grant No. JCYJ20241202130548062, the Fundamental Research Funds for the Central Universities No. 2025CX11016, and the Key Research Platforms and Projects of the Guangdong Provincial Department of Education under Grant No. 2023ZDZX1034.

Limitations

DART has several limitations. First, it relies on the programmatic verifiers provided by OSWorld for reward computation. Although this setting is commonly adopted by existing GUI-agent RL methods (Lu et al., 2025; Yang et al., 2025; Lai et al., 2025), it limits direct applicability to open-world tasks whose success criteria are ambiguous or require human judgment. Since our reward module is architecturally decoupled from the training and rollout services, DART can in principle incorporate learned reward models, human feedback, or LLM-as-a-judge signals. Evaluating these alternatives in non-verifiable environments is left for future work.

Second, DART focuses on the systems architecture and adaptive data curation required for efficient RL in stateful GUI environments, complementing advances in RL objectives. We evaluate its system-level benefits against a controlled synchronous baseline using identical hardware, environment resources, and training configurations. Existing asynchronous RL frameworks, such as DistRL, Sky-RL, and AReL, are designed for different execution environments and system interfaces, making their direct deployment in desktop GUI environments non-trivial. Integrating these frameworks into a unified GUI-agent testbed would enable a broader study of asynchronous system designs and represents a valuable direction for future work.

Finally, deploying autonomous GUI agents introduces safety and privacy risks, including unintended modifications to files or accounts, exposure of sensitive information, and exploitation of shortcuts in programmatic reward functions. Practical deployments should therefore execute agents in sandboxed environments, enforce explicit permission boundaries, require human confirmation for irreversible or high-risk operations, and maintain detailed execution logs for auditing. Developing stronger safeguards against unsafe actions and reward hacking remains an important direction for future work.

Ethics Statement

The human annotation in this study comprised GUI task annotation and failure-mode analysis of interaction trajectories generated in isolated OSWorld environments. No personal or sensitive information was collected, and all participants were fully informed of the purpose of the evaluation. We

identify no additional ethical concerns beyond the safety and privacy risks discussed in Limitations.

References

- Anthropic. 2025a. [Claude 3.7 Sonnet and Claude Code](#). Technical report, Anthropic. System Card. 12, 13, 14.
- Anthropic. 2025b. [Claude-4 Sonnet](#). Technical report, Anthropic. System Card. 14.
- Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibao Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. 2025. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*.
- Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.
- Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, omega, and kubernetes. In *ACM Queue*, volume 14, pages 70–93. ACM.
- Shiyi Cao, Dacheng Li, Fangzhou Zhao, Shuo Yuan, Sumanth R Hegde, Connor Chen, Charlie Ruan, Tyler Griggs, Shu Liu, Eric Tang, et al. 2025. Skyrl-agent: Efficient rl training for multi-turn llm agent. *arXiv preprint arXiv:2511.16108*.
- DeepSeek-AI. 2025. [Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning](#). Preprint, arXiv:2501.12948.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.
- Tianyu Fu, Anyang Su, Chenxu Zhao, Hanning Wang, Minghui Wu, Zhe Yu, Fei Hu, Mingjia Shi, Wei Dong, Jiayao Wang, Yuyang Chen, Ruiyang Yu, Siran Peng, Menglin Li, Nan Huang, Haitian Wei, Jiawei Yu, Yi Xin, Xilin Zhao, Kai Gu, Ping Jiang, Sifan Zhou, and Shuo Wang. 2025a. Mano report. *arXiv preprint arXiv:2509.17336*.
- Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. 2025b. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*.
- Zhi Gao, Bofei Zhang, Pengxiang Li, Xiaojian Ma, Tao Yuan, Yue Fan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and Qing Li. 2025. Multi-modal agent tuning: Building a vlm-driven agent for efficient tool usage. In *International Conference on Learning Representations*, volume 2025, pages 13354–13385.

- Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, et al. 2025. Seed1.5-v1 technical report. *arXiv preprint arXiv:2505.07062*.
- Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 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.
- Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, et al. 2026. Mobile-world: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6142–6167.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. 2025. Computerrl: Scaling end-to-end online reinforcement learning for computer use agents. *arXiv preprint arXiv:2508.14040*.
- Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025a. Screenspot-pro: Gui grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 8778–8786.
- Pengxiang Li, Zhi Gao, Bofei Zhang, Yapeng Mi, Xiaojian Ma, Chenrui Shi, Tao Yuan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, et al. 2025b. Iterative tool usage exploration for multimodal agents via step-wise preference tuning. *arXiv preprint arXiv:2504.21561*.
- Pengxiang Li, Zhi Gao, Bofei Zhang, Tao Yuan, Yuwei Wu, Mehrtash Harandi, Yunde Jia, Song-Chun Zhu, and Qing Li. 2024. Fire: A dataset for feedback integration and refinement evaluation of multimodal models. *Advances in Neural Information Processing Systems*, 37:101618–101640.
- Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2024. Showui: One vision-language-action model for generalist gui agent. In *NeurIPS 2024 Workshop on Open-World Agents*, volume 1.
- Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, et al. 2024. Autoglm: Autonomous foundation agents for guis. *arXiv preprint arXiv:2411.00820*.
- Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. 2025. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*.
- Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. 2025. Arpo: End-to-end policy optimization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*.
- Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Pengxiang Zhao, Guangyi Liu, et al. 2026. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 17608–17616.
- Run Luo, Lu Wang, Wanwei He, and Xiaobo Xia. 2025. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*.
- Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A Rodriguez, Montek Kalsi, Rabiul Awal, Nicolas Chapados, M Tamer Özsu, Aishwarya Agrawal, David Vazquez, et al. 2025. Ui-vision: A desktop-centric gui benchmark for visual perception and interaction. *arXiv preprint arXiv:2503.15661*.
- OpenAI. 2025a. [Computer-using agent \(cua\): Model for gui interaction and task automation](#). Research preview / API documentation.
- OpenAI. 2025b. [OpenAI O3 and O4-Mini System Card](#).
- Oracle Corporation. 2024. [MySQL Database Management System](#). Accessed: 2024-09-24.
- Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*.
- Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, et al. 2025. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*.

- 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*.
- Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. 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.
- Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354.
- Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. 2025a. Uitars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.
- Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, et al. 2025b. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*.
- Taiyi Wang, Zhihao Wu, Jianheng Liu, Jianye HAO, Jun Wang, and Kun Shao. 2025c. Distrl: An asynchronous distributed reinforcement learning framework for on-device control agents. In *The Thirteenth International Conference on Learning Representations*.
- Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, et al. 2025d. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library. *arXiv preprint arXiv:2506.06122*.
- Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Dikang Du, Hao Hu, Huarong Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Y. Charles, Zhilin Yang, and Tao Yu. 2025e. [Opencua: Open foundations for computer-use agents](#). *Preprint*, arXiv:2508.09123.
- Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Biyik, David Held, and Zackory Erickson. 2024. RL-rlm-f: reinforcement learning from vision language foundation model feedback. In *Proceedings of the 41st International Conference on Machine Learning*, pages 51484–51501.
- Rui Xie, Zhi Gao, Chenrui Shi, Zirui Shang, Lu Chen, and Qing Li. 2026. Guide: Resolving domain bias in gui agents through real-time web video retrieval and plug-and-play annotation. *arXiv preprint arXiv:2603.26266*.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.
- Chenyu Yang, Shiqian Su, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, et al. 2025. Zerogui: Automating online gui learning at zero human cost. *arXiv preprint arXiv:2505.23762*.
- Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. 2025. [Your efficient rl framework secretly brings you off-policy rl training](#).
- Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. 2025. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.
- Bofei Zhang, Zirui Shang, Zhi Gao, Wang Zhang, Rui Xie, Xiaojian Ma, Tao Yuan, Xinxiao Wu, Song-Chun Zhu, and Qing Li. 2026. Tongui: Internet-scale trajectories from multimodal web tutorials for generalized gui agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 12367–12375.
- Xintong Zhang, Zhi Gao, Bofei Zhang, Pengxiang Li, Xiaowen Zhang, Yang Liu, Tao Yuan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, et al. 2025. Chain-of-focus: Adaptive visual search and zooming for multimodal reasoning via rl. *arXiv e-prints*, pages arXiv–2505.
- Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. 2023. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*.
- Xurui Zhou, Gongwei Chen, Yuquan Xie, Zaijing Li, Kaiwen Zhou, Shuai Wang, Shuo Yang, Zhuotao Tian, and Rui Shao. 2026. Hiconagent: History

context-aware policy optimization for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13028–13038.

A LLM Usage Statement

We acknowledge the use of LLMs as a writing assistance tool during the preparation of this manuscript. The LLMs were utilized exclusively for improving language quality, including grammar correction, and enhancing clarity. All scientific contributions, including research conceptualization, methodology, experimental design, data analysis, and interpretation of results were conducted solely by the human authors. The LLMs did not generate any original research ideas, hypotheses, or substantive scientific content. The authors assume full responsibility for the accuracy, integrity, and originality of all content presented in this work, including any portions where language was refined with LLM assistance.

B System Prompt and Action Space

We follow the system prompt of the baseline model UI-TARS-1.5-7B (Qin et al., 2025), as shown in Figure 6.

C Additional Experimental Settings

Trainer. The training pipeline utilizes Fully Sharded Data Parallel (FSDP) (Zhao et al., 2023) via verl (Sheng et al., 2024) for distributed training across 8 NVIDIA H100 GPUs. The learning rate is set to 1×10^{-6} with a KL divergence regularization coefficient of $\beta = 0.1$. Following DAPO (Yu et al., 2025), dynamic clipping boundaries are configured with $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.28$. The rollout policy scaling parameter is set to $C = 1$.

Rollout Service. We deploy the rollout policy using vLLM (Kwon et al., 2023). The rollout service uses another 8 NVIDIA H100 GPUs, organized into four independent rollout workers with 2 GPUs per worker. Therefore, the complete training system uses 16 NVIDIA H100 GPUs in total: 8 GPUs for policy optimization and 8 GPUs for rollout generation. A load balancer dynamically distributes rollout requests across available workers. We implement *worker-wise model synchronization*, which updates the policy weights of each rollout worker independently, allowing the remaining workers to continue sampling without waiting for a global synchronization barrier. The sampling temperature is set to 1.0, and the initial number of rollouts per task is set to $n_{\text{rollout}} = 8$.

Env Cluster. We use Kubernetes (K8s) (Burns et al., 2016) to orchestrate 180 parallel Ubuntu Docker containers serving as environment instances. Each environment operates independently,

receiving actions from agents and returning screenshots as observations. Figure 7 shows the dashboard of our cluster. The environments are decoupled from both the trainer and rollout workers, allowing completed environments to be immediately reassigned to new rollout requests. For all controlled efficiency comparisons, the synchronous non-decoupled baseline and DART use exactly the same hardware allocation, number of environment instances, model checkpoint, and rollout configuration. Therefore, the reported efficiency differences are measured under a controlled system configuration rather than estimated from different hardware setups.

Data Manager. We implement a centralized *Data Manager* using MySQL (Oracle Corporation, 2024) that handles data storage and coordination across the entire training pipeline. The database architecture, illustrated in Figure 8 and summarized in Table 8, comprises 11 interconnected tables organized into four functional categories.

For model management, the *Data Manager* tracks model checkpoints and versions through the checkpoint, current_model, and model_registry tables, enabling seamless model versioning and deployment. The data management subsystem, consisting of datasets, dataset_usage_events, rollout_run, and rollout_chunk tables, maintains comprehensive records of trajectories, their usage patterns, and associated rewards. Each trajectory is uniquely identified and linked to its corresponding run, task, and model version.

For the *Trainer*, the *Data Manager* ensures balanced training by monitoring trajectory outcomes through the reward field in multiple tables. When sampling training data, the *Data Manager* guarantees each task contains at least one successful trajectory (positive reward) and one failed trajectory (negative or zero reward). If all sampled trajectories for a given task fail, the *Data Manager* queries the datasets and rollout_run tables to retrieve positive trajectories from previously collected data, using the trajectory_id and task_id as keys to maintain task consistency. The dataset_usage_events table tracks these data access patterns, recording each usage with timestamps and model versions to ensure reproducibility. Additionally, the trainable_group table aggregates trajectories ready for training, while the update_model_task table manages the model update pipeline, coordinating between checkpoints

System Prompt for GUI Agent

You are a **GUI agent**. You are given a **task** and your **action history**, with **screenshots**. You need to perform the next action to complete the task.

Output Format

Thought: ...

Action: ...

Action Space

```
click(start_box='<|box_start|>(x1,y1)<|box_end|>')
left_double(start_box='<|box_start|>(x1,y1)<|box_end|>')
right_single(start_box='<|box_start|>(x1,y1)<|box_end|>')
drag(start_box='<|box_start|>(x1,y1)<|box_end|>',
     end_box='<|box_start|>(x3,y3)<|box_end|>')
hotkey(key="")
type(content="") # If you want to submit, use "\n" at the end
scroll(start_box='<|box_start|>(x1,y1)<|box_end|>',
       direction='down or up or right or left')
wait() # Sleep for 5s and take a screenshot
finished(content='xxx') # Use escape characters \', \", \n
```

Note

- Use {language} in Thought part.
- Write a **small plan** and finally **summarize** your next action (with its target element) in one sentence in Thought part.
- My computer's password is 'password', feel free to use it when you need sudo rights.

User Instruction

{instruction}

Figure 6: System prompt template for DART-GUI agent with action space definition and output format specifications.



Figure 7: Dashboard of the distributed Ubuntu env cluster.

and deployment stages.

Dynamic Rollout Number. To balance sampling efficiency and task diversity during data collection, we adopt a dynamic rollout scheduling strategy that adaptively adjusts the rollout number according to the task success rate. Specifically, tasks with lower success rates are assigned more rollout attempts in the subsequent sampling stage, encouraging further exploration and increasing the probability of collecting successful trajectories. In contrast, tasks with consistently high success rates require fewer rollout attempts, thereby reducing redundant sampling and improving overall computational efficiency.

As shown in Table 7, the rollout number decreases progressively as the task accuracy increases. We discretize the success rate into several intervals and assign a corresponding rollout number for the next sampling iteration. For example, tasks with an accuracy lower than 60% are allocated 8 rollout attempts, while tasks exceeding 90% accuracy only receive 2 rollout attempts. This design enables the framework to automatically focus more computational resources on difficult tasks while gradually reducing the sampling frequency for well-mastered tasks.

Compared with using a fixed rollout number for all tasks, the proposed dynamic rollout strategy provides a more balanced trade-off between exploration and efficiency, preventing excessive sampling on saturated tasks and improving the utilization of rollout budgets across the entire task distribution.

Table 7: Mapping of Task Success Rate (Acc.) to Dynamic Rollout (DR.) numbers in the next sampling for the same task. By transposing the layout, we illustrate the discrete steps of frequency reduction as success rates improve.

Acc.(%)	[0.0, 0.6]	(0.6, 0.65]	(0.65, 0.7]	(0.7, 0.8]	(0.8, 0.9]	(0.9, 1.0]
DR. (n)	8	7	6	5	4	2

Details of the UI-Vision Benchmark. As shown in Table 9, UI-Vision (Nayak et al., 2025) includes fine-to-coarse-grained grounding tasks to evaluate the model’s performance on understanding professional software, spatial reasoning, and complex actions in desktop environments.

Construction of Unseen Evaluation Tasks. To evaluate generalization beyond the training set, members of our research team manually con-

structed 171 new GUI tasks that were excluded from model training. We reused the existing OS-World Docker environments while authoring new natural-language task instructions and corresponding programmatic evaluation scripts. All task content was created specifically for benchmark evaluation and was screened to ensure that it contained no personal, private, or sensitive information, nor any data collected from real users. No external annotators or human participants were involved in constructing this evaluation set.

C.1 Experience Pool of Trajectories

Training on challenging tasks poses a significant obstacle in reinforcement learning due to extremely low success rates, which results in insufficient positive trajectories in rollouts. It is common that all trajectories of a task fail, providing no effective learning signal for policy improvement. This severe imbalance between simple tasks and difficult tasks results in training instability, preventing the model from learning correct behavioral patterns. To address this limitation, we introduce an *Experience Pool* that serves as a repository of high-quality successful trajectories for challenging tasks, enabling dynamic supplementation for difficult tasks during training to ensure balanced learning signals.

Specifically, we pre-populate the *Experience Pool* by collecting and storing high-quality successful trajectories through preliminary sampling. During training, when the system detects that all trajectories in the current task fail, it automatically triggers the pool sampling mechanism to randomly retrieve a successful trajectory and incorporate it into the training batch. This design guarantees that every training task contains at least one positive sample while maintaining a reasonable balance between exploration and experience replay, thereby preventing performance degradation on challenging tasks and improving overall training stability.

D Comparison with Related Frameworks

In this section, we provide a detailed comparison between DART and other related RL frameworks, highlighting the specific architectural and algorithmic choices driven by the desktop GUI domain.

Comparison with DistRL. DistRL (Wang et al., 2025c) is a highly relevant baseline that also employs an asynchronous framework with decoupled training. However, the distinction between DART and DistRL is fundamental, driven by the differ-

Table 8: Database tables categorized by functionality.

Category	Table Count	Tables
Model Management	3	checkpoint, current_model, model_registry
Data Management	4	datasets, dataset_usage_events, rollout_run, rollout_chunk
Training	2	trainable_group, update_model_task
Inference	2	inference_node, inference_tasks
Total	11	

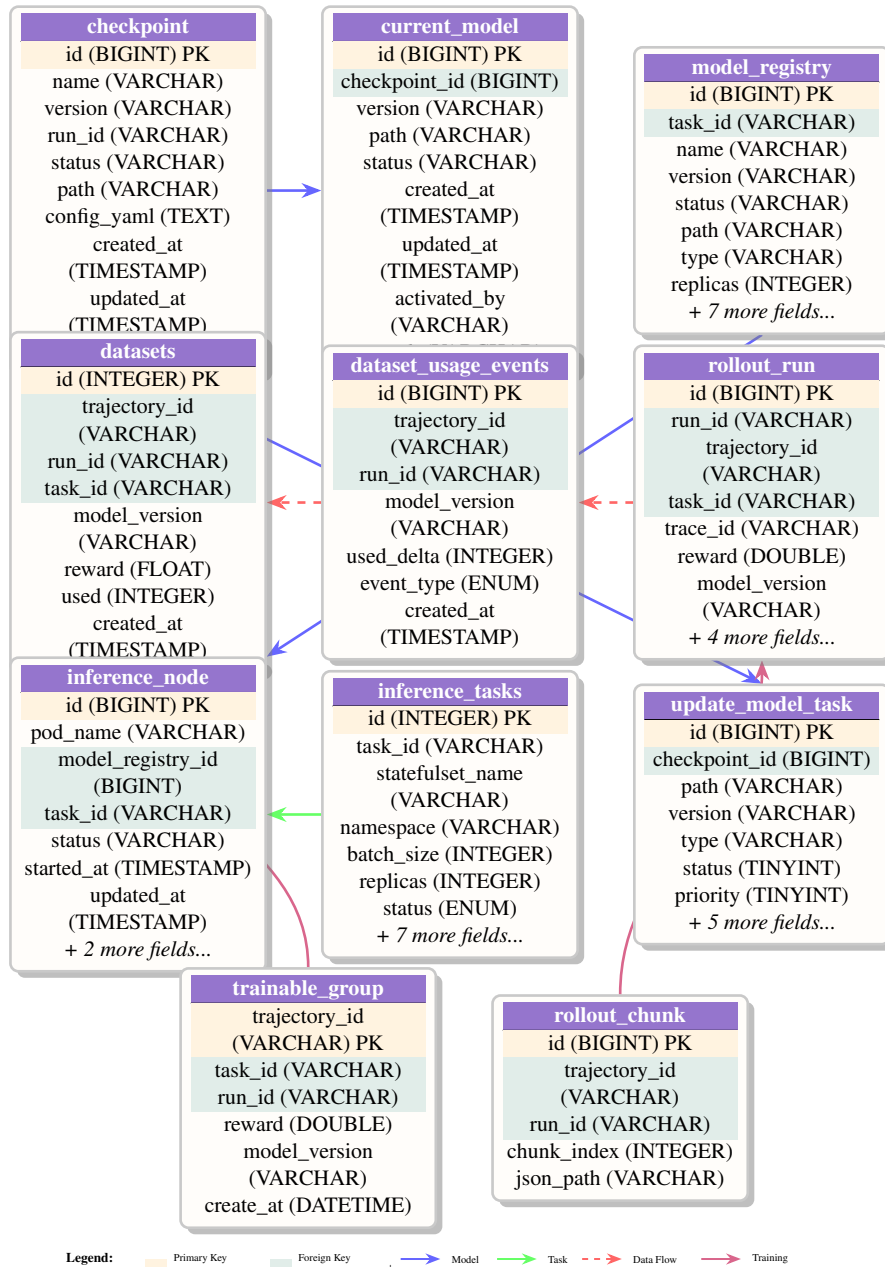


Figure 8: Database Schema Visualization showing the relationships between 11 tables. Golden cells indicate primary keys, green cells indicate foreign keys. Blue arrows represent model management relationships, green arrows show task dependencies, red dashed arrows indicate data flow between rollout and dataset tables, and purple arrows represent training data connections.

Table 9: Categorized list of software applications included in the UI-Vision benchmark (Nayak et al., 2025).

Category	Software Applications
Education	Anki, Zotero, GrassGIS, Calibre, Audacity, QGIS, OpenBoard, Mendeley
Browsers	Brave, Chromium, Mozilla Firefox, DuckDuckGo
Development	VSCode, Atom, FreeCAD, Eclipse, NetBeans, PyCharm, IntelliJ IDEA, Brackets, Geany, Bluefish, KDevelop, Komodo Edit, Code::Blocks, Qt Creator, Arduino IDE, Spyder, Ubuntu Terminal, Conky, Bash, gedit
Productivity	LibreOffice Calc, LibreOffice Draw, LibreOffice Impress, LibreOffice Writer, draw.io, Joplin, OpenProject, Affine, Zulip, PDFedit, OnlyOffice Calendar, OnlyOffice Document Editor, OnlyOffice Forms, OnlyOffice PDF Forms, OnlyOffice Presentation, OnlyOffice Spreadsheet, Nextcloud, Gnumeric, Simplenote, Cryptomator, WeKan, 7-Zip, GnuCash, Bitwarden, Metabase, Jitsi, Flameshot, Nemo
Creativity	Blender, GIMP, Inkscape, Krita, darktable, FontForge, MuseScore, Scribus, OpenShot, OBS Studio, Lightworks, Shotcut, Natron, OpenToonz, WordPress
Entertainment	VLC Media Player, Kodi, Element, Signal, Mastodon, Lemmy, Matrix, Emby

ent requirements of desktop GUI versus mobile GUI tasks. As detailed in Table 10, these domain differences lead to key innovations in system architecture, sampling strategy, and training algorithms. DART offers three specific advantages over DistRL beyond implementation details:

1. **Finer-Grained Decoupling for Long-Duration Tasks:** DART introduces a four-module decomposition by separating the Rollout Service from the Environment Cluster. This design directly addresses the bottleneck of environment availability in desktop GUI tasks, where episodes last 5–20 minutes. In contrast, DistRL’s integrated worker model suffices for mobile tasks lasting seconds to minutes.
2. **High-Freshness Sampling for Near-On-Policy Learning:** DART employs a high-freshness, single-use sampling strategy, where each trajectory is used exactly once and the version gap is constrained to one step. This approximates on-policy learning, avoiding the complexity of off-policy corrections (e.g., $\text{Retrace}(\lambda)$ used in DistRL). This strategy ensures greater training stability and higher throughput ($1.9\times$).
3. **Domain-Optimized Algorithmic Simplicity:** DART uses step-wise GRPO with rule-based rewards, tailored to the long-horizon nature of desktop tasks. This contrasts with DistRL’s

reliance on LLM-based rewards and learned value networks, which introduce additional complexity.

Comparison with Sky-RL. We also compare DART with Sky-RL (Cao et al., 2025), a representative framework for text-based tool use. While both systems aim to improve multi-turn RL efficiency, they address fundamentally different bottlenecks. As analyzed in Table 11, Sky-RL focuses on logic-heavy scenarios where CPU-bound tool execution is limiting, employing standard pipelined synchronization to ensure data consistency. In contrast, DART targets the visual-heavy Desktop GUI domain, where the high variance of VLM inference latencies creates synchronization challenges. To address this, DART implements a **per-worker non-blocking synchronization** strategy. This allows workers to stagger updates independently, preventing the system-wide idle time that would otherwise occur during heavy visual processing.

Empirical Efficiency Comparison. For empirical performance comparison, we focus on frameworks natively designed for GUI agents: ARPO (Lu et al., 2025) and ZeroGUI (Yang et al., 2025). As shown in Table 12, DART significantly outperforms existing methods:

- **vs. ARPO:** ARPO employs a standard synchronous rollout-training loop, architecturally equivalent to our “Non-Decoupled” baseline. DART achieves a $1.9\times$ speedup and significantly higher accuracy (42.13% vs 29.9%).

Table 10: Comparison between DART and DistRL.

Aspect	DART (Ours)	DistRL
Domain	Desktop GUI	Mobile GUI
Task Time	5-20 minutes per episode	Seconds to minutes per episode
Decoupled Modules	Four Modules: Trainer, Rollout Service, Environment Cluster, Data Manager	Two Modules: Host Learner, Worker (integrated inference + environment)
Samples	High-freshness, single-use trajectories	Replay-based, reused trajectories
Training Algorithms	Step-wise GRPO, a nearly on-policy RL algorithm	A-RIDE, an off-policy RL algorithm with off-policy corrections
Rewards & Advantages	Rule-based rewards (OSWorld scripts) + Group-normalized advantages	LLM-based rewards (Gemini-1.5-Pro) + Learned value network

Table 11: Comparison of efficiency bottlenecks and architectural optimization strategies between DART and Sky-RL.

Feature	DART (Ours)	Sky-RL
Target Domain	Desktop GUI (Visual-heavy)	Code/Research (Text/Logic-heavy)
Primary Bottleneck	VLM Inference & Rendering (Encoding high-res screenshots)	Tool Execution Latency (CPU-bound tasks like running tests)
Efficiency Strategy	Decoupled & Asynchronous (Separating Trainer/Rollout to maximize VLM throughput)	Async Pipeline (Hiding CPU tool latency behind GPU inference)
Model Sync	Per-Worker Non-Blocking Sync (Staggered updates to prevent idle time)	Standard/Pipelined Sync (Optimized for data flow consistency)

- **vs. ZeroGUI:** ZeroGUI introduces substantial overhead by replacing rule-based verification with an LLM-based reward model, requiring 4x VLM inferences per trajectory. DART achieves superior performance with a lightweight rule-based reward system.

E More Results and Ablation

E.1 Performance on Training and Held-out OSWorld Tasks

To better understand the generalization behavior of DART, we divide the 361 OSWorld evaluation tasks into 203 tasks used during online RL training and 158 held-out tasks that were never used for training. Following the task-selection protocol described in prior work, the training tasks were selected from those solvable by UI-TARS-1.5-7B under Pass@32. Therefore, this split is not a random partition of OSWorld, and the two subsets have substantially different levels of difficulty for the initial policy.

As shown in Table 13, DART improves the performance on the training split from 42.98% to 61.97%, yielding an absolute gain of 18.99 percentage points. More importantly, DART achieves 16.46% on the 158 held-out tasks, whereas the initial UI-TARS-1.5-7B policy obtains 0.00% under

the same evaluation budget. The comparable improvements on the training and held-out splits suggest that the policy improvement is not restricted to memorizing the training environments, but transfers to tasks that are not encountered during online RL.

Nevertheless, a considerable gap remains between the training and held-out results (61.97% versus 16.46%). This gap is expected because the training tasks were intentionally selected according to the solvability of the initial policy, while the held-out subset contains substantially more challenging tasks. Therefore, we interpret the held-out result as evidence of cross-task generalization rather than performance on a randomly sampled, identically distributed test split. The overall results are the task-count-weighted averages of the two subsets; for example, DART obtains $(203 \times 61.97 + 158 \times 16.46) / 361 = 42.05\%$.

E.2 Training Dynamics.

Figure 9 illustrates the training dynamics. The *Accuracy* (blue) steadily climbs from ~ 0.50 to over 0.70, confirming the agent successfully maximizes cumulative rewards. Simultaneously, the *Average Trajectory Length* (purple) drops from ~ 16 to 10 steps, demonstrating that the agent learns to optimize efficiency by identifying shorter reasoning

Table 12: Estimated Efficiency and Performance Comparison against ARPO and ZeroGUI.

Framework	Architecture	Reward Mechanism	Training Throughput	Env Util.(%)	Success Rate(%)
ARPO	Coupled (Sync)	Rule-based	22.6 actions/min	12.2	29.9
ZeroGUI	Coupled (Sync)	VLM-based (4x Voting)	19.4 actions/min	11.4	20.2
DART (Ours)	Decoupled (Async)	Rule-based	43.6 actions/min	67.7	42.1

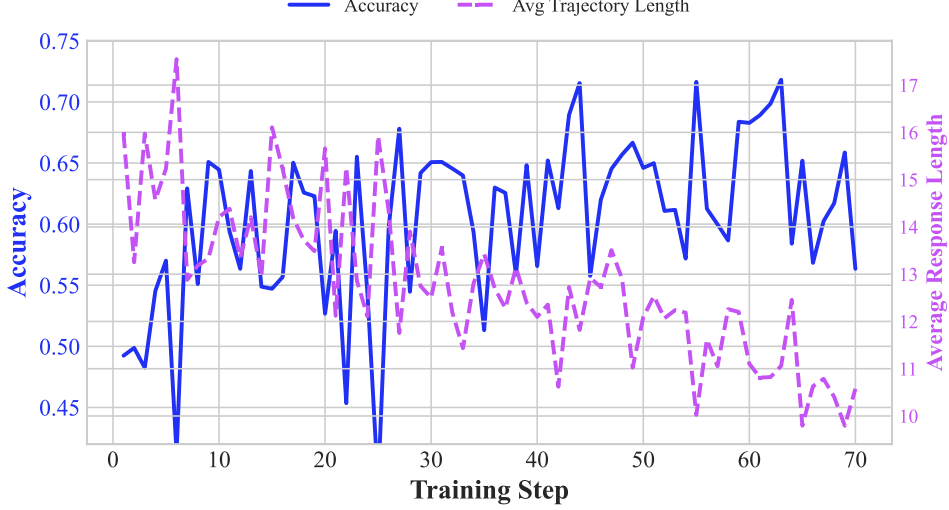


Figure 9: Training Dynamics.

Table 13: Performance on the OSWorld training and held-out task splits. All models are evaluated with a maximum of 30 interaction steps. DART results are averaged over three independent evaluation runs, with the standard deviation reported for the overall result. Scores are task-level rewards in $[0, 1]$, reported as percentages.

Task Split	# Tasks	UI-TARS-1.5-7B	DART	Improvement
Training tasks	203	42.98	61.97	+18.99
Held-out tasks	158	0.00	16.46	+16.46
All tasks	361	24.17	42.05 $\pm$ 1.50	+17.88

paths to the solution.

We conduct additional ablation studies on the full OSWorld benchmark to further isolate the contributions of our framework components and verify the robustness of our data strategy.

E.3 Full-Set Framework Ablation.

To verify the impact of the Decoupled Framework and the Distribution Alignment (DA) module beyond the subset used in the main text, we conducted experiments on the full OSWorld benchmark. As shown in Table 14, the results isolate the gains from specific components:

- **Effect of Decoupling:** Even without Distribution Alignment, our decoupled architec-

ture (‘Ours w/o DA’) achieves 34.77%, significantly outperforming the coupled baseline (24.17%). This confirms the intrinsic benefit of the asynchronous design in improving data collection and throughput.

- **Effect of Distribution Alignment:** Adding the DA module (‘Ours w/ DA’) boosts performance to 38.81%, validating that correcting distributional shifts is critical for stability at scale.

E.4 Impact of Experience Pool.

To address concerns that pre-collecting successful trajectories for hard tasks might lead to memorization, we conducted an ablation removing the Experience Pool entirely. As shown in Table 15, a marginal drop of 1.11 percentage points demonstrates that the model can successfully learn robust policies through online sampling alone, albeit with higher computational cost.

F Visualization

F.1 Key Steps of Tasks.

We visualize the comparison between the baseline and our model on several tasks, demonstrating

Table 14: Incremental analysis of the decoupled RL framework and data-curation components on the full OSWorld benchmark with a maximum of 30 interaction steps. The initial checkpoint is UI-TARS-1.5-7B before online RL training. Each subsequent row cumulatively adds the corresponding components. Ablation results are based on a single evaluation run.

Setting	Task Score (%)	Gain over Previous (pp)
Initial checkpoint (UI-TARS-1.5-7B, before RL)	24.17	–
Decoupled RL w/o DA	34.77	+10.60
+ DA	38.81	+4.04
+ DR + DTL + HE (Full DART)	42.13	+3.32

Table 15: Ablation study removing pre-collected trajectories (Experience Pool). Ablation results are based on a single evaluation run.

Configuration	Success Rate (%)
DART w/o pre-collected trajectories	41.02
DART (Full Method)	42.13

that through our RL training, our model can make correct actions at critical steps, thereby achieving successful trajectories. The comparative analysis clearly shows the improvement in decision-making at pivotal moments, as illustrated in Figure 10 and Figure 11.

F.2 Visualization of Extremely Difficult Tasks.

By leveraging pre-collected successful trajectories from the trajectory pool for extremely difficult tasks (where pass@32 failed), our trained model demonstrates the ability to generate correct trajectories on these challenging tasks. We visualize the critical steps that truly determine the success or failure of trajectories in these tasks and provide detailed analysis of the underlying reasons, as shown in Figure 12 and Figure 13.

F.3 Failure Cases

Visualization of Failure Cases. We also visualize representative failure cases to highlight the limitations of our model. These examples demonstrate situations where DART-GUI-7B makes mistakes at key steps, preventing successful task completion, as illustrated in Figure 14.

Statistics of Failure Cases. To provide a mechanistic understanding of the model’s limitations, we conducted a comprehensive human study on 65 failed tasks from DART-GUI-7B and 55 corrected tasks (where DART succeeded but the baseline failed). We categorize the trajectories into five distinct failure modes:

- **Reasoning Failures:** The agent correctly grounds UI elements but selects an incorrect action or logical sequence (e.g., clicking “Done” before changing a setting).
- **Grounding Errors:** The agent fails to locate the target element or generates hallucinated coordinates.
- **Looping/Stuck:** The agent enters a repetitive action sequence, or the environment remains trapped in a stagnant state (e.g., unclosable pop-ups or unresponsive pages) preventing progress.
- **Knowledge Limitations:** The agent lacks the specific domain knowledge or application usage experience required to solve the task.
- **Action Space Limitations:** The agent generates a valid intent, but the action fails due to API constraints or complex UI interactions (e.g., specific drag-and-drop mechanics).

The detailed statistical breakdown of these error modes is presented in Table 16.

We further visualize this distribution shift in Figure 15. The comparison highlights that DART significantly reduces Reasoning Failures (dropping from 61.8% in corrected tasks to 40.0% in remaining errors) and Grounding Errors (16.4% to 7.7%). As the model becomes more capable of handling reasoning and grounding challenges, Knowledge Limitations (rising to 38.5%) emerge as the primary bottleneck for the remaining unsolved tasks.

We analyze the correlation between failure modes and trajectory length. As shown in Figure 16, successful trajectories from DART-GUI-7B exhibit a remarkably even distribution across all length intervals (consistently ranging between 15% and 19%), demonstrating robustness across both short and long-horizon tasks. In contrast, baseline failures are heavily skewed toward the maximum

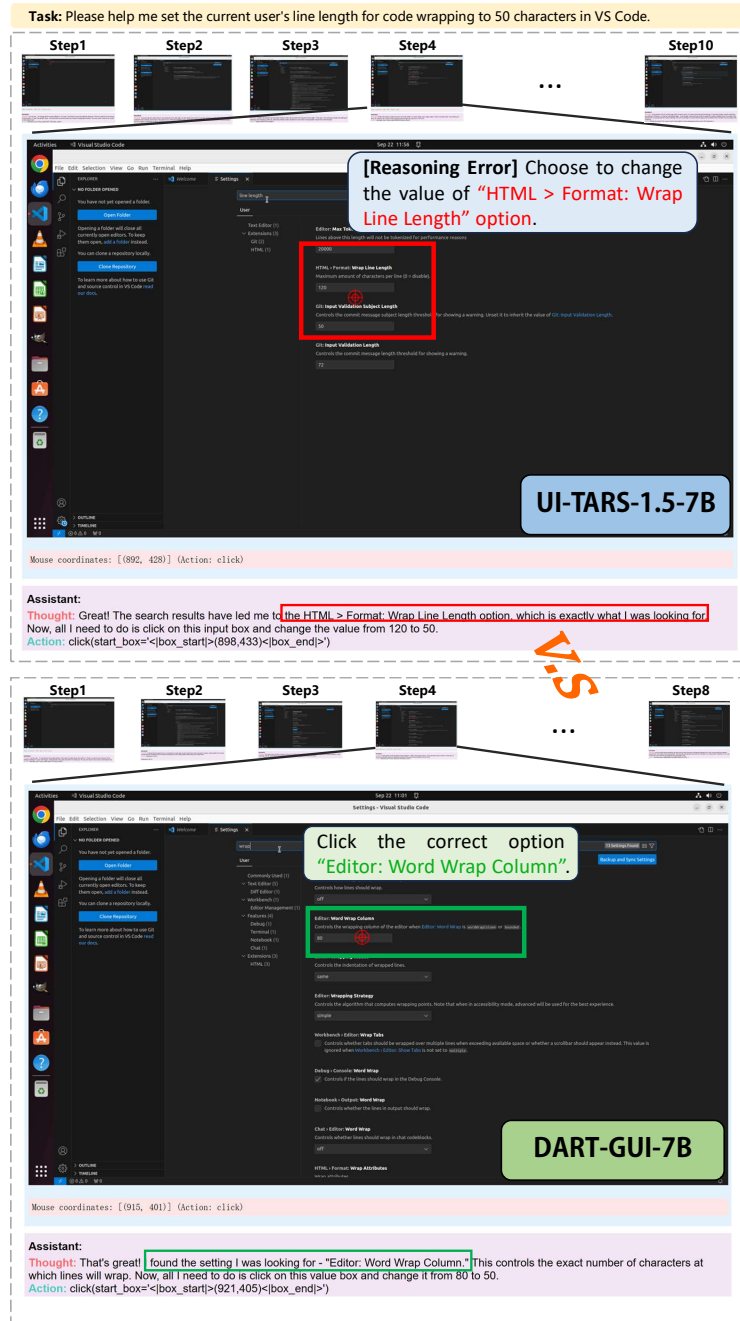


Figure 10: Case study comparing UI-TARS-1.5-7B and DART-GUI-7B on configuring line wrapping in VS Code. UI-TARS-1.5-7B exhibits a reasoning error by modifying the unrelated HTML > Format: Wrap Line Length option, whereas DART-GUI-7B correctly locates and sets the Editor: Word Wrap Column parameter to the desired value.

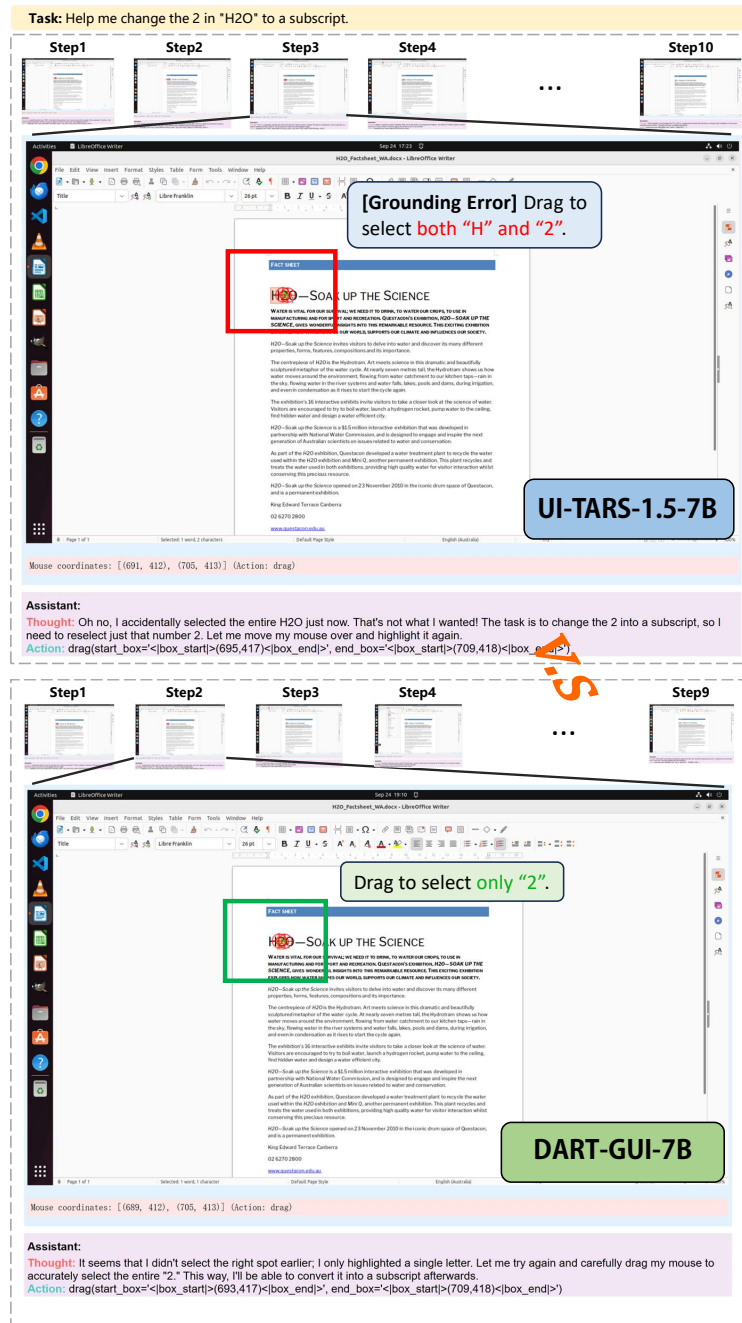


Figure 11: Case study comparing UI-TARS-1.5-7B and DART-GUI-7B on editing text in LibreOffice. UI-TARS-1.5-7B makes a grounding error by selecting both “H” and “2” in “H₂O”, whereas DART-GUI-7B correctly highlights only the “2” for conversion into a subscript.

Task: On it Whenever I launch a LibreOffice Impress, it uses both screens, one for current slide and next slide and another for actual presentation. What I want is to use only one monitor which shows presentation. I dont want the screen with Current slide and Next slide so that it can be used for other purposes. How should I achieve this?

Step1Step2Step3Step4...Step8

UI-TARS-1.5-7B

Mouse coordinates: [(406, 77)] (Action: click)

Assistant:
Thought: The interface has fully loaded now, and I've noticed there's a "Slide Show" option in the top menu bar. This should be where I can adjust how the slides are displayed. Let me click on it to see if I can find any options related to single-screen display.
Action: click(start_box=<[box_start>(409,78)<[box_end>")

DART-GUI-7B

Mouse coordinates: [(483, 74)] (Action: click)

Assistant:
Thought: The interface has finally finished loading, and now I need to adjust the presentation display settings. Let me see... Ah, there's a "Tools" option in the top menu bar, clicking on it should lead me to the preferences where I can modify how the slides are displayed.
Action: click(start_box=<[box_start>(486,75)<[box_end>")

Figure 12: Case study on an extremely difficult LibreOffice Impress task. The task requires configuring dual-slide display settings. The baseline model (top) incorrectly clicks "Slide Show" in the menu, leading to task failure. Our DART-GUI-7B model (bottom), trained with successful trajectories from the trajectory pool, correctly selects "Tools" to access the preferences panel where dual-slide display can be configured. This demonstrates our model's ability to learn from rare successful trajectories and solve previously intractable tasks through RL training.

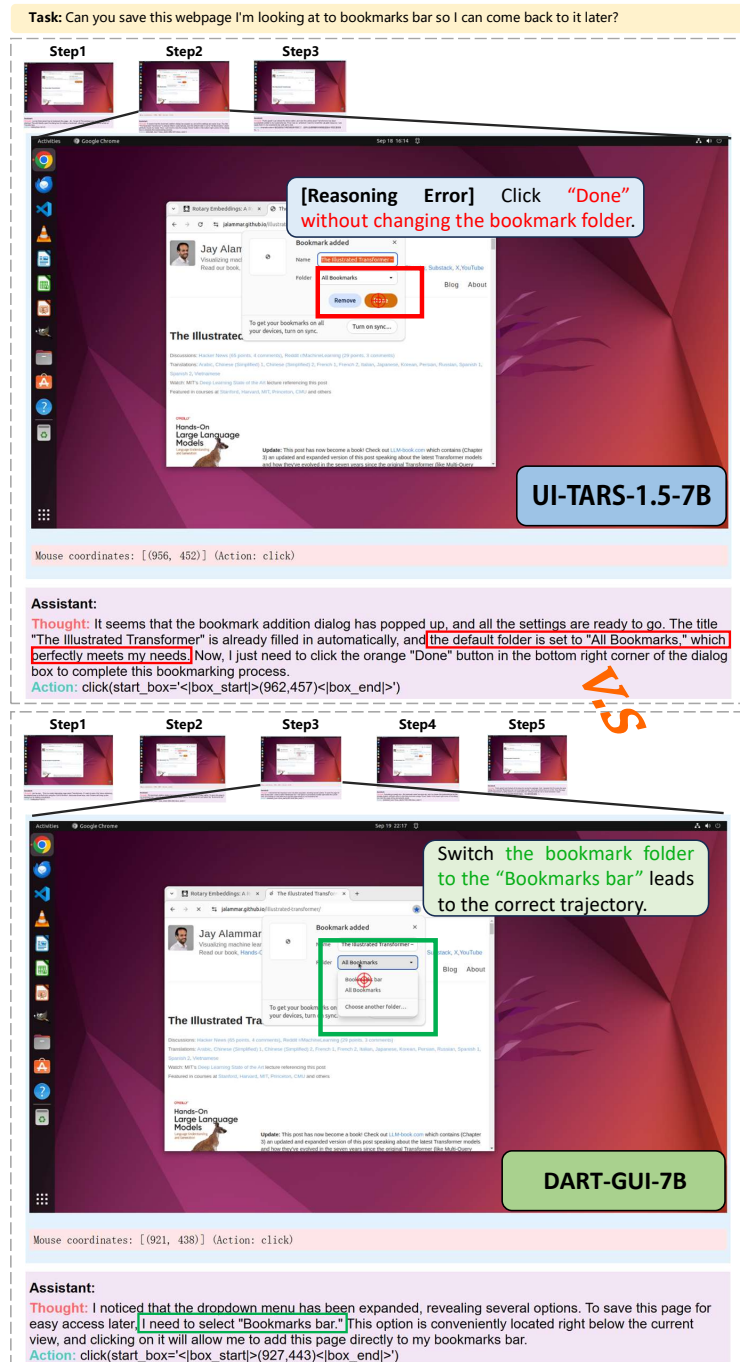


Figure 13: Case study on an extremely difficult bookmark saving task. The task requires saving a webpage to the bookmarks bar for quick access. The baseline model (top) makes a critical error by clicking "Done" without changing the bookmark folder from the default "All Bookmarks" to "Bookmarks bar," resulting in task failure. Our DART-GUI-7B model (bottom) correctly identifies the need to switch the folder dropdown to "Bookmarks bar" before confirming, successfully completing the task. This demonstrates our model's ability to understand subtle but crucial UI requirements that determine task success, learned through RL training on rare successful trajectories.

Table 16: Error distribution across tasks failed by the baseline but corrected by DART, and the remaining failures of DART-GUI-7B.

Error Category	Baseline Failures (Corrected by DART)	DART-GUI-7B Failures (Remaining Errors)
Reasoning Failures	61.82%	40.00%
Grounding Errors	16.36%	7.69%
Action Space Limitations	1.82%	6.15%
Looping/Stuck	1.82%	7.69%
Knowledge Limitations	18.18%	38.46%

length (peaking at 31.2%), indicating that the baseline model often struggles to terminate effectively or enters stuck states in complex scenarios.

Figure 17 presents the Pearson correlation coefficients between specific failure modes and task metrics. The statistical analysis identifies key failure patterns. There is a notable positive correlation (0.29) between Looping/Stuck states and trajectory length, confirming that agents stuck in repetitive loops tend to exhaust the step limit. Additionally, Knowledge Limitations show a positive correlation (0.18) with task complexity, suggesting that as tasks become more intricate, the primary challenge shifts from execution to a lack of specific domain knowledge.

Finally, we observe that failure modes vary significantly depending on the application interface, as illustrated in Figure 18. LibreOffice Calc, characterized by a dense grid of small cells, is dominated by Grounding Errors (100%), reflecting the difficulty in precise coordinate prediction. In contrast, applications like Chrome and LibreOffice Writer, which involve multi-step menu navigation, primarily exhibit Reasoning Failures (>75%), indicating that the main challenge in these environments lies in logical planning rather than visual element identification.

F.4 Human Annotation Protocol

Annotation Data and Assignment. We invited 20 engineers with relevant experience in GUI agents and software systems to conduct the failure-mode analysis. The annotation set contained 120 task cases, including 55 baseline failures that were corrected by DART-GUI-7B and 65 remaining failures of DART-GUI-7B. Every annotator independently evaluated the complete set of 120 cases, resulting in 2,400 case-level judgments. For each case, annotators were provided with the task instruction, the complete interaction trajectory, including screenshots, reasoning traces, and executed

actions, and the final task outcome. The presentation order of the cases was independently randomized for each annotator to reduce ordering effects.

Before annotation, the task descriptions and trajectory records were screened to exclude personally identifying, sensitive, offensive, or otherwise non-compliant content. All trajectories were generated in isolated benchmark environments and did not contain data collected from real users. Annotators were instructed to report any case that they considered sensitive, identifying, offensive, or otherwise inappropriate.

Annotation Instructions. All annotators received the same instructions:

Review the task instruction and the complete interaction trajectory, including the screenshots, reasoning traces, executed actions, and final outcome. Identify the earliest error that directly prevents successful task completion and assign exactly one dominant failure category:

1. **Reasoning Failure:** The agent correctly identifies or grounds the relevant interface elements but selects an incorrect action, plan, or logical sequence.
2. **Grounding Error:** The agent fails to locate the intended interface element, selects an incorrect element, or produces inaccurate or hallucinated coordinates.
3. **Looping/Stuck:** The agent repeatedly performs ineffective actions or remains in a stagnant state without making meaningful progress.
4. **Knowledge Limitation:** The agent lacks the task-specific or application-specific knowledge required to determine the correct operation.

5. **Action Space Limitation:** The agent expresses an appropriate intent, but the operation cannot be reliably executed using the available action interface, such as an unsupported compound key operation or a complex drag-and-drop action.

If multiple errors occur in the same trajectory, select the earliest error that causally prevents task completion. If the categorization is uncertain, select the most causally relevant category and provide an optional explanatory note. Report any case containing sensitive, identifying, offensive, or otherwise inappropriate material to the research team.

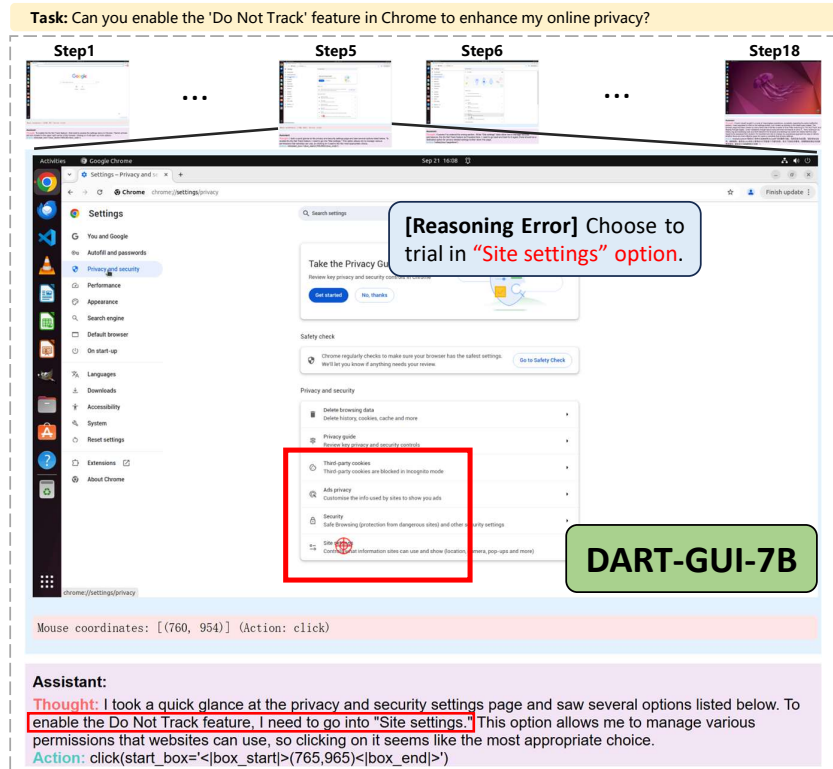
Aggregation. We retained every individual annotation rather than replacing the annotations with a single majority-vote label. For a task subset $\mathcal{S}$ and failure category c , we calculate its annotation-level proportion as

$$p_c = \frac{\sum_{i=1}^{20} \sum_{j \in \mathcal{S}} \mathbb{I}[y_{i,j} = c]}{20|\mathcal{S}|}, \quad (3)$$

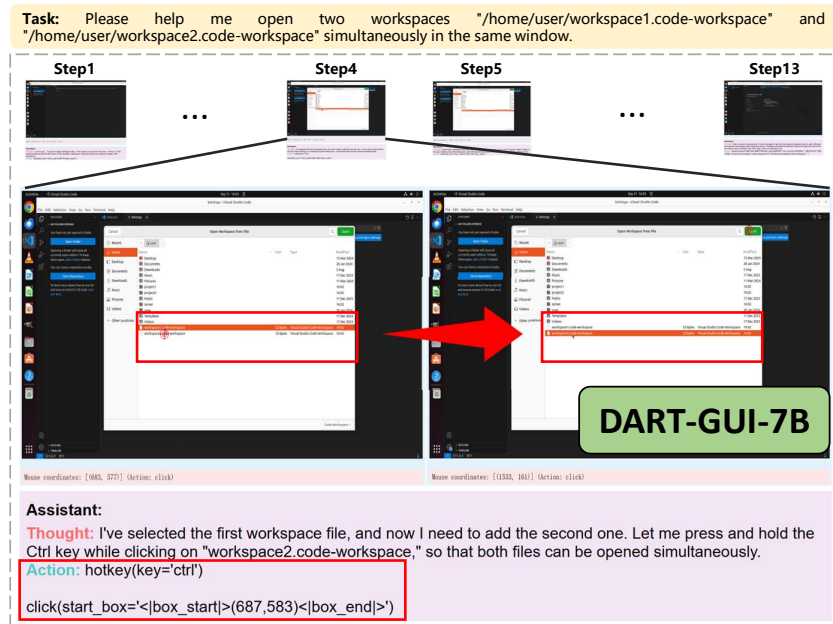
where $y_{i,j}$ is the category assigned by annotator i to case j . Consequently, the reported distributions are computed from 1,100 judgments for the 55 corrected baseline failures and 1,300 judgments for the 65 remaining DART-GUI-7B failures. This aggregation preserves annotator-level variation while giving every annotator and every case equal weight.

Recruitment, Compensation, and Consent. The annotators were engineers with relevant professional experience and were recruited through direct invitation. Participation was voluntary, and all annotators received financial compensation under the applicable institutional policy for professional annotation work. The compensation level was determined based on the expected workload and required expertise and exceeded the legal requirements.

Before participation, annotators were informed of the purpose of the study, the annotation procedure, and the intended use of their annotations in aggregate research analysis and publication. No sensitive demographic information was collected.



(a)



(b)

Figure 14: Failure cases of DART-GUI-7B. (a) For the task of enabling the “Do Not Track” feature in Chrome, the model incorrectly clicks the “Site settings” option instead of the “Third-party cookies” option required to access the relevant privacy control. (b) For the task of opening two workspaces simultaneously in VS Code, the model attempts a Ctrl+click sequence, but due to action space limitations, the execution corresponds to sequentially pressing “Ctrl” and clicking without holding “Ctrl”, which deselects the first workspace and leaves only the second one selected.

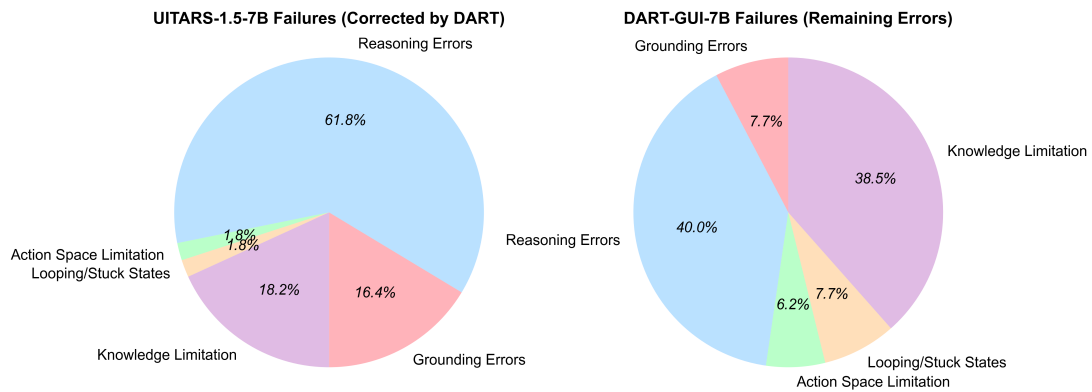


Figure 15: Comparison of Failure Mode Distributions. The charts contrast the error distribution of tasks failed by the baseline but corrected by DART (Left) against the remaining failures of DART-GUI-7B (Right).

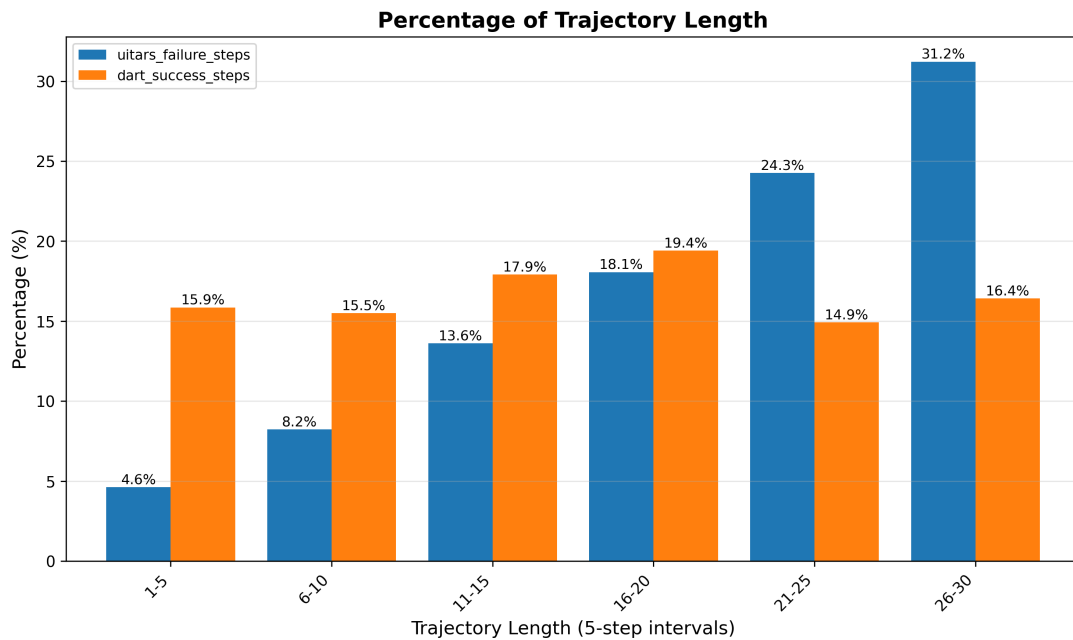


Figure 16: Distribution of Trajectory Lengths. A comparison between the trajectory lengths of DART's successful tasks (Orange) and the baseline's failed tasks (Blue).

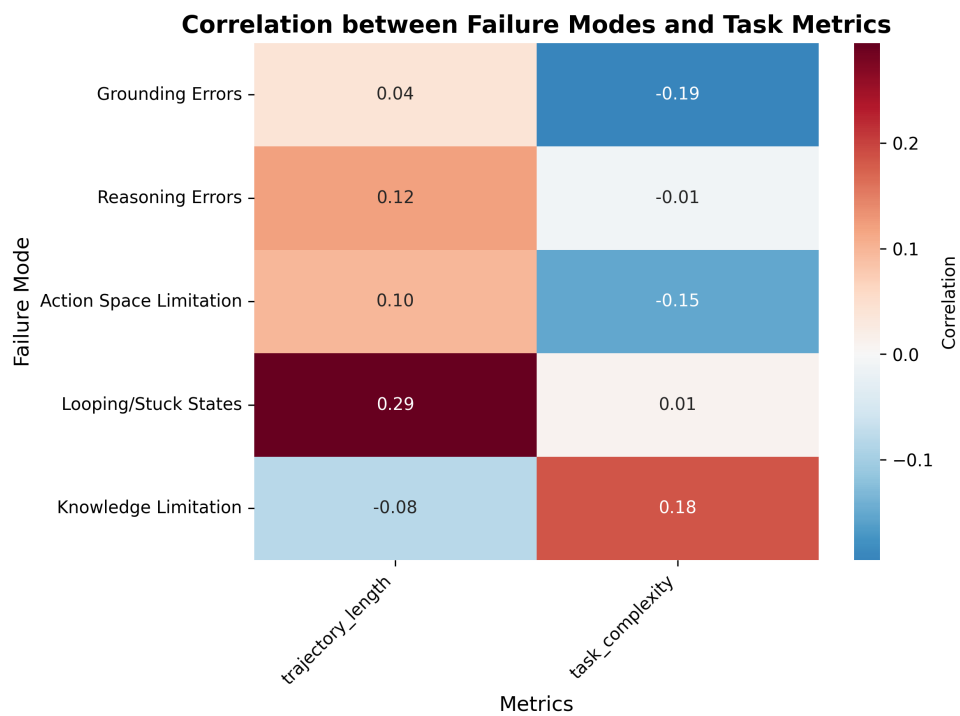


Figure 17: Correlation Matrix of Failure Modes and Task Metrics. Heatmap displaying the Pearson correlation coefficients between specific failure modes and task metrics (Trajectory Length and Task Complexity).

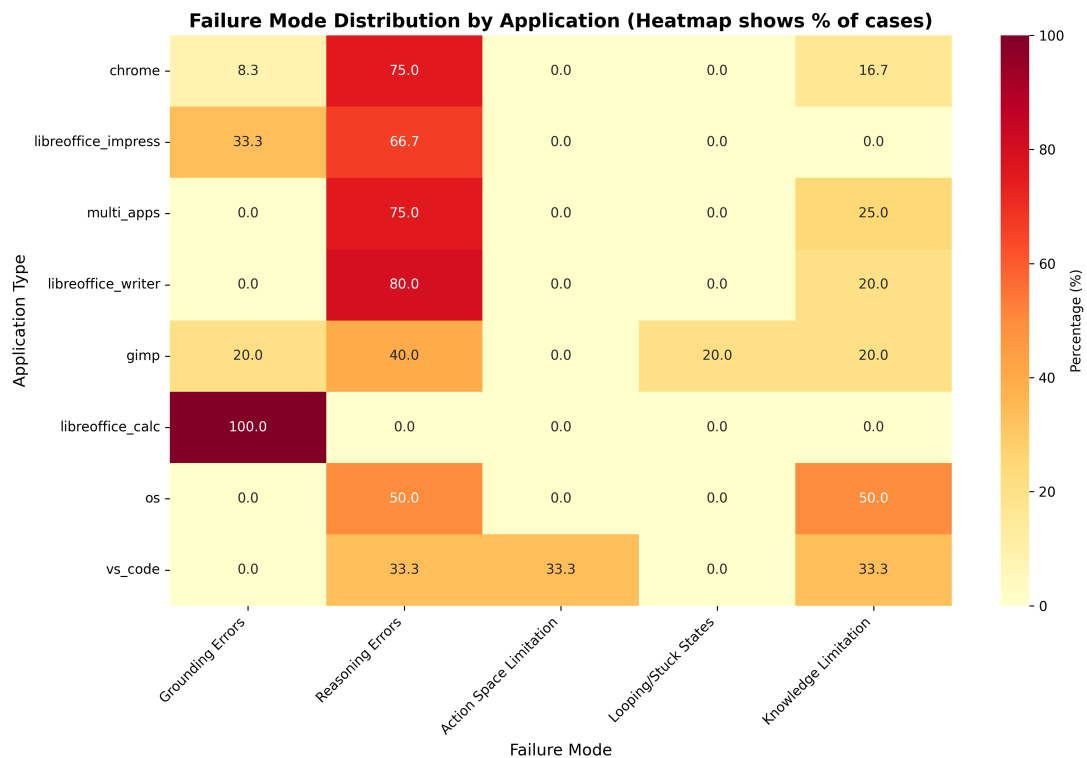


Figure 18: Failure Mode Distribution by Application Type. Heatmap showing the percentage of failure modes across different GUI applications.