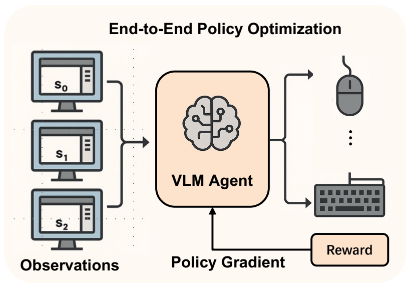
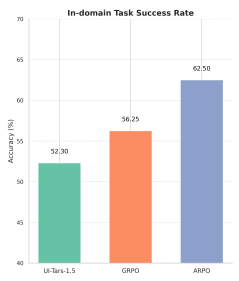
ARPO: End-to-End Policy Optimization for GUI Agents with Experience Replay

Fanbin Lu[1] Zhisheng Zhong[1] Shu Liu[2] Chi-Wing Fu[1] Jiaya Jia[2, 3]

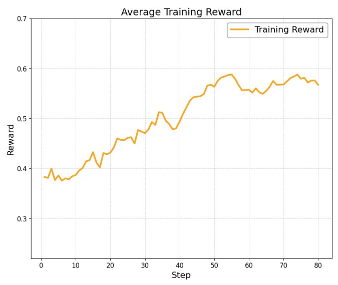
[1]CUHK [2]SmartMore [3]HKUST



(a) End-to-end GUI agent.



(b) Task completion rate.



(c) Training rewards.

Figure 1: **A**gentic **R**eplay **P**olicy **O**ptimization (ARPO) enables effective end-to-end policy optimization for GUI agents. (a) Our vision-language agent processes long-horizon visual observations and interaction histories to generate sequential actions and receive policy gradients from sparse, delayed rewards. (b) ARPO significantly boosts in-domain task success rates compared to baseline and GRPO-only training. (c) Average training reward steadily increases, demonstrating improved policy learning and sample efficiency in complex GUI environments.

Abstract

Training large language models (LLMs) as interactive agents for controlling graphical user interfaces (GUIs) presents a unique challenge to optimize long-horizon action sequences with multimodal feedback from complex environments. While recent works have advanced multi-turn reinforcement learning (RL) for reasoning and tool-using capabilities in LLMs, their application to GUI-based agents remains relatively underexplored due to the difficulty of sparse rewards, delayed feedback, and high rollout costs. In this paper, we investigate **end-to-end policy optimization** for vision-language-based GUI agents with the aim of improving performance on complex, long-horizon computer tasks. We propose **A**gentic **R**eplay **P**olicy **O**ptimization (**ARPO**), an end-to-end RL approach that augments Group Relative Policy Optimization (GRPO) with a replay buffer to reuse the successful experience across training iterations. To further stabilize the training process, we propose a task selection strategy that filters tasks based on baseline agent performance, allowing the agent to focus on learning from informative interactions. Additionally, we compare ARPO with offline preference optimization approaches, highlighting the advantages of policy-based methods in GUI environments. Experiments on the OSWorld benchmark demonstrate that ARPO achieves competitive results, establishing a new performance baseline for LLM-based GUI agents trained via reinforcement learning. Our findings underscore the effectiveness of reinforcement learning for training multi-turn, vision-language GUI agents capable of managing complex real-world UI interactions. Codes and models:https://github.com/dvlab-research/ARPO.git.

1 Introduction

Among various agent types, GUI agents that interact with the computer screen through vision-based perception and action have been of long-standing interest [2, 16, 4]. Most prior work relies on supervised fine-tuning (SFT) on large-scale trajectory data. These agents are typically trained through SFT on large-scale trajectory datasets, where the model learns to imitate human behavior by predicting the next action based on the current screenshot and interaction history. However, these agents lack the ability to self-correct and suffer from error accumulation in the working trajectory.

To address these limitations, we explore reinforcement learning (RL) in GUI agent training. In contrast to single-turn RL or static reward optimization, we adopt Group Relative Policy Optimization (GRPO) [5], a recent variant of PPO [18] that eliminates the need for a value function and estimates token-level advantages from group-wise reward normalization. GRPO has demonstrated promising results in mathematical reasoning [19] and tool-use agent [15]. It is a natural fit for training vision-language agents due to its ability to handle long sequences and multiple modalities with improved efficiency.

This paper tackles the challenge of end-to-end policy optimization for GUI agents, with a particular focus on multi-turn, multi-modal agent design, see Fig. 1a. Our goal is to maximize the rule-based reward from the environment over entire trajectories using GRPO. However, GUI environments typically offer sparse and delayed reward signals: agents receive feedback only upon task completion, and many complex tasks may yield no reward at all during early training phases. Moreover, the cost of rollouts in real desktop environments is non-trivial. GUI interaction involves operating system-level delays, which significantly slow down the data collection process. To overcome these obstacles, we develop a scalable distributed rollout system that enables parallel interaction with real desktop environments, such as OSWorld [25]. By batching inference across environments, we reduce latency and make efficient use of GPU resources, thus facilitating rollout collection at scale.

To further enhance training stability and sample efficiency, we introduce a task selection strategy that filters for those capable of producing successful rollouts under baseline agents. This curated subset enhances signal quality during early training and accelerates convergence. We further introduce an experience replay buffer tailored to GUI agent learning. This buffer stores successful trajectories on a per-task basis and dynamically injects them into GRPO training groups when all sampled rollouts in a group fail. The inclusion of at least one high-reward trajectory within each group ensures meaningful reward variance, which is critical for computing token-level advantages.

We conduct extensive evaluations on the OSWorld benchmark and observe that reinforcement learning effectively improves agent performance. We also find an interesting fact that RL training delivers strong gains on in-domain tasks, but hardly benefits out-of-domain agentic tasks.

Our contributions are summarized as follows:

- We propose an end-to-end policy optimization approach for training a GUI agent in challenging multi-turn, multi-modal environments using GRPO.
- We demonstrate that careful selection of training tasks is critical for maintaining reward diversity and ensuring stable policy optimization.
- We propose an experience replay buffer that retains successful trajectories, enhancing sample efficiency and stabilizing training in sparse-reward settings.
- We find that reinforcement learning substantially improves agent performance on in-domain tasks, while offering moderate generalization improvements to out-of-domain agentic tasks.

2 Related Works

GUI Agents. Recent advances in multimodal models have led to significant progress in GUI and web-based automation. SeeClick [2] and ScreenAgent [14] utilize large vision-language models (VLMs) with visual input processing to perform interactive tasks on user interfaces. Building on this, OmniAct [9] introduces a benchmark that focuses on generating executable actions from visually grounded natural language instructions. CogAgent [6] and UI-Tars [16] extend pretraining with large-scale web and desktop interface data, enhancing screen understanding and agent behavior. GUI-R1 [24] explores reinforcement learning to improve UI grounding in VLM-based agents. However, directly optimizing policy models for GUI agents in an end-to-end policy optimization way remains unexplored in current research.

Reinforcement Learning for Agents. Rule-based reinforcement learning (RL) has proven effective in fine-tuning large language models (LLMs) across a range of domains. OpenAI's o1 [8] and DeepSeek-R1 [5] demonstrate strong performance in tasks such as mathematical reasoning [19], code generation [11], and multi-modal inference [7, 12] through structured reward signals. ToolRL [15] extends this paradigm by introducing RL-based training for LLM agents that interact with external tools. Sweet-RL [29] introduces a multi-turn DPO framework to enhance long-horizon language agent behaviors. RAGEN [21] further advances multi-turn RL by applying it in live, rule-based environments for self-evolving agent training.

Despite these advancements, most existing work focuses on symbolic tasks or static tool use. Applying reinforcement learning to vision-language agents operating in dynamic, multimodal GUI environments remains a challenging task. In particular, this work aims to leverage rule-based rewards from live desktop environments for end-to-end policy optimization in multi-turn GUI agents.

3 Method

In this section, we first provide a brief introduction to the preliminaries of Group Relative Policy Optimization (GRPO). Then, we describe the architecture and training procedure of our GUI agent. The agent builds upon vision-language models (VLMs), enhanced with longer context windows and longer image, action chains. These modifications are essential for training complex GUI tasks with end-to-end reinforcement learning algorithms like GRPO.

3.1 Preliminary

Group Relative Policy Optimization (GRPO) [19] is a reinforcement learning algorithm designed to optimize language models efficiently without requiring an explicit value function or critic. GRPO modifies the standard Proximal Policy Optimization (PPO) objective by computing token-level advantages based on group-normalized rewards, making it particularly suitable for LLMs.

Given a batch of G responses $\{o_i\}_{i=1}^{G}$ from a query q, each consisting of a sequence of tokens $o_i = (o_i(1), ..., o_i(T))$, the GRPO objective is defined as:

$$J_{\text{GRPO}}(\theta) = \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min\left[\frac{\pi_\theta(o_i(t)|o_{i,<t})}{\pi_{\text{old}}(o_i(t)|o_{i,<t})}\hat{A}_{i,t},\ \text{clip}\left(\frac{\pi_\theta(o_i(t)|o_{i,<t})}{\pi_{\text{old}}(o_i(t)|o_{i,<t})}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right],$$

where $\hat{A}_{i,t}$ is the group-normalized advantage for token t in response o_i, computed as:

$$\hat{A}_{i,t} = \frac{r_i - \mu}{\sigma}, \quad \text{with } r_i \text{ the total reward of } o_i,$$

and μ, σ, the mean and standard deviation of rewards in the group.

3.2 Multi-turn GUI Agent

Unlike single-turn reinforcement learning, GUI agents are required to perform multi-turn reasoning and decision-making, interacting with dynamic environments that provide visual feedback. We adopt a Markov Decision Process (MDP) framework, where each agent trajectory comprises a sequence of screenshot observations s_t, mouse and keyboard actions a_t, and a scalar reward r in the end of the trajectory. The agent policy π_θ is optimized to maximize the rewards:

$$\tau = \{s_t, a_t\}_{t=0,1,\cdots,T-1}, \text{ where } a_t \sim P_\theta\big(\{s_i, a_i\}_{i<t}\big). \tag{1}$$

Our GUI agent is built upon the UI-Tars [16] framework and the Qwen2.5-VL architecture [1]. To predict the next action a_t the model tokenizes the entire history of screenshots and corresponding actions into the input context of the VLM model.

Our design results in a VLM model with at most 15 images input and $64K$ model context length to correctly process an entire GUI trajectory with 1080P resolution. Unlike prior short-context

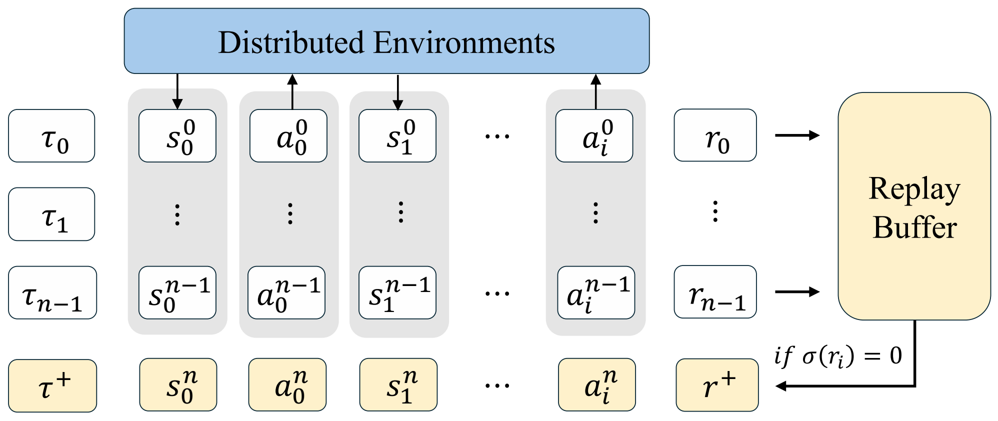


Figure 2: Illustration of the reinforcement learning procedure for our multi-turn GUI agent. For a single task, we use n parallel environments and perform rollouts to collect trajectories and rewards $\{\tau_i, r_i\}_{i=0,1,\cdots,n-1}$ in the environments. If all the rewards are zero, we fetch a positive trajectory τ^+ from the replay buffer to avoid gradient vanishing.

GUI agents [2, 4, 26], which truncate the trajectory and only process the most recent one or two screenshots, our approach leverages the full trajectory history, enabling the model to reason over long-term dependencies and optimize performance across the entire interaction sequence.

CoT for GUI Agents

To enhance the reasoning capabilities of VLM agents, we integrate the **Chain-of-Thought (CoT)** prompting technique [22] into our action generation process. Each action a_t is composed of:

- A **thinking part**, which represents the agent's internal reasoning.
- A **solution part**, which executes the resulting action.

This design allows the agent to perform more accurate and interpretable decision-making.

Action Space Definition

Our GUI agent adopts the action space defined in UI-Tars [16], including the following primitive operations:

- `LEFT_CLICK`, `RIGHT_CLICK`, `SCROLL`
- `TYPE_TEXT`, `PRESS_HOTKEY`

In addition to these, several meta-actions are used to manage the agent's workflow:

- `WAIT`: Pause and observe the environment.
- `FINISH`: Successfully complete the task.
- `FAIL`: Indicate task failure.
- `CALL_USER`: Request human intervention.

3.3 Distributed Trajectory Rollout

Training GUI agents through reinforcement learning requires scalable and efficient trajectory collection across rich, interactive desktop environments. To meet this need, we design a distributed trajectory rollout strategy tailored for parallel interaction with live environments, such as OSWorld [25].

We establish a set of rollout workers. Each worker consists of an interactive environment paired with a GUI agent that maintains a history of screenshots and corresponding actions, denoted as (s_t, a_t). Each rollout worker continuously captures screenshots of the current GUI environment and transmits them to a centralized language model inference server powered by VLLM [10]. The policy model processes these batched visual observations in parallel, predicting the next action for all environments simultaneously.

Unlike math [19] or tool-use [15] environments, interaction with live GUIs like OSWorld incurs non-trivial latency due to OS-level delays. Parallel trajectory rollout allows for efficient utilization of GPU resources on the inference server and minimizes the per-step decision latency.

3.4 End-to-End Policy Optimization with GRPO

We adopt GRPO [19] as our reinforcement learning algorithm to train vision-based GUI agents. GRPO eliminates the need for a value function by leveraging group-wise reward normalization to compute token-level advantages. This property makes it well-suited for training VLM agents with multiple image inputs and extended context length.

Reward Design. To effectively guide policy optimization, we design a structured reward function incorporating both task-level success and action format correctness.

- **Trajectory Reward:** For each task, we have a scalar trajectory-level reward r_t, based on task completion. A reward of $r_t = 1$ is assigned if the agent successfully completes the task as defined by the OSWorld [25], and $r_i = 0$ otherwise. This binary reward provides a high-level training signal to encourage successful multi-turn planning and execution.
- **Action Format Reward:** During rollout, each response from the VLM agent is parsed into discrete actions. If a response fails to conform to the required action schema and cannot be parsed, we assign a penalty of $r_f = -1$. This encourages the model to generate syntactically valid and executable actions.

Training Objective. We treat GUI interaction as a multi-turn MDP, where the agent observes a sequence of screenshots s_t and generates actions a_t to complete a task instruction $x \in \mathcal{D}$. The trajectory $\tau = (s_0, a_0, \ldots, a_n)$ is encoded by a VLM agent with extended context, enabling long-horizon reasoning over multiple steps and observations. Our training objective is to maximize the expected reward over tasks and trajectories:

$$\max_{\theta} \; \mathbb{E}_{x \sim \mathcal{D},\, \tau \sim \pi_\theta}[r_t(x, \tau) + r_f(x, \tau)]. \tag{2}$$

We optimize this objective using GRPO, which estimates token-level advantages via group-normalized trajectory rewards, allowing efficient and scalable training without a value function.

Valuable Tasks Selection for GRPO. Despite recent progress in the variants of GRPO [28], the task of training GUI agents remains difficult, particularly due to the sparse reward signals associated with complex desktop environments like OSWorld [25]. Many tasks in this benchmark are not reliably solvable by current state-of-the-art agents [16, 4], even when given multiple attempts. As a result, these tasks generate limited feedback during rollouts, which can hinder effective policy optimization training for GRPO.

To improve the sampling efficiency, we introduce a task filtering procedure to identify a subset of "valuable" tasks, those capable of producing successful trajectories under a baseline agent. Specifically, we evaluate each task in OSWorld using the UI-Tars-1.5 model, performing 16 rollouts per task. A task is retained in the GRPO training set if the agent completes it successfully in at least one of these attempts. This method yields a curated set of 128 tasks that are more amenable to early-stage learning, allowing the policy optimization to benefit from informative reward signals.

3.5 Experience Replay Buffer

Dynamic Sampling [28] has been proposed to improve the sample efficiency of GRPO by removing training groups in which all rewards are uniform. In such cases, the computed token-level advantages are zero across the group, resulting in vanishing gradients and slowed convergence. However, this strategy becomes less effective in GUI interaction settings due to two primary challenges: the high cost of obtaining trajectories and the infrequency of successful rollouts.

Unlike mathematical reasoning tasks, which typically follow well-defined logical chains, GUI-based tasks sometimes require a certain amount of exploratory interactions with the environment, resulting in sparse reward signals. Therefore, successful trajectories are rare but especially informative. Preserving and reusing them is critical for the training progress.

To address this, we introduce an experience replay buffer that caches successful trajectories on a per-task basis. During training, if an entire GRPO training group consists of only failed trajectories (*i.e.*, all with zero reward), we randomly replace one of them with a previously stored successful trajectory from the buffer for the corresponding task. This guarantees that, as long as the agent has successfully completed a task once, its training group in the later training process will include at least one rollout with a non-zero reward signal, as illustrated in Fig. 2. The buffer is updated dynamically during rollout. To prevent the stored samples from diverging too significantly from the current policy, we impose a fixed-size limit on the buffer and evict the oldest entries when full.

4 Experiments

4.1 Implementation Details

Training Details. We use the 7B UI-Tars-1.5 model [16] as the base and conduct training using the VERL framework [20]. For trajectory rollout, we set up 256 parallel virtual environments and the rollout number for each task is 8. A total of 128 tasks are sampled from the OSWorld benchmark, according to the strategy described in Sec. 3.4, and training is performed over 15 epochs. Rollouts are conducted with a batch size of 32 and a temperature of 1.0 to encourage exploration. For policy optimization, we use the AdamW optimizer [13] with a learning rate of 1×10^{-6} and a mini-batch size of 8 per device. The gradient accumulation number is 4. Following DAPO [28], we set the clipping parameters to $\epsilon_{\text{low}} = 0.2$ and $\epsilon_{\text{high}} = 0.3$ to balance exploration and exploitation. During evaluation, the temperature is lowered to 0.6 for more stable performance. We remove the KL divergence loss to remove the need for the reference model.

Datasets and Benchmarks. We evaluate our method on the OSWorld benchmark [25], a recently proposed real-computer environment designed for evaluating multimodal agents on open-ended GUI tasks. OSWorld contains 369 tasks across diverse domains such as office productivity, web browsing, system management, and multi-app workflows. Each task is executed within virtual machines using real applications and evaluated via execution-based scripts. The benchmark supports full GUI interaction with mouse and keyboard actions, enabling rigorous assessment of multi-turn vision-based agents in realistic desktop environments.

Evaluation Metrics. We follow the standard rule-based evaluation protocol defined in OSWorld [25]. Each agent trajectory receives a scalar reward between 0 and 1.0 from the environment. We notice that previous works [16] replace the last action with a FAIL action when the maximum step number is reached in a rollout. While this approach may prevent unstable or endlessly running behaviors during evaluation, it will hack the rewards of the real impossible tasks defined in the benchmark. To provide a more accurate assessment of agent capabilities for RL, we introduce a stricter evaluation protocol that prohibits final action replacement, denoted as **OSWorld Hard**.

4.2 Experimental Results

We evaluate the performance of our ARPO method on the OSWorld benchmark [25], comparing it against several recent GUI agents. As shown in Table 1, our approach achieves the highest success rates across both evaluation settings. Specifically, applying ARPO to the UI-Tars-1.5 base model results in a success rate of 29.9% on the standard OSWorld setting and 23.8% on the stricter OSWorld Hard variant—improving upon the original UI-Tars-1.5 model by 6.4% and 5.6%, respectively. These results highlight the effectiveness of reinforcement learning with GRPO and structured experience replay in enhancing multi-turn GUI decision-making. Additionally, ARPO shows consistent gains across earlier model versions; for example, UI-Tars-7B-DPO improves from 15.6% to 20.4% with ARPO. All the models are tested with a maximum step number limit of 15 for a single trajectory.

Table 1: OSWorld evaluation performance for GUI Agents. All models are evaluated at a maximum execution length of 15. We provide numerical results for two metrics: **OSWorld** and **OSWorld Hard**.

Model	**GPT-4o**	**OSWorld**	**OSWorld Hard**
Aria-UI [27]	✓	15.2%	-
Aguvis-7B [26]	✓	14.8%	-
Aguvis-72B [26]	✓	17.0%	-
OS-Atlas-7B [23]	✓	14.6%	-
UI-Tars-7B-DPO		15.6%	11.3%
UI-Tars-7B-DPO + GRPO		18.3% (+2.7%)	16.4% (+5.1%)
UI-Tars-7B-DPO + ARPO		**20.4%** **(+4.8%)**	**18.0%** **(+6.7%)**
UI-Tars-7B-1.5		23.5%	18.2%
UI-Tars-7B-1.5 + GPRO		26.0% (+2.5%)	20.9% (+2.7%)
UI-Tars-7B-1.5 + ARPO		**29.9%** **(+6.4%)**	**23.8%** **(+5.6%)**

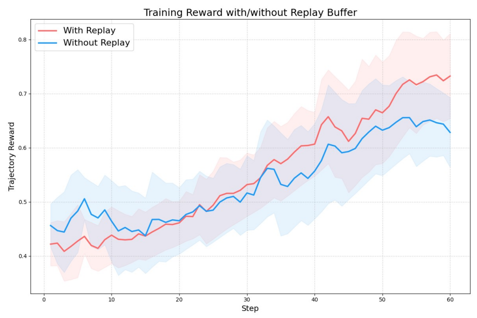


Figure 3: Ablation study of the replay buffer.

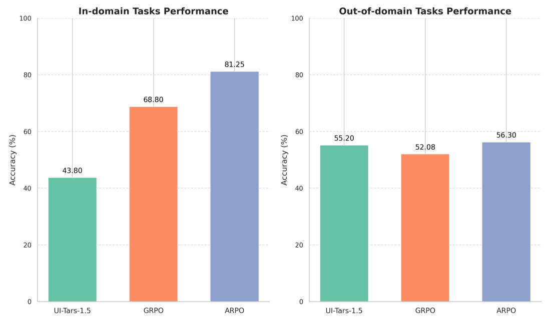


Figure 4: Ablation study for GRPO and ARPO in-domain and out-of-domain RL training tasks.

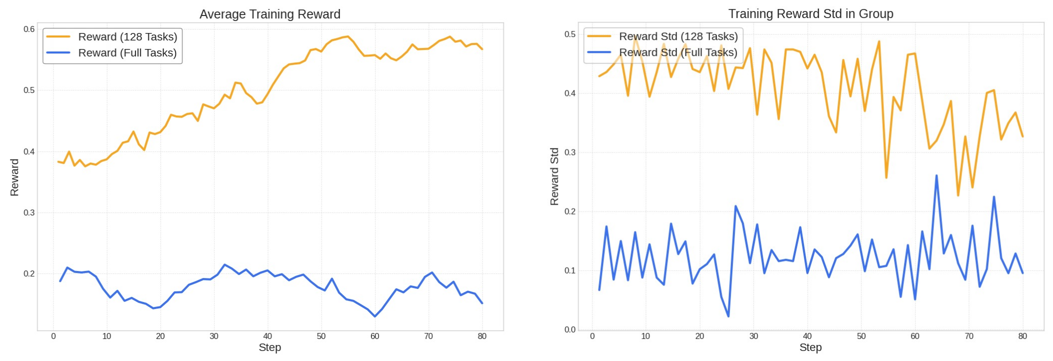
4.3 Ablation on the Replay Buffer

We evaluate the impact of the experience replay buffer by comparing training trajectories with and without its use, as shown in Fig. 3. The model equipped with the replay buffer begins to outperform the baseline around Step 30 and maintains a consistent advantage throughout the remainder of training. This improvement is attributed to the buffer's ability to retain successful trajectories. This gain is largely due to the buffer's ability to retain high-reward trajectories from earlier stages, which serve as strong learning signals in later updates. By maintaining reward diversity within GRPO groups and non-zero advantages, the replay buffer supports more stable optimization and accelerates convergence. By the end of training, the model with replay achieves a higher average trajectory reward (0.75 vs. 0.65), demonstrating that leveraging past successes substantially improves both sample efficiency and overall policy performance in sparse-reward GUI environments.

The benefits of the replay buffer extend beyond reward curves. As shown in Fig. 4, the in-domain task success rate climbs from 68.8% with GRPO to 81.25% with ARPO, a 12.5% absolute improvement. This substantial gain highlights the replay buffer's critical role in enhancing policy generalization and downstream performance.

4.4 Does RL training generalize well to OOD GUI agent tasks?

To assess the generalization ability of RL training, we evaluate model performance on both in-domain and out-of-domain (OOD) tasks. Specifically, we select 32 tasks from the training task set for reinforcement learning, using the remaining 96 as OOD tasks. As shown in Fig. 4, reinforcement learning substantially improves in-domain accuracy: GRPO achieves 68.8% and ARPO reaches 81.25%, compared to 43.8% for the base UI-Tars-1.5 model. However, on OOD tasks, gains are more modest. UI-Tars-1.5 achieves 55.2%, while GRPO slightly underperforms at 52.08%. ARPO, however, recovers generalization capability, scoring 56.3%, slightly above the base model, indicating that structured trajectory grouping and replay mitigate overfitting. Overall, while reinforcement



(a) Training rewards.

(b) Reward variance during RL training.

Figure 5: Training performance comparison for RL training with selected subset and full set.

learning effectively improves the in-domain success rate of VLM agents, strong generalization still depends on broader task diversity, carefully designed reward signals, and larger-scale training compute.

4.5 Valuable Task Selection for GRPO Training

As outlined in Sec. 3.4, we adopt a task selection strategy for GRPO training by filtering out tasks that consistently fail to provide meaningful reward signals. To evaluate the impact of this approach, we conduct an ablation study comparing GRPO performance when trained on a curated subset of 128 valuable tasks versus the full task set. As illustrated in Fig. 5a, training on the selected subset leads to significantly higher average trajectory rewards and faster convergence speed from the early stages of training.

Fig. 5b shows that the standard deviation of rewards within GRPO groups is consistently higher when training on the curated task set. This increased variance is critical for GRPO, which relies on within-group reward diversity to compute token-level advantages. In contrast, training on the full task set results in flatter reward distributions with reduced variance.

4.6 Comparison with Offline Preference Optimization

In Fig. 6, we compare the performance between GRPO and offline preference optimization algorithms. For a fair comparison, all methods are trained on the same task set with an equal number of rollouts. We compare GRPO with reject sampling, DPO [17], and KTO [3]. For reject sampling, we take only the positive trajectory for SFT training. For DPO, we randomly sample a positive and a negative trajectory per task to create paired training data. For KTO, we threshold the scalar rewards at 0.5 to generate binary labels for training.

ARPO achieves the highest score (27.3%), followed by GRPO (26.0%), both outperforming preference-based methods by a significant margin. Among the baselines, KTO performs best (24.6%), while DPO and Reject Sampling lag behind at 22.4% and 21.8% , respectively. These results suggest that direct trajectory-level optimization with rule-based rewards provides stronger learning signals than offline preference modeling. The added experience replay in ARPO further enhances stability and sample efficiency in sparse-reward GUI settings.

4.7 Rollout Efficiency Analysis

Fig. 7 shows that increasing the number of parallel environments significantly improves training efficiency. We show the rollout time for a single batch of trajectories (in minutes) and the total time to sample all trajectories in an epoch (in hours), respectively. As the batch size increases, although the rollout time per batch grows from 3 minutes (8 environments) to 19 minutes (256 environments), the total time per epoch drops sharply from over 6 hours to just around 1.2 hours.

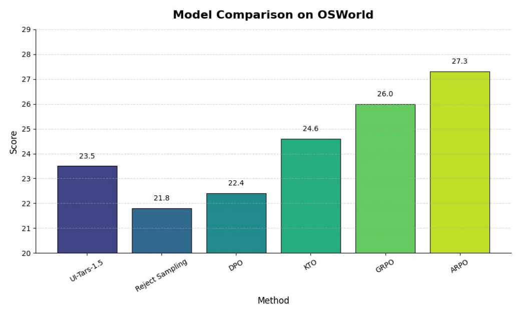


Figure 6: Comparisons with offline preference optimization methods.

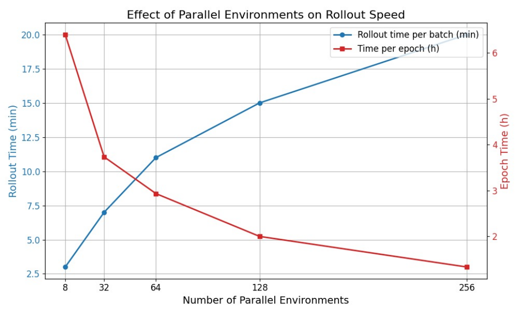
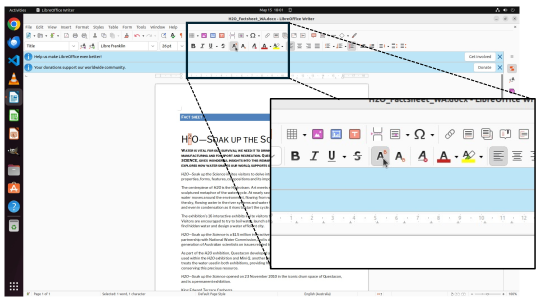


Figure 7: Rollout Efficiency.

Task: Help me change the 2 in "H2O" to a subscript.



Thought: I noticed that there is a subscript feature in the toolbar, which is exactly what I need. To change the "2" in H2O to a subscript, I first need to select that "2." So now, I will use my mouse to click and drag from the position of the first "H" all the way to where the "O" is located. This way, I can accurately select just that middle number.
Action #1: drag(start_box='(689,493)', end_box='(710,495)')

Thought: I just selected the "2" in H2O, and now I need to locate the subscript button on the toolbar. It should be in that row of formatting tools, and clicking it will change this number into a subscript.
Action #2: click(start_box='(863,149)')

Thought: I just accidentally clicked the wrong button and changed the "2" in H2O to a superscript, which is not what we wanted. Let me press Ctrl+Z to undo that mistake and start over.
Action #3: hotkey(key='ctrl z')

Figure 8: Trajectory showing the ARPO-trained agent correcting a misclick error.

This speedup stems from two main factors: (1) Larger batches allow the VLLM server to perform more efficient GPU inference, and (2) OS-level delays in GUI environments are overlapped for all parallel environments. As a result, scaling to 256 environments enables high-throughput rollouts, making RL training in real desktop settings much more practical.

4.8 Qualitative Analysis: Self-Correction Behavior in GUI Agent

Figure 8 illustrates a trajectory where the ARPO-trained agent demonstrates self-corrective behavior. Initially, it selects the superscript button instead of the subscript button. It realizes the mistake by observing the current screen and decides to use the Ctrl-Z hotkey to revert the previous operation. Notably, the success rate for the specific before and after ARPO are 25% vs. 62.5%.

5 Conclusion

In this work, we present a reinforcement learning approach for training GUI agents using vision-language models enhanced with longer input context and multi-turn, multi-modal screenshot processing. By introducing ARPO, a variant of GRPO tailored for GUI agents, we demonstrate that rule-based reward signals can effectively guide end-to-end policy optimization in complex GUI environments. Our experiments show that careful task selection significantly improves learning stability and reward variance.

This study highlights the potential of combining multimodal understanding with reinforcement learning to enable more adaptive and capable GUI agents. Future directions include expanding the task set to cover a broader range of real-world applications, extending the context length of agents further to support more sophisticated trial-and-error behaviors, and investigating the use of learned reward models to autonomously evaluate trajectories, reducing reliance on manually crafted reward functions.

References

[1] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025. 3

[2] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*, 2024. 2, 4

[3] Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024. 8

[4] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024. 2, 4, 5

[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. 2, 3

[6] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290, 2024. 2

[7] Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint arXiv:2503.06749*, 2025. 3

[8] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024. 3

[9] Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*, 2024. 2

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023. 5

[11] Jiawei Liu and Lingming Zhang. Code-r1: Reproducing r1 for code with reliable rewards. *arXiv preprint arXiv:2503.18470*, 2025. 3

[12] Yuqi Liu, Bohao Peng, Zhisheng Zhong, Zihao Yue, Fanbin Lu, Bei Yu, and Jiaya Jia. Seg-zero: Reasoning-chain guided segmentation via cognitive reinforcement. *arXiv preprint arXiv:2503.06520*, 2025. 3

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6

[14] Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. Screenagent: A vision language model-driven computer control agent. *arXiv preprint arXiv:2402.07945*, 2024. 2

[15] Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025. 2, 3, 5

[16] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025. 2, 3, 4, 5, 6

[17] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023. 8

[18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2

[19] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. *URL https://arxiv. org/abs/2402.03300*. 2, 3, 5

[20] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*, 2024. 6

[21] Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025. 3

[22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022. 4

[23] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024. 7

[24] Xiaobo Xia and Run Luo. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025. 2

[25] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024. 2, 4, 5, 6

[26] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024. 4, 7

[27] Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. Aria-ui: Visual grounding for gui instructions. *arXiv preprint arXiv:2412.16256*, 2024. 7

[28] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. 5, 6

[29] Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *arXiv preprint arXiv:2503.15478*, 2025. 3