STEP: Success-Rate-Aware Trajectory-Efficient Policy Optimization

Yuhan Chen[1] **Yuxuan Liu**[2] **Long Zhang**[3] **Pengzhi Gao**[1] **Jian Luan**[1] **Wei Liu**[1*]
[1]MiLM Plus, Xiaomi Inc. [2]Renmin University of China [3]Wuhan University
{chenyuhan5, gaopengzhi, luanjian, liuwei40}@xiaomi.com
yuxuanliu@ruc.edu.cn
zlongooo@whu.edu.cn

Abstract

Multi-turn interaction remains challenging for online reinforcement learning. Current GRPO-based methods—either at the trajectory level or the step level—still suffer from fundamental challenges in multi-turn settings: they allocate sampling uniformly across tasks regardless of difficulty, propagate misleading learning signals that penalize correct intermediate actions in failed trajectories, and incur high sample-collection costs under long-horizon environments. Step-level variants (e.g., GIGPO) mitigate some interaction-cost constraints by decomposing trajectories, yet they retain GRPO's sampling imbalance and still struggle with heterogeneous multi-turn tasks. To address these issues, we propose *STEP* (Success-rate-aware Trajectory-Efficient Policy Optimization), a framework that dynamically allocates sampling based on per-task success rates and performs fine-grained step-level optimization. *STEP* maintains a smoothed success-rate record to guide adaptive trajectory resampling, allocating more effort to harder tasks. It then computes success-rate-weighted advantages and decomposes trajectories into step-level samples, followed by a step-level GRPO augmentation that strengthens updates on low-success tasks. Experiments on OSWorld and AndroidWorld show that *STEP* substantially improves sample efficiency and training stability over both trajectory-level and existing step-level GRPO variants, converging faster and generalizing better under the same sampling budget.

1 Introduction

Large language models (LLMs) have been increasingly adopted as agents for multi-turn decision-making, where they must reason, plan, and act over extended interactions with delayed and sparse rewards. Such applications include program synthesis (Zhang et al., 2024), interactive gameplay (Narasimhan et al., 2015), robotic control (Brohan et al., 2023) and GUI automation (Qin et al., 2025; Ye et al., 2025; Huang et al., 2025; Liu et al., 2026). To improve these agents' adaptability, reinforcement learning (RL) has become a key paradigm for online policy optimization through feedback-driven interaction.

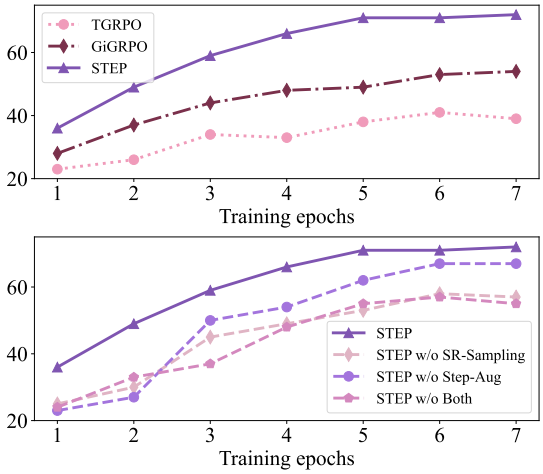


Figure 1: Number of tasks in the OSWorld training subset (128 tasks) that achieve a success rate above 60% during training across different methods. We report results for trajectory-level GRPO (T-GRPO), GIGPO, our proposed method (*STEP*) and three ablation variants of *STEP*. Further details are provided in Section 6.2.

Among RL-based methods, Group Relative Policy Optimization (GRPO) (Shao et al., 2024) has been widely adopted for its efficiency and scalability. In multi-turn settings, GRPO is typically applied either at the trajectory level—treating the full sequence of decisions and feedback in an episode as a single training sample—or in more recent step-level variants that decompose trajectories to enable training under longer interaction horizons under de-

* Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 30681–30692

vice constraints. Despite their differences, both formulations share the same underlying optimization mechanism and thus inherit several fundamental challenges in multi-turn reinforcement learning:

(1) **Uniform sampling across tasks.** GRPO allocates equal sampling effort to all tasks, regardless of their difficulty or success rate. In multi-turn settings, where task complexity varies widely, this uniform allocation leads to inefficient use of the sampling budget: the agent repeatedly trains on simple, already-solved tasks while complex, low-success tasks—those that provide the most informative learning signals—receive insufficient attention.

(2) **Inaccurate credit assignment.** Multi-turn tasks typically provide only a final outcome reward after the entire trajectory is completed, and this terminal feedback must be propagated back to all intermediate steps. As a result, correct steps within an otherwise failed episode are penalized, leading to inaccurate gradients or reward signals.

(3) **Inefficient sample collection.** Compared with single-turn scenarios, multi-turn tasks require continuous interaction with the environment. Each rollout depends heavily on environment latency, and multiple runs are needed to collect one trajectory, making sample collection highly inefficient.

To address these challenges, we propose *STEP*(Success-rate-aware Trajectory-Efficient Policy optimization), a framework that dynamically allocates sampling and learning effort based on per-task success rates and performs fine-grained, step-level optimization. *STEP* maintains smoothed success-rate records to guide adaptive trajectory resampling, decomposes only the successful trajectories into step-level samples for success-rate-weighted credit assignment, and applies step-level GRPO augmentation for low-success tasks. This design enables both efficient use of sampling budgets and stable, high-quality learning, leading to faster convergence and better generalization in multi-turn RL scenarios.

We evaluate our approach on two general-purpose GUI benchmarks, OSWorld and AndroidWorld, which feature complex, multi-turn interaction environments suitable for comprehensive assessment. As depicted in Figure 1, our *STEP* consistently outperforms existing methods, achieving higher efficiency and effectiveness than trajectory-level GRPO and GIGPO.

To sum up, we make three major contributions:

(1) We systematically analyze the challenges in multi-turn reinforcement learning and provide several key insights.

(2) Based on these insights, we propose *STEP*, a step-level training framework specifically designed for multi-turn reinforcement learning.

(3) Through extensive experiments, we demonstrate the effectiveness of *STEP*, achieving improvements both in training efficiency and performance.

2 Related Work

LLM Agents for Multi-Turn Interaction Recent advances in Large Language Models (LLMs) (Yao et al., 2022; Brohan et al., 2023) have expanded their role from static language understanding to interactive agents that perceive, reason, and act in dynamic environments. Research has increasingly explored these agents across diverse domains, including embodied navigation in simulated homes (Shridhar et al., 2020; Li et al., 2024), multi-step web and mobile task execution leveraging structured pages and APIs (Hong et al., 2023; Gur et al., 2023; Furuta et al., 2023; Gou et al., 2024), and adaptive decision-making in interactive games (Narasimhan et al., 2015; Wang et al., 2024). A common goal across these studies is to enable LLMs to maintain coherent perception–reasoning–action loops over multiple turns, which requires robust contextual understanding and long-horizon planning. Some approaches (Schick et al., 2023; Wang et al., 2023; Zhang et al., 2023; Liu et al., 2025a) address this by constructing modular workflows that combine multiple components to perform complex tasks, showing potential for improved performance. More recently, methods have increasingly focused on training LLMs directly on interaction data using supervised fine-tuning (SFT) (Zhang and Zhang, 2023), or reinforcement learning (RL) (Sutton and Barto, 1998), allowing models to acquire task-relevant patterns from environmental interactions.

Reinforcement Learning for Large Language Models An early and influential application of reinforcement learning (RL) in large language models (LLMs) is RLHF (Stiennon et al., 2020; Ouyang et al., 2022), which aligns model outputs with human preferences. More recently, RL has been increasingly employed to enhance reasoning and logical deduction in LLMs, using methods such as PPO (Schulman et al., 2017), DPO (Rafailov et al., 2023), and GRPO (Shao et al., 2024). In particular, group-based RL algorithms such as GRPO, Dr. GRPO (Liu et al., 2025b), and DAPO (Yu et al.,

2025) have shown promise due to their low computational cost and efficient updates. By leveraging a group of samples from the same query, these methods estimate advantages without introducing an additional value function. They have achieved strong performance in tasks such as mathematical reasoning, search, and tool use, though these tasks are predominantly single-turn. Recent studies (Wang et al., 2025; Lu et al., 2025) have extended these approaches to multi-turn interactions by treating entire trajectories as sequences of independent steps. Some methods, such as GIGPO (Feng et al., 2025) and MobileAgentv3 (Ye et al., 2025), further decompose trajectories into steps to mitigate device constraints, enabling long-horizon training. However, these approaches still overlook the fundamental challenges inherent to multi-turn settings.

3 Preliminaries

Multi-Turn Tasks Formally, given a task Q and an initial environment state, the LLM agent interacts with the environment over multiple steps to accomplish the task. At each step t, the agent observes a state $S_t = \langle Q, H_t, I_t \rangle$, where Q denotes the task description, H_t the interaction history, and I_t the current environment observation (e.g., a screenshot). Based on S_t, the agent generates a textual response using an LLM policy π_θ, from which an action A_t is extracted and executed in the environment, yielding an immediate reward R_t. This process continues until the episode terminates or the step limit is reached, producing a trajectory:

$$\mathcal{T} = \{\mathcal{S}_1, R_1^*, \ldots, \mathcal{S}_t, R_t^*, \ldots, \mathcal{S}_T, R_T\},$$

where each state-action pair $\mathcal{S}_t := (S_t, A_t)$ corresponds to step t. Here, $*$ indicates that intermediate rewards R_t^* may be unavailable in some scenarios, and R_T denotes the final trajectory reward $R_\mathcal{T}$. The training objective then is to update the LLM policy π_θ to maximize the expected reward across tasks.

Group-based RL For a given task in multi-turn scenarios, GRPO samples a group of N trajectories $\mathcal{G}_\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_N\}$ under the old policy $\pi_{\theta_{\text{old}}}$, with each trajectory $\mathcal{T}$ associated with a final reward R. The trajectory advantages are then computed based on the reward statistics of the sampled group:

$$\text{Adv}(\mathcal{T}_i) = \frac{R(\mathcal{T}_i) - \text{mean}\big(R(\mathcal{T}_j) \mid \mathcal{T}_j \in \mathcal{G}_\mathcal{T}\big)}{\text{std}\big(R(\mathcal{T}_j) \mid \mathcal{T}_j \in \mathcal{G}_\mathcal{T}\big)}.$$

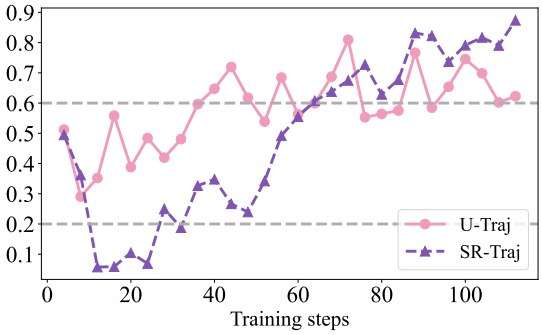


Figure 2: Proportion of high-success task trajectories over training under different sampling strategies.

By leveraging this group-based estimation, GRPO is highly memory-efficient, making it a practical and scalable choice for large-scale RL training.

4 Multi-Turn Challenges in GRPO

While effective for single-turn tasks, GRPO may encounter several challenges in complex multi-turn reasoning, including: (1) Uniform sampling across tasks, (2) Misaligned learning signals, and (3) Inefficient sample collection. In the following, we discuss each of these challenges in detail.

Uniform Sampling Across Tasks Multi-turn tasks are typically harder and require more training epochs, making efficient sampling crucial. However, GRPO allocates the same number of sampled trajectories to each task, regardless of difficulty or success rate. As a result, many successful trajectories come from simple, already-solved tasks, while harder tasks—those offering more informative signals—receive limited attention. To quantify this imbalance, we measure the proportion of high-success trajectories (success rate $> 80\%$) among all successful sampled trajectories and the result is shown in the Figure 2. We observe that this ratio with the uniform sampling strategy (U-Traj) remains consistently high throughout training, even in the early stages—averaging around 60%. Yet, as shown in Figure 1, only a small number of tasks are high-success at that stage. Consequently, GRPO exposes the agent primarily to tasks it has already mastered, limiting early learning signals, encouraging local optima, and reducing generalization.

Misaligned Learning Signals In multi-turn RL, rewards are often given only at the trajectory level, where a single incorrect action can invalidate an otherwise correct sequence. We hypothesize that such coarse feedback overlooks important inter-

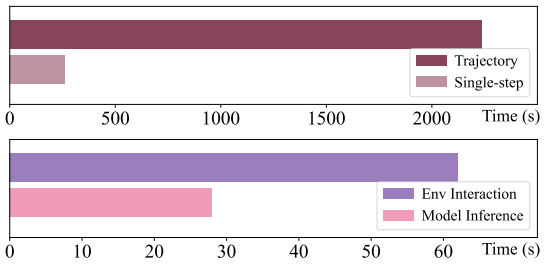


Figure 3: Analysis of sample collection efficiency. Top: wall-clock time per turn comparing environment interaction and model inference. Bottom: sampling efficiency comparing trajectory-level and single-step inference.

mediate signals. To examine this, we sampled 100 failed trajectories from OSWorld and manually compared them with successful ones. Remarkably, 78% of failed trajectories contained sub-sequences identical to those in successful trajectories, and 38.56% of individual steps were valid. Representative examples from OSWorld and AndroidWorld (Appendix A) show that correct reasoning steps are frequently penalized due to an incorrect final outcome. This indicates that trajectory-level rewards insufficiently capture partial correctness and limit effective credit assignment.

Inefficient Sample Collection Another critical challenge is the inefficiency of trajectory-level sampling. Multi-turn RL requires full trajectories for every training update, each involving multiple model calls and environment interactions. To quantify this, we measured the wall-clock time for a batch rollout of 256 trajectories and report the time per turn (top of Figure 3). The results indicate that most of the time is spent on environment interactions, while model inference accounts for only a small fraction. We further compare trajectory-level and single-step sampling. To ensure fairness, we re-infer all turns from the trajectory rollouts (3,856 turns, average length 15) in parallel. As shown in Figure 3 (bottom), collecting full trajectories is about 8.5× slower than parallel single-step sampling. The bottleneck primarily stems from expensive environment calls and the inherently sequential nature of generating full trajectories.

From these observations, we draw three insights:

(1) Uniform sampling wastes budget on already-solved tasks, hindering the learning of other valuable tasks. Sampling budgets should be dynamically adjusted based on per-task success rates.

(2) Failed trajectories produce a large amount of misleading learning signals; focusing on successful trajectories can more effectively guide the model toward correct behaviors.

(3) Increasing the number of samples through parallelization—without incurring additional environment interaction costs—can substantially enhance overall sampling efficiency.

5 Method

Building upon the insights above, we introduce our *STEP* (Success-rate-aware Trajectory-Efficient Policy optimization). As shown in the Figure 4, *STEP* adaptively adjusts both trajectory sampling and policy learning based on per-task success rates through three core components: (a) Success-rate guided trajectory sampling (SR-Traj), (b) Success-rate weighted advantage (SR-Adv), and (c) Step-level GRPO augmentation (SL-GRPO).

5.1 Success-Rate Guided Trajectory Sampling

We propose a success-rate guided trajectory sampling strategy that dynamically reallocates sampling resources based on per-task success rates.

To track success rates, we maintain two key structures:

- **Global success-rate record** $\hat{s}$, which stores the estimated success rate $\hat{s}_i$ for each task Q_i.
- **Task cache** C_Q, containing tasks with intermediate success rates ($0 < \hat{s}_i < s_0$), where s_0 is a predefined threshold.

Based on these structures, the sampling procedure is organized into two main parts: (i) Sampling Budget Reallocation and (ii) Tracking Update.

Sampling Budget Reallocation In each trajectory collection round, every task Q_i is expanded into N copies (following GRPO). For each copy[1], a **replacement function** decides whether to substitute the original task with one sampled from C_Q. The replacement probability is defined by a logistic function:

$$p_{\text{rep}}(\hat{s}_i) = \frac{1}{1 + \exp\big(-\kappa(\hat{s}_i - s_0)\big)},$$

where κ controls the sharpness of the transition around s_0. So the final $\hat{Q}_{i,j}$ for the j-th copy of task Q_i is then

$$\text{rep}_{i,j} \sim \text{Bernoulli}\big(p_{\text{rep}}(\hat{s}_i)\big),$$
$$\hat{Q}_{i,j} = \begin{cases} Q_k, & \text{if rep}_{i,j} = 1,\ Q_k \in C_Q, \\ Q_i, & \text{otherwise.} \end{cases}$$

[1]To preserve diversity and avoid forgetting, one copy is kept; replacement is applied to the remaining N-1 copies.

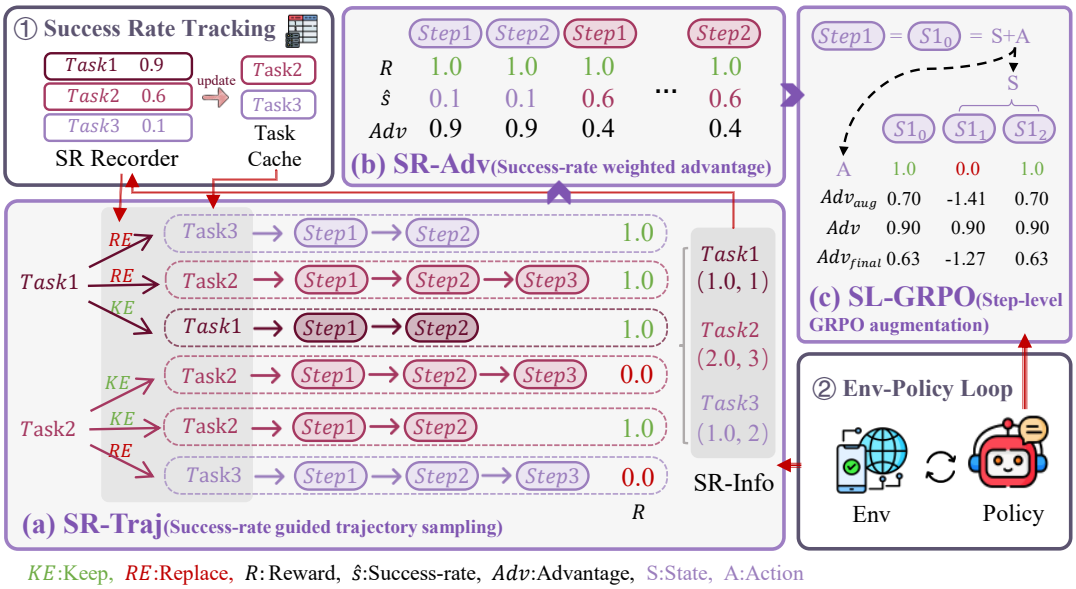


Figure 4: The framework of our *STEP*, which consists of three core components: (a) Success-rate guided trajectory sampling (SR-Traj, §5.1), which allocates the sampling budget based on dynamically updated success rates; (b) Success-rate weighted advantage (SR-Adv, §5.2), where we design a specific advantage function for *STEP*; and (c) Step-level GRPO augmentation (SL-GRPO, §5.3), which performs data augmentation without additional environment interaction costs.

After performing this sampling reallocation, we proceed to collect the trajectories. Intuitively, tasks with higher success rates are more likely to be replaced, thereby focusing sampling on tasks with lower success rates, which are expected to provide more informative signals.

Tracking Update In the end of each collection round, the success-rate record and cache are updated using a smoothed rule to stabilize estimates across varying sample sizes. Let N_i^τ and U_i^τ denote the number of collected and successful trajectories for task i in the current round τ. The update is

$$\hat{s}_i^\tau = \frac{U_i^\tau + \alpha U_i^{\tau-1}}{N_i^\tau + \alpha N_i^{\tau-1}} = \frac{U_i^\tau + \alpha \hat{s}_i^{\tau-1} N}{N_i^\tau + \alpha N},$$

$$\alpha = \begin{cases} 1 - \frac{N_i^\tau}{N}, & \text{if } N_i^\tau < N, \\ 0, & \text{otherwise,} \end{cases}$$

$$C_Q^\tau = \{Q_j \mid 0 < \hat{s}_j^\tau < s_0\}.$$

Here, we use an adaptive discount factor α that scales the influence of past estimates based on the number of collected trajectories. This helps maintain smooth and stable updates across rounds, avoiding overly aggressive changes when data are scarce. The updated $\hat{s}$ and C_Q are carried over to the next round for continued tracking and sampling.

5.2 Success-Rate-Weighted Advantage

After trajectory collection, we compute advantages for policy optimization. To reduce the influence of noisy or misleading signals from failed trajectories, we **use only successful trajectories**. We propose a success-rate-weighted advantage estimator that incorporates trajectory-level information and decomposes trajectories into individual steps to provide fine-grained learning signals.

Success-Rate-Weighted Trajectory Advantage Due to the adaptive sampling strategy introduced above, different tasks yield unequal numbers of trajectories, which makes standard group-based advantage normalization unreliable. Noting that the mean reward of a task group can serve as a proxy for its success rate, we introduce a success-rate-weighted advantage, which combines each trajectory's reward with the current task success rate. For each successful trajectory $\mathcal{T}_{i,j}$ of task Q_i, the advantage is defined as

$$\text{Adv}(\mathcal{T}_{i,j}) = (1 - \hat{s}_i) \cdot R_{\mathcal{T}_{i,j}}, \quad R_{\mathcal{T}_{i,j}} > 0,$$

where $\hat{s}_i$ denotes the smoothed success rate of task i, and $R_{\mathcal{T}_{i,j}}$ is the corresponding trajectory reward. This formulation assigns stronger learning signals to tasks with lower success rates, thereby encouraging the model to prioritize underperforming tasks.

Step-level Decomposition Each trajectory is then decomposed into step-level samples, with the same advantage assigned to all steps:

$$\text{Adv}(\mathcal{S}_{\mathcal{T},t}) = \text{Adv}(\mathcal{T}), \quad \forall t \in [1, T],$$

where $\mathcal{S}_{\mathcal{T},t}$ represents the sample at step t of the trajectory, and T is the trajectory length. Since training is performed on step-level samples, this decomposition also allows flexible organization of the history H_t within each $\mathcal{S}_{\mathcal{T},t}$ (see Section 3) using t_r past responses and t_I past observations.

5.3 Step-level GRPO Augmentation

To further improve learning efficiency and stability, we enrich the training set with valuable samples without extra environment interactions and minimal computational cost. Specifically, we selectively perform data augmentation on step samples from tasks **with low success rates** ($\hat{s}_i <= s_{\text{low}}$), which typically correspond to steps with higher trajectory advantages.

For step samples selected from low-success trajectories, each is represented as

$$\mathcal{S} = \langle S, A, \text{Adv}(\mathcal{S}) \rangle,$$

where S denotes the state, A the action, and $\text{Adv}(\mathcal{S})$ the success-rate–weighted advantage.

To augment these samples, we expand each $\mathcal{S}$ into a set of step-level variants by prompting the model with the same state S to generate $n = N/2 - 1$ alternative actions, where N is the group number we used in trajectory sampling; this setting balances diversity and efficiency without introducing additional hyperparameters. This **yields a group of candidate step samples**:

$$\mathcal{G}_{\mathcal{S}} = \{\mathcal{S}_k = \langle S, A_k \rangle \mid k = 0, \ldots, n,\ A_0 = A\}.$$

The original step $\mathcal{S}$ is thus replaced by its augmented group $\mathcal{G}_{\mathcal{S}}$, which represents localized perturbations around the original decision.

For each $\mathcal{S}_k \in \mathcal{G}_{\mathcal{S}}$, we define a step reward based on whether its action A_k **matches the reference action** A_0. Each augmented group is then evaluated to assign relative advantages among its members:

$$R(\mathcal{S}_k) = \begin{cases} 1, & \text{if } A_k \text{ matches } A_0, \\ 0, & \text{otherwise,} \end{cases}$$

$$\text{Adv}_{\text{aug}}(\mathcal{S}_k) = \frac{R(\mathcal{S}_k) - \text{mean}\big(R(\mathcal{S}_j) \mid \mathcal{S}_j \in \mathcal{G}_{\mathcal{S}}\big)}{\text{std}\big(R(\mathcal{S}_j) \mid \mathcal{S}_j \in \mathcal{G}_{\mathcal{S}}\big)}.$$

Finally, the step-level advantage used for policy optimization combines the trajectory-level credit and the local augmentation signal:

$$\text{Adv}_{\text{final}}(\mathcal{S}_k) = \text{Adv}(\mathcal{S}) \cdot \text{Adv}_{\text{aug}}(\mathcal{S}_k).$$

This step-level formulation encourages the policy to refine local action boundaries around high-value steps while maintaining consistency with the trajectory-level objective.

6 Experiments

6.1 Experiments Setups

Benchmarks. We selected two widely recognized benchmarks in the graphical user interface (GUI) domain—**OSWorld** (Xie et al., 2024) and **AndroidWorld** (Rawles et al., 2024). These benchmarks were chosen because they represent classical and challenging multi-turn interaction tasks in GUI-based environments, making them suitable for evaluating the robustness and generalization ability of our method. **OSWorld** provides a real-computer environment containing 369 tasks across diverse domains such as office productivity, web browsing, system management, and multi-application workflows. Following ARPO (Lu et al., 2025), we sample 128 tasks from the OSWorld benchmark as our training set. **AndroidWorld** offers a fully functional Android environment with reward signals for 116 programmatic tasks across 20 real-world Android applications. We select a subset of 43 tasks from these applications as the training set. Both benchmarks use rule-based rewards evaluated only after completing a trajectory, assigning 1.0 to successful executions and 0.0 otherwise.

Baselines We adopt UI-Tars-DPO-7B (Qin et al., 2025) and GUI-OWL-7B (Ye et al., 2025) as our base models, and use two baselines: (1) trajectory-level GRPO (T-GRPO) (Shao et al., 2024), which treats each full trajectory as a single sample, and (2) GIGPO (Feng et al., 2025), which computes trajectory-level advantage and then decomposes the trajectory into step-level samples. Methods such as DAPO (Yu et al., 2025) or other replay-based approaches (e.g., ARPO) are excluded, as they can be integrated with our approach.

Training Details All RL training methods use identical hyperparameters. The training batch size is 16, with a rollout number $N = 16$, and a PPO train size of 256. The temperature is 0.7 during training and 0 during evaluation. The spe-

Method	OSWorld (*Train Set*)	OSWorld	AndroidWorld (*Train Set*)	AndroidWorld
UI-Tars-DPO-7B	41.4	16.8	29.8	29.7
+ T-GRPO	48.4 (+7.0)	18.9 (+2.1)	33.3 (+3.5)	31.0 (+1.3)
+ GIGPO	55.5 (+14.1)	21.1 (+4.3)	39.2 (+9.4)	34.0 (+4.3)
+ *STEP* (Ours)	**62.5 (+21.1)**	**23.8 (+7.0)**	**47.6 (+17.8)**	**45.7 (+16.0)**

Table 1: Results on OSWorld and AndroidWorld based on UI-Tars-DPO-7B. The leading results are highlighted with **bold fonts**. Our *STEP* demonstrates superior performance in both benchmarks.

Method	OSWorld (*Train Set*)	OSWorld
GUI-OWL-7B	63.3	29.5
+ GIGPO	64.7 (+1.4)	29.0 (-0.5)
+ *STEP* (Ours)	**71.9 (+8.6)**	**34.2 (+4.7)**

Table 2: Results on OSWorld based on GUI-OWL-7B.

Method	OSWorld (*Train Set*)	OSWorld
STEP (Ours)	**62.5**	**23.8**
w/o SR-Sampling	57.0 (-5.5)	21.7 (-1.1)
w/o Step-Aug	60.1 (-2.4)	23.0 (-0.8)
w/o Both	56.2 (-6.3)	21.4 (-1.4)

Table 3: Ablation study based on UI-Tars-DPO-7B.

cific hyperparameters of our method are: threshold $s_0 = 0.6$, low threshold $s_{\text{low}} = 0.2$, and $\kappa = 10$. Full training settings and hyperparameters are provided in Appendix B.

6.2 Results

The overall performance on OSWorld and AndroidWorld is summarized in Table 1 and Table 2, while the ablation results for *STEP*'s core components are shown in Table 3. Additionally, Figure 1 illustrates the evolution of task categories with success rates above 60% across training epochs, offering further insights into the learning dynamics.

6.2.1 Main Results

As shown in Tables 1 and 2, *STEP* achieves the best overall performance across all evaluated settings, indicating consistent improvements over existing GRPO-based baselines. On OSWorld, *STEP* yields clear gains on the *Train Set*, which translate into stronger overall results. For example, when built on UI-Tars-DPO-7B, it attains 62.5 on the *Train Set* and 23.8 overall, and further improves to 71.9 and 34.2 with a stronger backbone, outperforming GIGPO in both settings. Similar trends are observed on AndroidWorld, where *STEP* achieves an overall score of 45.7, surpassing T-GRPO and GIGPO by 14.7 and 11.7 points, respectively, demonstrating robust performance across environments.

Figure 1 shows that methods leveraging step-level samples, including GIGPO and *STEP*, converge faster and reach higher performance than T-GRPO. One possible explanation is that step-level training decomposes long trajectories into shorter contexts, increasing exposure to state-similar samples and facilitating more efficient learning. Compared with GIGPO, *STEP* consistently achieves further improvements, suggesting a more effective use of step-level supervision.

We also observe differences in how improvements on the *Train Set* transfer to overall evaluation. On OSWorld, gains on the *Train Set* lead to relatively modest overall improvements, whereas on AndroidWorld, the transfer is more consistent. We attribute this difference to dataset construction: OSWorld training tasks are pre-sampled to favor reward-yielding trajectories (Lu et al., 2025), which introduces a larger distribution gap from the full evaluation set, while AndroidWorld adopts random sampling across apps, allowing overall performance to more closely reflect training gains.

6.2.2 Ablation Study

We conduct ablation experiments to assess the contribution of each core component in *STEP*. Three variants are evaluated: (1) w/o SR-Sampling, which removes success-rate sampling; (2) w/o Step-Aug, which removes step augmentation; (3) w/o Both, which disables both components, leaving only our advantage estimation applied to GIGPO.

As shown in Table 3, removing either SR-Sampling or Step-Aug results in a clear performance drop, demonstrating that both components are essential to the effectiveness of *STEP*. Specifically, excluding SR-Sampling causes a 5.5-point decrease on the *Train Set*, suggesting that adaptive sampling plays a key role in stabilizing training and

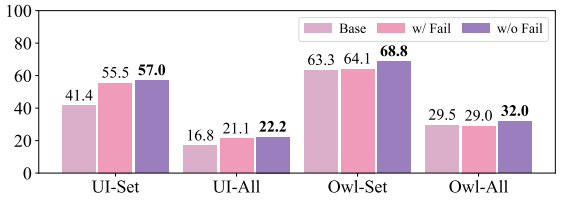


Figure 5: Effect of misleading learning signals on UI-Tars-DPO-7B(UI) and GUI-OWL-7B(Owl).

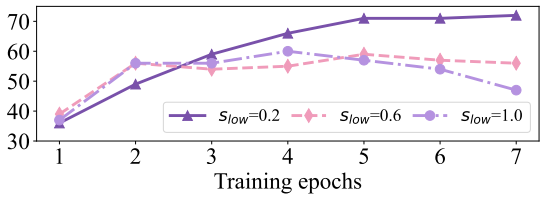


Figure 6: Number of tasks in OSWorld training subset (128 tasks) with a success rate above 60% during training across different s_{low} based on our *STEP*.

improving sample quality. Meanwhile, as shown in Figure 1, removing Step-Aug slows down the growth in task diversity, indicating that augmenting intermediate steps provides additional supervision signals that enhance learning efficiency. The variant without both components (21.4 on OSWorld) performs slightly better than GIGPO (21.1). This result suggests that our advantage estimation is reasonable and that training solely on successful trajectories indeed provides measurable benefits.

7 Discussion

7.1 SR-Traj Mitigates Over-Sampling

Uniform task sampling (U-Traj, used in T-GRPO and GIGPO) tends to over-train on high-success tasks. In contrast, our method, Success-Rate Guided Trajectory Sampling (SR-Traj), addresses this imbalance by adjusting the sampling budget based on the dynamic task success rates. To evaluate this effect, we track the proportion of trajectories from high-success tasks ($s_i \geq 80\%$) during training, computing the average proportion every four training steps. Figure 2 visualizes these trends, providing a direct comparison of how U-Traj and SR-Traj handle mastered tasks over time.

Results As shown in Figure 2, SR-Traj substantially reduces the proportion of high-success trajectories early in training (below 20%), providing more opportunities to explore less-mastered tasks and generating a richer set of informative samples. As training progresses, this proportion increases, reflecting the overall improvement in task success rates. This dynamic pattern indicates that, given the same sampling budget, SR-Traj effectively mitigates premature overfitting to easier tasks, supporting more efficient learning and robust generalization in later stages.

7.2 Effect of Misleading Learning Signals

To investigate the impact of misleading learning signals, we conduct GiGPO experiments on UI-Tars-DPO-7B (UI) and GUI-OWL-7B (Owl) under three training conditions: Base (original performance), w/ Fail (including failure trajectories), and w/o Fail (excluding failure trajectories). Performance was evaluated on both the training subset (Set) and the full dataset (All) of OSWorld. Results are shown in Figure 5.

Results As Figure 5 illustrates, excluding failure trajectories (w/o Fail) consistently yields the highest performance across all groups, despite using fewer training samples. For UI-Tars-DPO-7B, Base scores 41.4 on the subset, w/ Fail improves this to 55.5, and w/o Fail further increases it to 57.0. For GUI-OWL-7B, w/ Fail slightly improves subset performance but can slightly reduce overall performance, whereas w/o Fail consistently achieves the best results.

These findings suggest that failure trajectories may introduce misleading signals that hinder model performance, underscoring the importance of careful management of learning signals.

7.3 Effect of Success-Rate Threshold on Step Augmentation

As mentioned previously, we applied step augmentation to tasks with low success rates (s_{low} = 0.2). To further verify the effect of this design, we conducted experiments by applying augmentation to tasks under different success-rate thresholds (0.2, 0.6 and 1.0). We track the evolution of task categories with success rate above 60% across training epochs according to the main experiments. The results are present in Figure 6.

Results As illustrated in the figure, all settings exhibit a rapid improvement in the early stages of training, confirming that step augmentation serves as an effective approach to accelerate model learning. However, the growth trends for higher thresholds (both 0.6 and 1.0) slow down noticeably and even decline in later stages, suggesting that ap-

Method	Time (min/step)	Speedup
T-GRPO	45.67	1.0×
GIGPO	24.59	1.86×
Our *STEP*	26.25	1.74×

Table 4: Average training time per step for different methods.

plying augmentation to a broader range of tasks may suppress the learning signals from low-success tasks, thereby slightly reducing the model's generalization ability.

7.4 Analysis about Training Effciency

We evaluate training efficiency by measuring the average time per training step. The efficiency experiments are conducted on OSWorld, where we deploy a unified setup of 16 GPUs across two nodes. The environments are simulated on a remote server with 128 parallel instances.

Results Table 4 summarizes the results. Both GIGPO and our *STEP* substantially speed up training, achieving 1.86× and 1.74× faster steps, respectively, nearly halving the time compared to T-GRPO. We attribute this improvement to the substantial shortening of context length in both methods, which reduces inference and training overhead. Remarkably, our method retains high efficiency despite the extra cost of step augmentation during sampling. This is likely due to the avoidance of environment interaction overhead and the efficient parallel inference enabled by step rollout.

8 Conclusion

In this paper, we systematically analyze the challenges of multi-turn reinforcement learning, including uniform task sampling, inaccurate credit assignment, and inefficient sample collection. To address these issues, we propose *STEP* (Success-rate-aware Trajectory-Efficient Policy Optimization), which maintains smoothed per-task success rates to guide adaptive trajectory resampling, decomposes trajectories into step-level samples with success-rate-weighted advantages, and applies step-level GRPO augmentation to improve learning on low-success tasks. Extensive experiments demonstrate that *STEP* substantially outperforms trajectory-level GRPO in both efficiency and effectiveness. As agents face increasingly complex environments, reinforcement learning methods must evolve accordingly. For multi-turn scenarios, we envision *STEP* as a step-level framework that can serve as a foundation and reference point, offering insights for future research in efficient and adaptive multi-turn RL.

Limitations

Our method improves efficiency and performance compared to GRPO by filtering out failed trajectories and distributing trajectory rewards across all steps of successful ones. However, since failed trajectories are entirely discarded, potentially valuable sub-trajectories within them remain unused, leading to a waste of sampling resources. Moreover, even successful trajectories may contain suboptimal or ineffective actions. These observations indicate that the current reward assignment in multi-turn scenarios remains coarse-grained. Future work could explore more fine-grained, step-level reward mechanisms that selectively leverage informative segments from both successful and partially successful trajectories to further enhance learning stability and accuracy.

References

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Krzysztof Choromanski, Tianli Ding, Danny Driess, Kumar Avinava Dubey, Chelsea Finn, Peter R. Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil J. Joshi, Ryan C. Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Sergey Levine, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael S. Ryoo, Grecia Salazar, Pannag R. Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Ho Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Ted Xiao, Tianhe Yu, and Brianna Zitkovich. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *ArXiv*, abs/2307.15818.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for llm agent training. *ArXiv*, abs/2505.10978.

Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. 2023. Multimodal web navigation with instruction-finetuned foundation models. *ArXiv*, abs/2305.11854.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2024. Navigating the digital world as humans do: Universal visual grounding for gui agents. *ArXiv*, abs/2410.05243.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *ArXiv*, abs/2307.12856.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Cogagent: A visual language model for gui agents. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14281–14290.

Kun Huang, Weikai Xu, Yuxuan Liu, Quandong Wang, Pengzhi Gao, Wei Liu, Jian Luan, Bin Wang, and Bo An. 2025. Mobileipl: Enhancing mobile agents thinking process via iterative preference learning. *arXiv preprint arXiv:2505.12299*.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, Weiyu Liu, Percy Liang, Fei-Fei Li, Jiayuan Mao, and Jiajun Wu. 2024. Embodied agent interface: Benchmarking llms for embodied decision making. *ArXiv*, abs/2410.07166.

Yuxuan Liu, Hongda Sun, Wei Liu, Jian Luan, Bo Du, and Rui Yan. 2025a. Mobilesteward: Integrating multiple app-oriented agents with self-evolution to automate cross-app instructions. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pages 883–893.

Yuxuan Liu, Weikai Xu, Kun Huang, Changyu Chen, Jiankun Zhao, Pengzhi Gao, Wei Liu, Jian Luan, Shuo Shang, Bo Du, et al. 2026. Come: Empowering channel-of-mobile-experts with informative hybrid-capabilities reasoning. *arXiv preprint arXiv:2602.24142*.

Zi-Yan Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025b. Understanding r1-zero-like training: A critical perspective. *ArXiv*, abs/2503.20783.

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*, abs/2505.16282.

Karthik Narasimhan, Tejas D. Kulkarni, and Regina Barzilay. 2015. Language understanding for text-based games using deep reinforcement learning. In *Conference on Empirical Methods in Natural Language Processing*.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haolin Chen, Zhaojian Li, Haihua Yang, Hai-Yi Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *ArXiv*, abs/2501.12326.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *ArXiv*, abs/2305.18290.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, Will Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy P. Lillicrap, and Oriana Riva. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *ArXiv*, abs/2405.14573.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junmei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *ArXiv*, abs/2010.03768.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan J. Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2020. Learning to summarize from human feedback. *ArXiv*, abs/2009.01325.

Richard S. Sutton and Andrew G. Barto. 1998. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 9:1054–1054.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi (Jim) Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao

Sang. 2024. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *ArXiv*, abs/2406.01014.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Monica Lam, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Fei-Fei Li, Lijuan Wang, Yejin Choi, and Manling Li. 2025. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *ArXiv*, abs/2504.20073.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *ArXiv*, abs/2404.07972.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629.

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

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Honglin Yu, Weinan Dai, Yuxuan Song, Xiang Wei, Haodong Zhou, Jingjing Liu, Wei Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yong-Xu Wu, and Mingxuan Wang. 2025. Dapo: An open-source llm reinforcement learning system at scale. *ArXiv*, abs/2503.14476.

China. Xiaoyan Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. Appagent: Multimodal agents as smartphone users. *ArXiv*, abs/2312.13771.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Annual Meeting of the Association for Computational Linguistics*.

Zhuosheng Zhang and Aston Zhang. 2023. You only look at screens: Multimodal chain-of-action agents. *ArXiv*, abs/2309.11436.

A Misaligned Learning Signals in OSWorld and AndroidWorld.

We present cases of misaligned learning signals in both OSWorld and AndroidWorld. As shown in Figure 7 and Figure 8, successful and failed trajectories for the same task often share similar intermediate reasoning steps, but diverge at a critical decision point where the failed trajectory takes an incorrect action. This example illustrates that failure trajectories, although their final outcome is negative, often contain many valid intermediate steps. Penalizing the entire trajectory can lead to incorrect learning for these correct steps.

B Setting

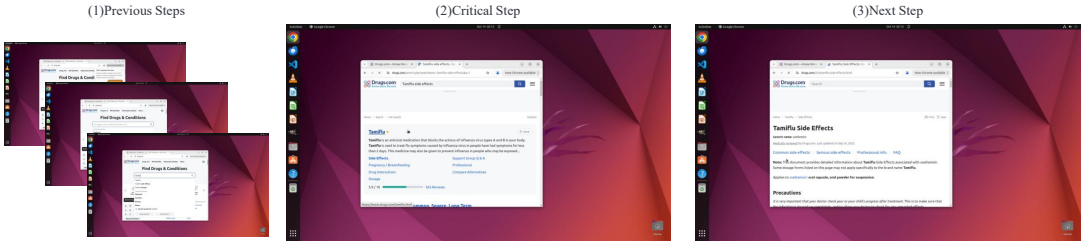
We apply dynamic batch updates in our method and GIGPO, as both decompose trajectories into step-level sample, resulting in a variable number of step samples. For each step sample in GIGPO and *STEP*, the number of history responses t_r and screenshots t_I in H_t are 3 and 0, respectively. Other detail settings are provided in Table 5 and Table 6.

Hyperparameters	All methods
Train batch size	16
PPO batch size	256
Training epoches	8
Rollout numbers	16
Image tokens	1350
Temperature	0.7
Learning rate	1e-6
Kl coefficient	0.001
GPU numbers	16

Table 5: Model configurations for all methods.

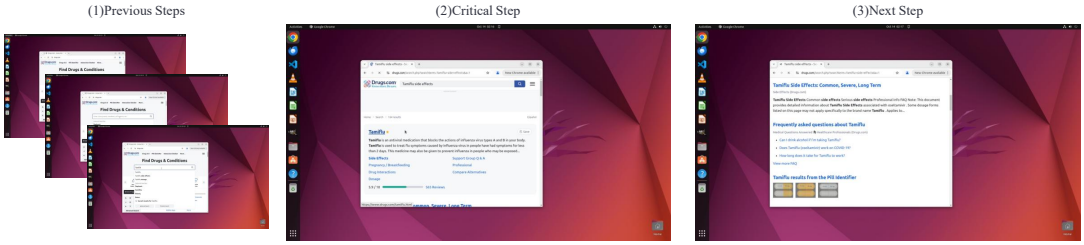
Hyperparameters	T-GRPO	GIGPO&*STEP*
Max turns (OSWorld)	20	30
Max turns (AndroidWorld)	20	25
Prompt length(UI)	3076	4096
Response length(UI)	29692 (256 per turn)	256
Prompt length(OWL)	-	6144
Response length(OWL)	-	1024

Table 6: Model configurations across methods. "UI" refers to UI-Tars-DPO-7B, and "OWL" refers to GUI-OWL-7B.



Critical Action Content: Thought: To proceed with gathering detailed information about the side effects of Tamiflu, I need to click on the "Side Effects" link. This will likely lead to a more comprehensive view of the side effects, which is essential for completing the task. Click on the "Side Effects" link to access detailed information about the side effects of Tamiflu.
Critical Action: click(start_box='(235,618)')

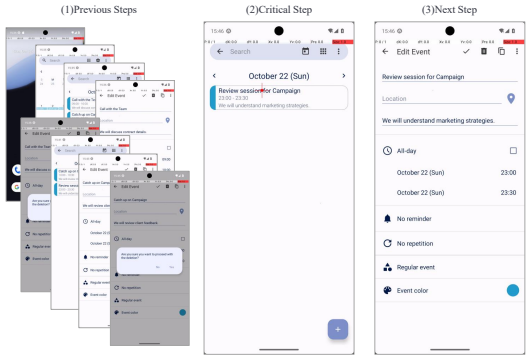
(a) Successful Trajectory



Critical Action Content: Thought: To access the complete information about the side effects of Tamiflu, I need to scroll down the page. This will allow me to view the additional content that is currently not visible, ensuring I gather all relevant details about the drug's side effects. Scroll down the page to reveal more information about the side effects of Tamiflu.
Critical Action: scroll(direction='down', start_box='(307,487)')

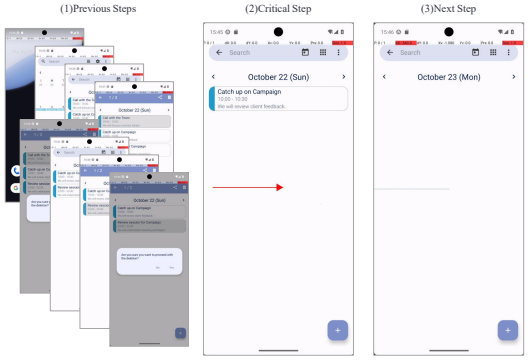
(b) Failed Trajectory

Figure 7: Comparison of success and failure trajectories in OSWorld for the task “Show side effects of Tamiflu.”



Critical Action Content: Thought: To proceed with the task of deleting all events for October 22, I need to tap on the "Review session for Campaign" event. This action will open its details, allowing me to access further options for managing or removing the event, which is a necessary step in completing the task. Tap on the "Review session for Campaign" event to open its details.
Critical Action: click(start_box='(387,206)')

(a) Successful Trajectory



Critical Action Content: Thought: To ensure that all events on October 22, 2023, are reviewed and potentially deleted, I need to scroll through the page to reveal any additional events that may not currently be visible. This step is necessary to locate and manage all events scheduled for that day. Scroll the page in the up direction to display more information and reveal any additional events on October 22, 2023.
Action: scroll(start_box='(65,495)', end_box='(531,495)')
Critical Action: scroll(start_box='(65,495)', end_box='(531,495)')

(b) Failed Trajectory

Figure 8: Comparison of success and failure trajectories in AndoridWorld for the task “In Simple Calendar Pro, delete all the calendar events on 2023-10-22.”