MOBILERL: ONLINE AGENTIC REINFORCEMENT LEARNING FOR MOBILE GUI AGENTS

Yifan Xu[1*†], Xiao Liu[1,2*], Xinghan Liu[1†], Jiaqi Fu[1†], Hanchen Zhang[1†], Bohao Jing[2], Shudan Zhang[1†], Yuting Wang[2], Wenyi Zhao[2], Yuxiao Dong[1]

[1] Tsinghua University [2] Z.AI

ABSTRACT

Building general-purpose graphical user interface (GUI) agents has become increasingly promising with the progress in vision language models. However, developing effective mobile GUI agents with reinforcement learning (RL) remains challenging due to the heavy-tailed distribution of task difficulty and the inefficiency of large-scale environment sampling. We present an online agentic reinforcement learning framework MOBILERL to enhance GUI agents in mobile environments. Its core component is the Difficulty-Adaptive GRPO (ADAGRPO) algorithm. In ADAGRPO, we design difficulty-adaptive positive replay and failure curriculum filtering to adapt the model to different task difficulties. We introduce the shortest-path reward adjustment strategy to reshape rewards concerning the task length in multi-turn agentic tasks. Those strategies jointly stabilize RL training, improve sample efficiency, and generate strong performance across diverse mobile apps and tasks. We apply MOBILERL to two open models (Qwen2.5-VL-7B-Instruct and GLM-4.1V-9B-Base). The resultant MOBILERL-9B model achieves state-of-the-art results in terms of success rates on both AndroidWorld (75.8%) and AndroidLab (46.8%). The MOBILERL framework is adopted in the AutoGLM products, and also open-sourced at `https://github.com/THUDM/MobileRL`.

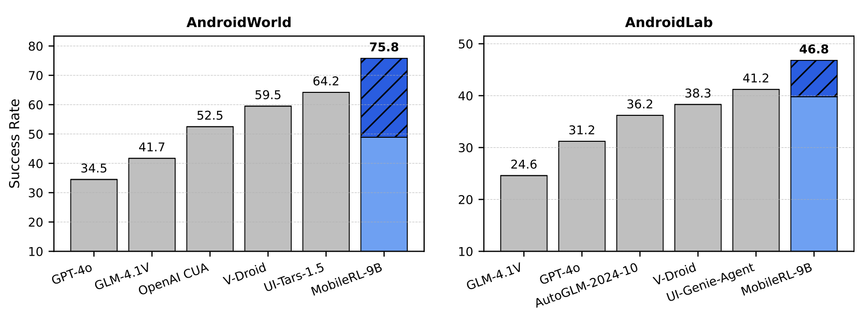


Figure 1: Task success rates on AndroidWorld (Rawles et al., 2024) and AndroidLab (Xu et al., 2024); hatched areas indicate gains from MOBILERL on top of the SFT model.

1 INTRODUCTION

GUI agents—powered by vision language models—have enabled zero-shot interaction with web pages and mobile interfaces (Hong et al., 2023; OpenAI; Bai et al., 2025; Liu et al., 2024). To improve them, significant efforts have focused on supervised fine-tuning or offline imitation learning over static expert demonstrations (Rawles et al., 2023; Xu et al., 2024; Bai et al., 2024; Lu et al., 2025). However, these methods suffer from limited behavior coverage and poor error recovery (Chang et al., 2022).

Reinforcement learning (RL) with verifiable rewards presents a promising alternative (DeepSeek-AI et al., 2025; Hou et al., 2025). Yet, existing datasets with single-step expert labels (Qin et al., 2025; Luo et al., 2025) are insufficient for training or evaluating policies on agentic tasks (i.e., planning

[*] YX and XL contributed equally.
[†] Work partially done while these authors interned at Z.AI.

and reasoning over multi-step action sequences). Although early progress has been made on online learning for GUI agents (Bai et al., 2024; Dong et al., 2025; Dai et al., 2025), efficiently scaling agentic RL in interactive mobile simulators remains largely unexplored.

Specifically, it faces the following technical challenges: (i) *Complex instruction following under sparse positive signals*: base models usually struggle to reliably produce correct action commands for complex, GUI-specific instructions. Due to the heavy cost and latency of mobile emulation, correctly-executed rollouts are rare, resulting in data-inefficient early exploration. (ii) *Large and unstable task difficulty spectrum*: some tasks can succeed with multiple rollouts, while others are persistently unsolvable for the model. Naive sampling wastes computational budget and under-utilizes scarce but informative trajectories (Xu et al., 2024). (iii) *Sampling bottlenecks in large-scale mobile environments*: deploying and managing hundreds of concurrent mobile instances is resource-intensive and hard to reproduce across setups. Low sampling throughput further limits both the scale and efficiency of online agentic RL.

To address these challenges, we present an adaptive online agentic RL framework MOBILERL for advancing mobile GUI agents. MOBILERL consists of three components: reasoning-free supervised fine-turning (SFT), reasoning SFT, and agentic RL. The two SFT stages provide a warm-up for RL. Specifically, reasoning SFT enhances the handling of long and compositional instructions, reduces costly on-policy trials in mobile simulators, and enables the broad use of open or human-labeled datasets without relying on proprietary models.

To enable effective online agentic RL, we introduce Difficulty–Adaptive Group Relative Policy Optimization (ADAGRPO). Built upon group relative policy optimization (GRPO) (Shao et al., 2024), its core idea is to adapt optimization to instance difficulty and explicitly reward solution efficiency. ADAGRPO designs three key strategies: (i) *Difficulty-Adaptive Positive Replay (*ADAPR*)* maintains a curated buffer of challenging, high-quality trajectories and balances them with fresh on-policy samples. In sparse-reward mobile environments, difficult successes are rare yet highly informative; replaying them amplifies their learning signal and stabilizes policy updates. (ii) *Failure Curriculum Filtering (FCF)* down-weights persistently unsolvable tasks using online difficulty statistics, reallocating computational budget toward challenging but feasible instances. Given the heavy-tailed difficulty distribution observed in mobile agent benchmarks (Xu et al., 2024; Rawles et al., 2024), pruning hard dead-ends improves sample efficiency while retaining signal from recoverable failures. (iii) *Shortest-Path Reward Adjustment (SPA)* reshapes the reward function based on completion length, assigning higher returns to shorter solutions. Length-sensitive rewards counteract bias toward verbose and better align with user preferences in mobile interaction contexts.

We implement MOBILERL in the AgentRL framework (Zhang et al., 2025), which supports multi-task, multi-turn agentic RL training. Unlike previous Android simulator implementations (Toyama et al., 2021; Rawles et al., 2024)—which generally do not support true concurrent execution, our framework sustains high throughput that orchestrates hundreds of Dockerized Android virtual devices (AVDs) across multiple machines. This setup enables concurrent interaction with over 1,000 environments while preserving reproducibility. Since most open-source benchmarks and simulators are built upon the Android operating system (Toyama et al., 2021; Rawles et al., 2024), this design ensures seamless compatibility and faithful reproduction of environment behaviors.

We train MOBILERL on Qwen2.5-VL-7B-Instruct (Bai et al., 2025) and GLM-4.1V-9B-Base (Team et al., 2025), producing MobileRL-7B and MobileRL-9B, respectively. First, MobileRL-9B lifts the success rates to 75.8% on ANDROIDWORLD and 46.8% on ANDROIDLAB, significantly outperforming previous state-of-the-art results (64.2% and 41.2%, respectively). Second, though MobileRL-7B is substantially smaller than the 72B-parameter UI-TARS-1.5 (Qin et al., 2025) and UI-GENIE-AGENT (Xiao et al., 2025), it achieves significantly better performance (e.g., +7.8% on ANDROIDWORLD). Third, extensive ablation studies demonstrate the design of ADAGRPO. Finally, the MOBILERL framework is also deployed in the AutoGLM (Liu et al., 2024) production pipeline.

2 MOBILERL

We study mobile GUI agents and introduce the MOBILERL framework, as shown in Figure 2, which aims to address three key challenges in interactive mobile environments: (i) following complex

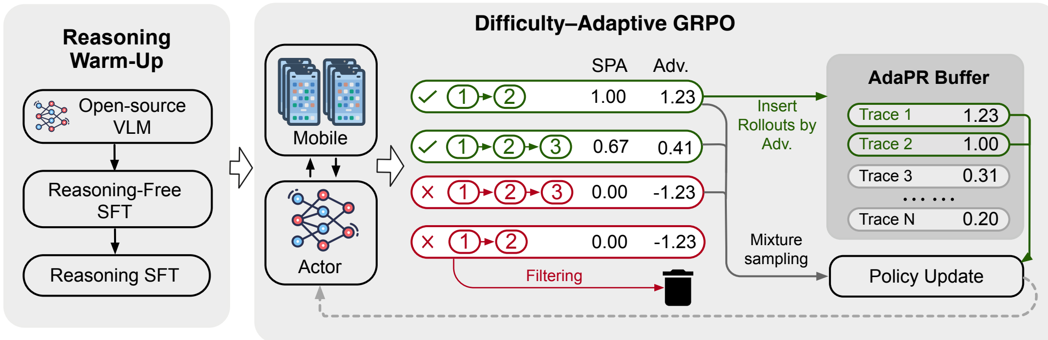


Figure 2: Overview of MOBILERL. It consists of 1) reasoning warm-up with both reasoning-free SFT and reasoning SFT and 2) online agentic RL with ADAGRPO. In ADAGRPO, the warmed-up policy interacts with mobile environments to generate rollouts, which are scored by shortest-path reward adjustment (SPA). High-quality positive trajectories are stored in the AdaPR buffer, while low-performing rollouts are pruned via failure curriculum filtering.

instructions under sparse and delayed rewards; (ii) handling a heavy–tailed and unstable task difficulty distribution; and (iii) overcoming large–scale sampling bottlenecks in mobile emulators.

Given a natural–language instruction (e.g., "open the calendar and add an event for tomorrow at 3 pm"), the agent autonomously performs closed–loop interactions with the mobile device. First, it perceives the current screen, grounds UI elements, and executes a sequence of actions without human intervention. The feedback is sparse and it can be observed only upon successful task completion, at which point the interaction terminates or a predefined horizon is reached. The goal is to learn a policy that generates strong performance across applications and tasks, minimizes unnecessary interactions, and maximizes task success.

Problem Formulation. We model the mobile GUI agent as a finite–horizon Markov Decision Process (MDP) (Littman, 2009), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \mu_0, H)$. The state space $\mathcal{S}$ comprises possible GUI states; each state s_t contains a screenshot of the current device screen and its structured UI hierarchy extracted from XML metadata. The discrete action space $\mathcal{A}$ includes atomic operations such as `Tap`, `Swipe`, `Type`, `LongPress`, and `Finish`. The transition function $P(s_{t+1} \mid s_t, a_t)$ captures the dynamics induced by the Android OS and installed applications. Rewards are binary and terminal: the agent receives 1 if the instruction is completed successfully and 0 otherwise.

Formally, at the start of an episode (t=0) the agent receives an instruction $c \in \mathcal{C}$ and initial state $s_0 \sim \mu_0$. At each timestep t, the agent observes $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$ according to its conditional policy $\pi_\theta(a_t \mid s_t, c)$. The environment then transitions to s_{t+1}. An episode yields a trajectory $\tau = (s_0, a_0, s_1, \ldots, s_T)$ and terminates upon successful completion of the instruction or when the horizon $H \in \mathbb{N}$ is reached ($T \leq H$). Because rewards are sparse and terminal, the return reduces to the final outcome $r_T = R(s_T) \in \{0, 1\}$, and the learning objective is: $\theta^\star = \arg\max_\theta \mathbb{E}_{(s_0,c)\sim\mu_0,\ \tau\sim\pi_\theta}\big[R(s_T)\big]$.

To enable accurate action grounding and interaction with mobile interfaces, we define the agent's observation space as a dual presentation: the current screenshot of the mobile phone and its corresponding compressed Extensible Markup Language (XML) information. The preprocessing steps for XML simplification are detailed in Appendix D.

2.1 THE MOBILERL FRAMEWORK

To build a powerful mobile use agent, we present the MOBILERL framework. It comprises three components: reasoning-free supervised fine-tuning (SFT) on expert demonstration data, an iterative warm-up stage by reasoning SFT, and agentic RL with a difficulty–adaptive policy optimization strategy we developed in this work.

Reasoning-Free SFT. In agentic RL training, sampling in virtual-device environments is usually inefficient; thus, starting online RL directly from a base model was found to be excessively time-consuming in preliminary experiments. Therefore, we perform SFT with expert demonstration data obtained by following the data collection protocol of (Xu et al., 2024). The curated instruction-tuning expert dataset covers 52 applications and 500k interaction steps. Note that this data is reasoning-free.

Reasoning SFT. To further construct a stronger reasoning policy initializer, we perform reasoning SFT via an iterative reasoning refinement strategy over the expert dataset. Manually collected expert demonstrations dataset for mobile use often contains only the final action sequence, omitting intermediate reasoning. Training solely on such "black-box" trajectories yields opaque policies, while many unlabeled tasks remain unused. We leverage an off-the-shelf instruction model to activate expert data and bootstrap a reasoning-augmented training set from raw demonstrations, yielding a structured and transparent policy initialization.

Concretely, we iteratively build the reasoning instruction–tuning pairs in three stages:

- Stage 0 (Bootstrap sampling): For each task x with expert answer a^*, the Instruct model M generates diverse candidate reasoning–action pairs (c_k, a_k) (via, e.g., temperature/nucleus sampling). Whenever $a_k = a^*$, we retain (x, c_k, a^*) in $\mathcal{D}_R$.
- Stage 1 (Supervised fine-tuning): Train an initial reasoning policy π_0^R on $\mathcal{D}_R$.
- Stage 2 (Iterative refinement): At iteration t, π_t^R proposes candidates; those matching a^* are scored by conciseness and $\log P(c)$. The best explanation c^* is added to $\mathcal{D}_{\text{new}}$, and π_{t+1}^R is obtained by fine-tuning. We stop when the match rate saturates.

The resulting reasoning-oriented fine-tuning corpus (about 71.4k steps) is trained for two epochs to produce the reasoning warm-up model used for agentic RL training.

Agentic RL. During agentic RL (multi-turn) training, we face the challenges of immediate reward assignment and sampling efficiency, which are discussed and addressed by building upon the group relative policy optimization (GRPO) (Shao et al., 2024).

Briefly, GRPO advances proximal policy optimization (Schulman et al., 2017) by replacing the learned value baseline with an on-the-fly, group-relative baseline computed from a set of trajectories for the same task. Given a state s, we sample a group of G actions $a_1, \ldots, a_G \sim \pi_{\theta_{\text{old}}}(\cdot \mid s)$ and define the group-relative advantage using the reward function R:

$$A^{\text{GRPO}}(s, a_i) = \frac{R(s, a_i) - \text{avg}_{j=1}^{G} R(s, a_j)}{\text{std}_{j=1}^{G} R(s, a_j)}.$$

Let T_i denote the number of steps for sample i and $s_{i,t}, a_{i,t}$ be the state and action at step t. Define the per-step importance ratio and the per-step advantage as

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t} \mid s_{i,t})}{\pi_{\theta_{\text{old}}}(a_{i,t} \mid s_{i,t})}, \qquad \hat{A}_{i,t} \;\equiv\; A^{\text{GRPO}}(s_{i,t}, a_{i,t}).$$

The policy maximizes the clipped surrogate

$$J^{\text{GRPO}}(\theta) = \mathbb{E}_{\substack{s \sim d_{\pi_{\theta_{\text{old}}}}, \\ a_{1:G} \sim \pi_{\theta_{\text{old}}}(\cdot|s)}} \Bigg[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{T_i} \sum_{t=1}^{T_i} \min\Big(\rho_{i,t}(\theta)\, \hat{A}_{i,t},\ \text{clip}\big(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon\big)\, \hat{A}_{i,t}\Big) - \beta\, D_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\theta_{\text{ref}}}\big) \Bigg],$$

where $D_{\text{KL}}(p\|q) = \sum_a p(a) \log \frac{p(a)}{q(a)}$ and $d_{\pi_{\theta_{\text{old}}}}$ denotes the state visitation distribution.

2.2 Difficulty–Adaptive GRPO

We develop Difficulty–Adaptive GRPO (AdaGRPO) with three strategies—shortest-path reward adjustment (SPA), difficulty–adaptive positive replay (AdaPR) and failure curriculum filtering (FCF)—to address the challenges faced in mobile agentic RL.

First, in multi-turn mobile agentic tasks—where immediate rewards are absent within a single round, unlike single-turn settings (Lu et al., 2025; Team et al., 2025)—the reward allocation strategy must be redesigned. Beyond assigning a uniform terminal reward, we introduce *Shortest-Path Reward Adjustment (SPA)*, which reshapes rewards with respect to task length. The goal of the adjustment is to provide more informative learning signals, guiding the model toward accurate and efficient completion paths and facilitating the computation of trajectory-level advantages.

Second, a further challenge arises from the uniform sampling strategy employed in standard GRPO. In mobile use scenarios—where each sample carries a significant computational cost—this approach results in poor sample efficiency, particularly due to the repeated inclusion of inherently unsolvable tasks. To mitigate this, we adapt data collection and training based on instance difficulty through two mechanisms: Difficulty–Adaptive Positive Replay (AdaPR) and Failure Curriculum Filtering (FCF). At the same time, we restrict redundant successful trajectories to avoid unnecessary updates and promote training efficiency.

2.2.1 Shortest-Path Reward Adjustment (SPA)

In mobile tasks, the environment returns a binary terminal reward $r \in \{0, 1\}$ (Xu et al., 2024; Rawles et al., 2024) indicating task success. Previous approaches typically broadcast this reward to every timestep, i.e., $R(s_t, a_t) = r,\ t = 0, \dots, T$, so that the per-step signal remains aligned with the sparse objective. However, assigning identical rewards to all successful rollouts biases training toward *longer* trajectories, since they contribute more gradient terms. To counteract this, we introduce SPA, which re-scales the reward for each trajectory τ_i as

$$R^{\text{SPA}}(s_t, a_t) = \begin{cases} 1 - \alpha\, \dfrac{T_i - T_{\min}}{T_i}, & r(\tau_i) = 1, \\ 0, & r(\tau_i) = 0, \end{cases} \qquad T_{\min} = \min_{\tau_j \in \mathcal{T}_{\text{succ}}} |\tau_j|,\ \alpha \in (0, 1], \tag{1}$$

where $T_i = |\tau_i|$ is the length of trajectory τ_i, and $\mathcal{T}_{\text{succ}} = \{\tau_j \mid r(\tau_j) = 1\}$ denotes the set of *successful* trajectories for the current problem instance. Here $T_{\min}$ is the length of the shortest successful trajectory in $\mathcal{T}_{\text{succ}}$, and $\alpha \in (0, 1]$ controls the penalty strength. In this formulation, shorter sequences are not automatically considered better; unsuccessful early terminations still receive a reward of 0. This adjustment encourages the policy to prefer shorter, successful paths without sacrificing the success rate.

Trajectory-level Advantage. Given a task, we sample G trajectories $\mathcal{G} = \tau_1, \dots, \tau_G$. For trajectory τ_i with T_i valid steps, we define the trajectory-level advantage as the average of standardized SPA rewards:

$$A_{\text{traj}}(\tau_i) = \frac{1}{T_i} \sum_{t=1}^{T_i} \frac{R^{\text{SPA}}(s_{i,t}, a_{i,t}) - \text{avg}j \in \mathcal{G}, R^{\text{SPA}}(sj, t, a_{j,t})}{\text{std}j \in \mathcal{G}, R^{\text{SPA}}(sj, t, a_{j,t})}. \tag{2}$$

2.2.2 Difficulty–Adaptive Positive Replay (AdaPR)

Difficulty-Adaptive Positive Replay (AdaPR) strategically retains and reuses challenging, high-value trajectories while blending them with fresh on-policy samples. In sparse-reward mobile environments, successful but difficult rollouts are rare yet highly informative; leveraging them effectively strengthens the learning signal and stabilizes policy updates. We introduce three key components of AdaPR: buffer construction for high-quality trajectory selection, mixture sampling to balance replay and exploration, and negative rollout pruning to reduce noise.

Buffer Construction. At iteration t, the rollout set is $\mathcal{T}_t = \{\tau^{(1)}, \dots, \tau^{(N)}\}$, collected under the current policy π_{θ_t}. We compute $A_{\text{traj}}(\tau)$ via equation 2 and insert the top κ trajectories into the replay buffer $\mathcal{B}$.

Mixture Sampling. Each policy update is performed on a mini-batch of M trajectories obtained from the mixture distribution

$$q(\tau) \;=\; \gamma\, p_{\mathcal{B}}(\tau) \;+\; (1-\gamma)\, p_{\text{on}}(\tau) \tag{3}$$

where p_{on} is the on-policy distribution π_{θ_t} and $p_{\mathcal{B}}$ is the empirical distribution over $\mathcal{B}$. To keep the replay contribution under control, at most γM trajectories with the highest *current* advantage are drawn from $\mathcal{B}$, preserving on-policy diversity.

Pruning Negative Rollouts. To stabilize training, we prune trajectories with the lowest advantages, reducing noisy samples in the replay buffer. High-advantage trajectories, if stored, are expected to be sampled only once on average. Thus, we maintain a maximum positive-to-negative trajectory ratio of 1:2 by randomly discarding the lowest-advantage trajectories.

2.2.3 Failure Curriculum Filtering (FCF)

To avoid repeatedly sampling tasks that yield zero reward—which wastes computation and hinders the collection of positive advantage data, we propose failure curriculum filtering. In FCF, any task producing all-zero rewards for two consecutive epochs enters a three-epoch cooldown, during which its sampling probability is reduced according to $w_{\text{task}} = \exp(-f)$, where f is the number of consecutive failure epochs. Tasks with consistently high failure counts are permanently removed from the sampling pool. For stability, failure histories from previous training are retained.

Summary. In summary, MobileRL consists of reasoning-free SFT, reasoning-augmented warm-up, and difficulty-adaptive RL for training mobile GUI agents. Reasoning-free SFT helps build a strong action foundation from expert demonstrations, while reasoning SFT adds intermediate reasoning to improve instruction following and policy transparency. On top of this initialization, agentic RL with AdaGRPO addresses the challenges of sparse terminal rewards, heavy-tailed task difficulty, and expensive sampling. Specifically, SPA reshapes terminal rewards for denser feedback, AdaPR strategically reuses challenging successful trajectories, and FCF filters out persistently-unsolvable tasks.

3 Experiments

3.1 Experiments Settings

Datasets and Benchmarks. We evaluate on two interactive Android benchmarks: AndroidWorld with 116 tasks across 20 apps (Rawles et al., 2024) and AndroidLab with 138 tasks across 9 apps (Xu et al., 2024), totaling 254 tasks that span diverse, real-world goals. Both sets offer instruction-following environments where agents act to complete tasks. We use AndroidWorld's rule-based trajectory rewards; for AndroidLab, which lacks training rewards, we generate signals via a VLM-based reward model (see Appendix C).

Baselines. Our baselines encompass both closed- and open-source agents and models. The closed-source LLMs include GPT-4o-2024-11-20 (OpenAI, 2023) and Claude-Sonnet-4-20250514-thinking (Anthropic, 2023), and closed-source agents UI-Tars-1.5 (Qin et al., 2025) and Auto-GLM (Liu et al., 2024). The open-source VLMs, including Qwen2.5-VL-7B-Instruct (Bai et al., 2025), GLM-4.1V-9B-Thinking (Team et al., 2025), UI-Tars-7B (Qin et al., 2025), V-Droid (Dai et al., 2025) and UI-Genie-Agent (Xiao et al., 2025), are used.

3.2 Main Results

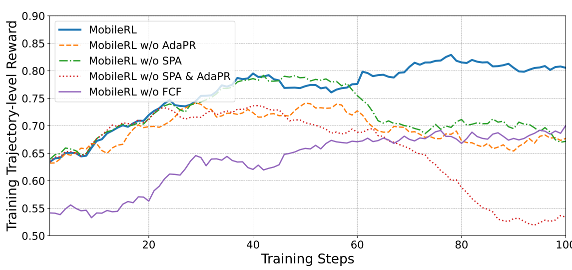
We evaluate MobileRL by using Qwen2.5-VL-7B and GLM-4.1V-9B-Base as backbones on AndroidWorld and AndroidLab. As shown in Table 1, MobileRL significantly outperforms both proprietary models (e.g., GPT-4o and Claude-Sonnet-4) and open-source models (e.g., V-Droid and UI-Tars-7B). With Qwen2.5-VL-7B as the backbone, MobileRL achieves 72.0%on AndroidWorld and 42.5%on AndroidLab, outperforming previous state-of-the-art methods. By using GLM-4.1V-9B as the backbone, the performance of MobileRL further improves to 75.8%on AndroidWorld and 46.8%on AndroidLab, achieving the best results across all models.

Table 1: Success rates (%) of proprietary and open-source models on AndroidWorld and AndroidLab for mobile GUI interaction tasks.

Models	**#Params**	AndroidWorld	AndroidLab
Proprietary Models			
GPT-4o-2024-11-20 (OpenAI, 2023)	-	34.5	31.2
Claude-Sonnet-4-20250514-thinking (Anthropic, 2023)	-	41.0	40.6
UI-Tars-1.5 (Qin et al., 2025)	-	<u>64.2</u>	38.3
AUTOGLM-2024-10 (Liu et al., 2024)	-	–	36.2
Open Models			
Qwen2.5-VL-7B-Instruct (Bai et al., 2025)	7B	27.6	10.1
GLM-4.1V-9B-Thinking (Team et al., 2025)	9B	41.7	24.6
UI-Tars-7B (Qin et al., 2025)	7B	33.0	32.6
V-Droid (Dai et al., 2025)	8B	59.5	38.3
UI-Genie-Agent (Xiao et al., 2025)	72B	-	<u>41.2</u>
MOBILERL *(Ours)*			
w/ Qwen2.5-VL-7B	7B	72.0	42.5
w/ GLM-4.1V-9B-Base	9B	**75.8**	**46.8**

Models	AndroidWorld	AndroidLab
Qwen2.5-VL-7B-Instruct (Bai et al., 2025)	27.6	10.1
+ Reasoning-Free SFT	50.2$_{+22.6}$	36.9$_{+26.8}$
+ Reasoning SFT	56.8$_{+6.6}$	38.7$_{+1.8}$
+ ADAGRPO (MOBILERL-7B)	72.0$_{+15.2}$	42.5$_{+3.8}$
GLM-4.1V-9B-Base (Team et al., 2025)	7.7	10.1
+ Reasoning-Free SFT	48.9$_{+41.2}$	39.8$_{+29.7}$
+ Reasoning SFT	66.1$_{+17.2}$	40.3$_{+0.5}$
+ ADAGRPO (MOBILERL-9B)	75.8$_{+9.7}$	46.8$_{+6.5}$

(a) Improvements of task success rates by incrementally applying Reasoning-Free SFT, Reasoning SFT, and ADAGRPO to the base models, respectively.



(b) Training trajectory-level rewards from the AndroidWorld environment with respect to training steps.

Models	AndroidWorld
MOBILERL	**71.1**
w/o ADAPR	63.6
w/o SPA	69.1
w/o ADAPR &SPA	58.5
w/o FCF	64.8
w/o ADAGRPO	56.8

(c) Test performance on the AndroidWorld test set under different variants.

Figure 3: Ablation studies of the MOBILERL framework and its ADAGRPO algorithm. All test set results are averaged over three runs to mitigate the impact of randomness.

3.3 ABLATION STUDY

To evaluate the contributions of the MOBILERL framework and the components of the ADAGRPO algorithm, we start with two base models and progressively apply Reasoning-Free SFT, Reasoning

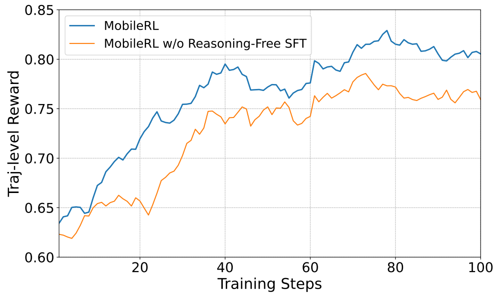


Figure 4: Effect of reasoning-free SFT evaluated on AndroidWorld.

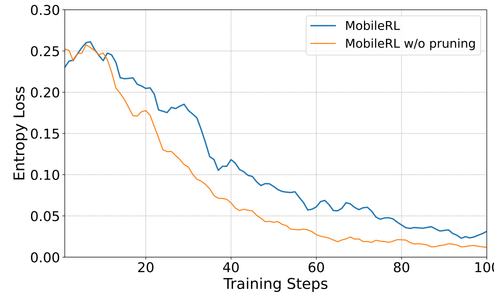


Figure 5: Effect of the pruning-negative strategy evaluated on AndroidWorld.

SFT, and ADAGRPO. Then, using the Qwen2.5-VL-7B-Instruct model trained with Reasoning-Free SFT and Reasoning SFT as the initialization point, we conduct an analysis of the impact of each component of ADAGRPO—AdaPR, SPA, and FCF.

MOBILERL Ablation. We summarize stage-wise gains in success rate (SR) in Table 3a. For Qwen2.5-VL-7B, the combined improvements of MOBILERL are +44.4% on AndroidWorld and +32.4% on AndroidLab; For GLM-4.1V-9B, the overall gains are +68.1% and +36.7%. Overall, Reasoning-Free SFT delivers a strong initial lift and Reasoning SFT offers additional improvements. Building upon the strong foundation established by the preceding stages, the ADAGRPO stage further augments the final performance, achieving an improvement exceeding 10% on the *AndroidWorld* dataset, where rewards are verifiable, and over 5% on the *AndroidLab* test set, which is evaluated using the reward model.

ADAGRPO Ablation. The design of ADAGRPO covers SPA, ADAPR, and FCF. The ablations are performed on four settings: (i) MOBILERL w/o ADAPR (no replay), (ii) MOBILERL w/o SPA (no reward shaping), (iii) MOBILERL w/o ADAPR & SPA (neither), and (iv) MOBILERL w/o FCF (uniform sampling).

We report on-policy trajectory reward curves during training (excluding replayed trajectories) in Figure 3b and the final success rates on the AndroidWorld test set in Table 3c. To avoid bias from the AndroidLab reward model, we use only AndroidWorld in this study.

Each component of ADAGRPO contributes to improving the performance of MOBILERL. Specifically, we have the following observations:

- **FCF under constraints.** With a 100-step budget ($>$ 40 hours), FCF plays a key role in filtering. Removing it biases early sampling toward overly hard tasks, yielding many negatives and a lower reward ceiling. The w/o FCF curve keeps rising, suggesting stronger results with more efficient pipelines or simulators.
- **FCF only (w/o ADAPR & SPA).** Training is initially stable but collapses after about 30 steps, indicating that ADAPR and SPA are necessary for stabilizing the training.
- **Effect of ADAPR.** After about seven steps (once the replay buffer is populated), the gap between w/o ADAPR and the full MOBILERL method grows, showing the benefit of replay.
- **Effect of SPA.** Noticeable gains of SPA appear after roughly 60 steps, likely because the lack of step-wise control leads to overly long trajectories. Section 3.3 confirms that SPA effectively constrains trajectory length.

Is Reasoning-Free SFT Still Necessary? We apply supervised fine-tuning on the expert dataset without reasoning traces, which we term *Reasoning-Free SFT*. The fine-tuning data in this stage contains only action sequences, without intermediate thought processes. This raises a key question: *Is fine-tuning with expert data that lacks reasoning still beneficial?*

We compare MOBILERL (Reasoning-Free SFT + Reasoning SFT + ADAGRPO) with MOBILERL without Reasoning-Free SFT in Figure 4. Interestingly, our experiments show that incorporating Reasoning-Free SFT consistently improves performance on AndroidWorld. It suggests that without

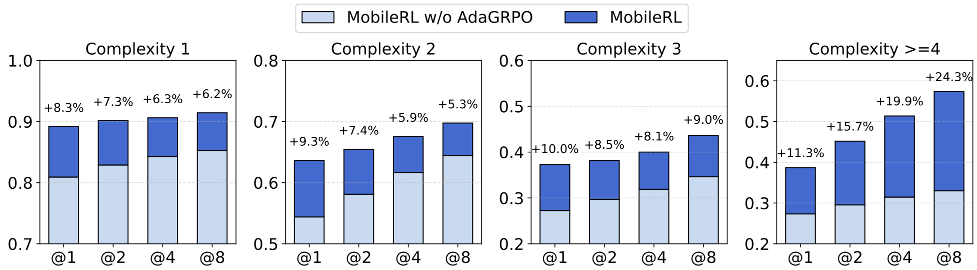


Figure 6: Pass@k on AndroidWorld by task complexity (Rawles et al. (2024)) levels. Pass@k is the fraction of tasks solved within the top-k attempts.

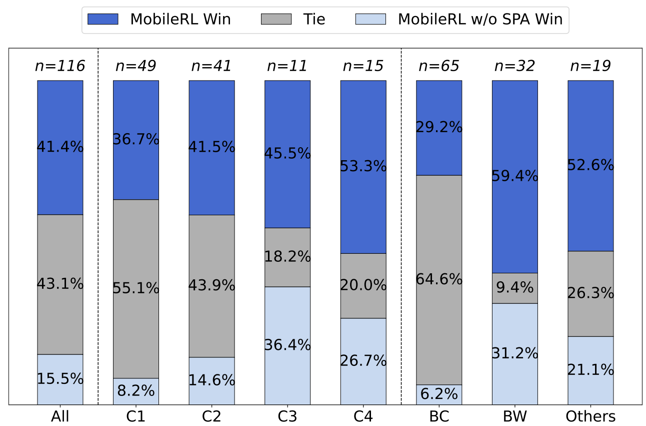


Figure 7: Win rates of MOBILERL vs. MOBILERL w/o SPA, where a *win* means completing a task with fewer steps. n denotes the number of task templates per category. Categories: *All* (all templates); *C1, C2, C3, C4* (complexity levels 1,2,3,4, respectively); *BC/BW* (both correct/both wrong); *Others* (exactly one method correct).

explicit reasoning traces, (Reasoning-Free) SFT contributes to stabilizing training and enhancing final results for MOBILERL.

Effect of Pruning Negative Trajectories. We further study a pruning strategy that discards overly frequent erroneous trajectories from the reinforcement learning buffer before each update. As depicted in Figure 5, this filtering keeps the policy entropy consistently higher during training. By pruning trajectories whose advantages remain persistently negative, the agent avoids being driven by detrimental gradients; probability mass is instead spread over a broader action space, fostering exploration and delaying premature convergence. This pruning strategy ultimately leads to more robust policy learning.

Success Rates by Task Complexity. We divide the AndroidWorld test set by rounded-up *Complexity* (Rawles et al., 2024): complexity level 1 (1–10 steps), complexity level 2 (11–20), complexity level 3 (21–30), and complexity level 4+ (>30). We run eight test trials at temperature 1.0 and report pass@1/2/4/8 in Figure 6.

Our method yields consistent gains at all difficulty levels, with improvements increasing alongside complexity. Notably, post-RL pass@1 exceeds pre-RL pass@8, indicating substantial effectiveness. The relatively modest changes in pass@8 suggest exploration is preserved but better targeted. Consistent with ADAPR 's design for heavy-tailed difficulty, the largest gains occur on high-complexity tasks (complexity levels 3 and 4).

Impact of SPA on Step Efficiency. Although SPA has the smallest impact on overall accuracy in the ablation, its effect on step efficiency is clear. As shown in Figure 7, partitioning tasks by complexity reveals that MOBILERL with SPA consistently completes tasks in fewer steps across all difficulty levels. Moreover, when we compare cases where both models are correct (BC), both are wrong (BW), and those where only one is correct, MOBILERL with SPA more frequently yields shorter trajectories in every group—most prominently in the BW setting—indicating that SPA systematically reduces completion length regardless of final correctness.

4 RELATED WORK

Mobile GUI Agents. Recent work leverages powerful language models to build agents that operate real PCs and phones (Agashe et al., 2025; Qin et al., 2025; Lai et al., 2025b), including Android agents that perceive GUIs and act via taps, swipes, and text (Toyama et al., 2021; Xu et al., 2024). To improve action prediction and learning, frameworks explore multimodal exploration (Yang et al., 2023), modular reasoning (Lai et al., 2025a), verifier-driven control (Dai et al., 2025), and small-LM code-based execution (Wen et al., 2025). Yet many systems still rely on offline RL or single-turn data: DigiRL uses offline demonstrations (Bai et al., 2024); U1-R1 trains on single-step episodes (Lu et al., 2025); and UI-Tars applies DPO in an offline regime (Qin et al., 2025), leaving online, multi-turn RL for adaptive mobile agents largely unexplored.

Benchmarks for Mobile Agents. Benchmarking generally follows two tracks. Static or replay-based settings—AndroidControl, Android in the Wild, MobileAgentBench, and Mobile-Bench—offer many tasks but lack real-time interaction (Li et al., 2024; Rawles et al., 2023; Wang et al., 2024; Deng et al., 2024). Interactive emulator environments—AndroidWorld, AndroidLab, and B-MOCA—span diverse apps and realistic tasks yet remain challenging for current agents (Rawles et al., 2024; Xu et al., 2024; Lee et al., 2024). To the best of our knowledge, all public mobile GUI benchmarks to date target the Android operating system.

5 CONCLUSION

In this work, we present MOBILERL, an agentic RL framework that advances mobile GUI agents. It achieves this by combining staged initialization with an adaptive reinforcement learning algorithm (ADAGRPO). Training begins with reasoning-free SFT on large-scale expert demonstrations to establish a solid task foundation, followed by a reasoning-augmented warm-up stage that adds intermediate rationales to improve transparency and reduce cold-start exploration costs. Building on this, we introduce Difficulty–Adaptive GRPO (ADAGRPO), which enhances GRPO with shortest-path reward adjustment, adaptive positive replay, and failure curriculum filtering. These three strategies improve sample efficiency and guide policies toward more accurate and efficient task completion. Experiments on AndroidWorld and AndroidLab demonstrate that MOBILERL with open models significantly outperforms both open-source and proprietary baselines.

For future work, we aim to improve the broader generalization of MOBILERL across diverse mobile platforms, application domains, and unseen UI layouts. In addition, we plan to develop a general verification model that can automatically assess the correctness, reliability, and safety of agent actions, enabling more robust and trustworthy deployment in real-world scenarios.

REFERENCES

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents, 2025. URL `https://arxiv.org/abs/2504.00906`.

Anthropic. Introducing claude, 2023. URL `https://www.anthropic.com/index/introducing-claude`.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning, 2024. URL `https://arxiv.org/abs/2406.11896`.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Jonathan D. Chang, Masatoshi Uehara, Dhruv Sreenivas, Rahul Kidambi, and Wen Sun. Mitigating covariate shift in imitation learning via offline data without great coverage, 2022. URL https://arxiv.org/abs/2106.03207.

Gaole Dai, Shiqi Jiang, Ting Cao, Yuanchun Li, Yuqing Yang, Rui Tan, Mo Li, and Lili Qiu. Advancing mobile gui agents: A verifier-driven approach to practical deployment, 2025. URL https://arxiv.org/abs/2503.15937.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Liujianfeng Li, Ang Li, Jian Luan, Bin Wang, Rui Yan, and Shuo Shang. Mobile-bench: An evaluation benchmark for LLM-based mobile agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Long Papers)*, pp. 8813–8831, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.478.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic reinforced policy optimization, 2025. URL https://arxiv.org/abs/2507.19849.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents, 2023.

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. T1: Advancing language model reasoning through reinforcement learning and inference scaling, 2025. URL https://arxiv.org/abs/2501.11651.

Hanyu Lai, Junjie Gao, Xiao Liu, Yifan Xu, Shudan Zhang, Yuxiao Dong, and Jie Tang. Androidgen: Building an android language agent under data scarcity, 2025a. URL `https://arxiv.org/abs/2504.19298`.

Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. Computerrl: Scaling end-to-end online reinforcement learning for computer use agents, 2025b. URL `https://arxiv.org/abs/2508.14040`.

Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. Benchmarking mobile device control agents across diverse configurations, 2024. URL `https://arxiv.org/abs/2404.16660`.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on computer control agents. *arXiv preprint arXiv:2406.03679*, 2024.

Michael L Littman. A tutorial on partially observable markov decision processes. *Journal of Mathematical Psychology*, 53(3):119–125, 2009.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, Junjie Gao, Junjun Shan, Kangning Liu, Shudan Zhang, Shuntian Yao, Siyi Cheng, Wentao Yao, Wenyi Zhao, Xinghan Liu, Xinyi Liu, Xinying Chen, Xinyue Yang, Yang Yang, Yifan Xu, Yu Yang, Yujia Wang, Yulin Xu, Zehan Qi, Yuxiao Dong, and Jie Tang. Autoglm: Autonomous foundation agents for guis, 2024. URL `https://arxiv.org/abs/2411.00820`.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning, 2025. URL `https://arxiv.org/abs/2503.21620`.

Run Luo, Lu Wang, Wanwei He, and Xiaobo Xia. Gui-r1 : A generalist r1-style vision-language action model for gui agents, 2025. URL `https://arxiv.org/abs/2504.10458`.

OpenAI. Computer-using agent. URL `https://openai.com/index/computer-using-agent/`.

OpenAI. Gpt-4 technical report, 2023.

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, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL `https://arxiv.org/abs/2501.12326`.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088*, 2023.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL `https://arxiv.org/abs/2402.03300`.

GLM-V Team, Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, Baoxu Wang, Boyan Shi, Changyu Pang, Chenhui Zhang, Da Yin, Fan Yang, Guoqing Chen, Jiazheng Xu, Jiali Chen, Jing Chen, Jinhao Chen, Jinghao Lin, Jinjiang Wang, Junjie Chen, Leqi Lei, Letian Gong, Leyi Pan, Mingzhi Zhang, Qinkai Zheng, Sheng Yang, Shi Zhong, Shiyu Huang, Shuyuan Zhao, Siyan Xue, Shangqin Tu, Shengbiao Meng, Tianshu Zhang, Tianwei Luo, Tianxiang Hao, Wenkai Li, Wei Jia, Xin Lyu, Xuancheng Huang, Yanling Wang, Yadong Xue, Yanfeng Wang, Yifan An, Yifan Du, Yiming Shi, Yiheng Huang, Yilin Niu, Yuan Wang, Yuanchang Yue, Yuchen Li, Yutao Zhang, Yuxuan Zhang, Zhanxiao Du, Zhenyu Hou, Zhao Xue, Zhengxiao Du, Zihan Wang, Peng Zhang, Debing Liu, Bin Xu, Juanzi Li, Minlie Huang, Yuxiao Dong, and Jie Tang. Glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning, 2025. URL https://arxiv.org/abs/2507.01006.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

Luyuan Wang, Yongyu Deng, Yiwei Zha, Guodong Mao, Qinmin Wang, Tianchen Min, Wei Chen, and Shoufa Chen. Mobileagentbench: An efficient and user-friendly benchmark for mobile llm agents. 2024. arXiv preprint arXiv:2406.08184.

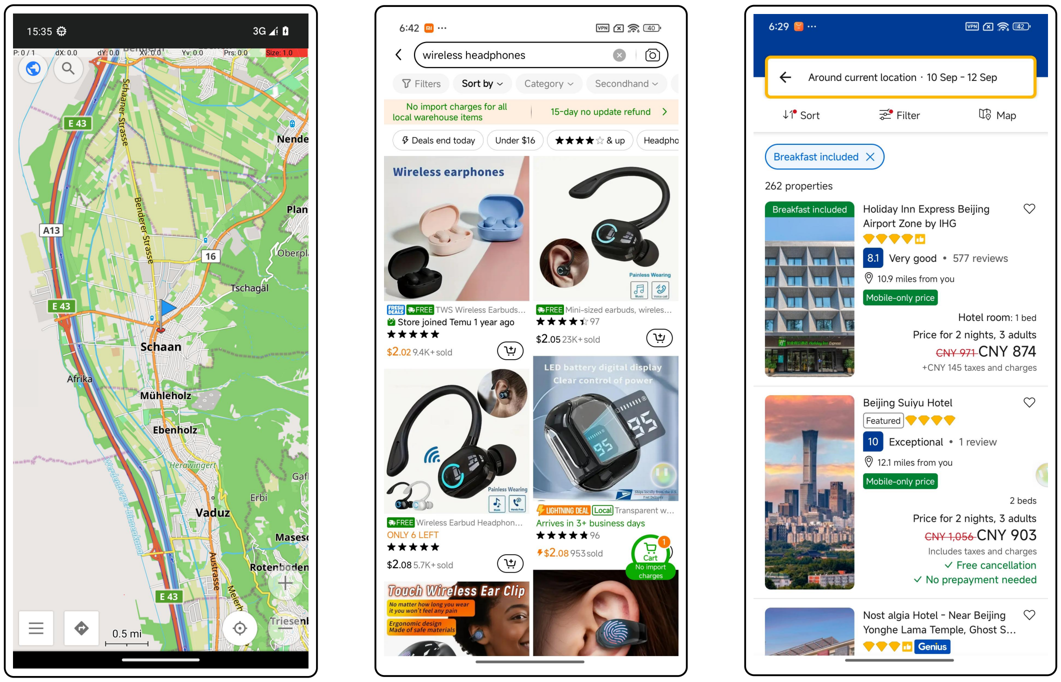
Hao Wen, Shizuo Tian, Borislav Pavlov, Wenjie Du, Yixuan Li, Ge Chang, Shanhui Zhao, Jiacheng Liu, Yunxin Liu, Ya-Qin Zhang, and Yuanchun Li. Autodroid-v2: Boosting slm-based gui agents via code generation, 2025. URL https://arxiv.org/abs/2412.18116.

Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, Shuai Ren, Yafei Wen, Xiaoxin Chen, Aojun Zhou, and Hongsheng Li. Ui-genie: A self-improving approach for iteratively boosting mllm-based mobile gui agents, 2025. URL https://arxiv.org/abs/2505.21496.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents, 2024. URL https://arxiv.org/abs/2410.24024.

Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023.

Hanchen Zhang, Xiao Liu, Bowen Lv, Xueqiao Sun, Bohao Jing, Iat Long Iong, Zehan Qi, Hanyu Lai, Yifan Xu, Rui Lu, Zhenyu Hou, Hongning Wang, Jie Tang, and Yuxiao Dong. Agentrl: Scaling agentic reinforcement learning with a multi-turn, multi-task framework. *arXiv preprint*, 2025.



(a) Add a location marker for 47.16, 9.51 in the OsmAnd maps app.

(b) Search for wireless headphones in temu, and sort by price low to high.

(c) Search for hotels on Booking, check-in date is 09-10, check-out date is 09-12, sort by prices.

Figure 8: Example mobile tasks finished by our agent. Our agent can automatically perform tasks according to human instructions in academic benchmarks and real-world applications.

A Case Study

We present a case study from **AndroidWorld** evaluating three agents: the Reasoning-Free SFT agent, Reasoning SFT agent, and MobileRL agent. We choose `SimpleCalendarAddRepeatingEvent`, which requires creating a recurring event titled "Review session for Budget Planning," starting on 2023-10-15 at 14:00, lasting 60 minutes, repeating weekly without end, and including the description: "We will understand business objectives. Remember to confirm attendance."

All agents successfully configured the title and basic settings in the initial steps. Their performance diverged in subsequent steps. The Reasoning-Free SFT agent made timing errors (setting 16:00 instead of 15:00) and executed redundant checks, revealing weak task understanding (Figure 9). The Reasoning SFT agent skipped an adjustment step, yielding an incorrect event duration (Figure 10). By contrast, the full MobileRL agent completed the task accurately and efficiently, satisfying all requirements without redundant operations (Figure 11).

B Training Details

B.1 Hyperparameters

Main hyperparameters are listed in Table 2.

B.2 Action Space.

We design a series of actions based on AndroidLab (Xu et al., 2024), which supports a concise set of actions for GUI interaction, as described in Table 3.

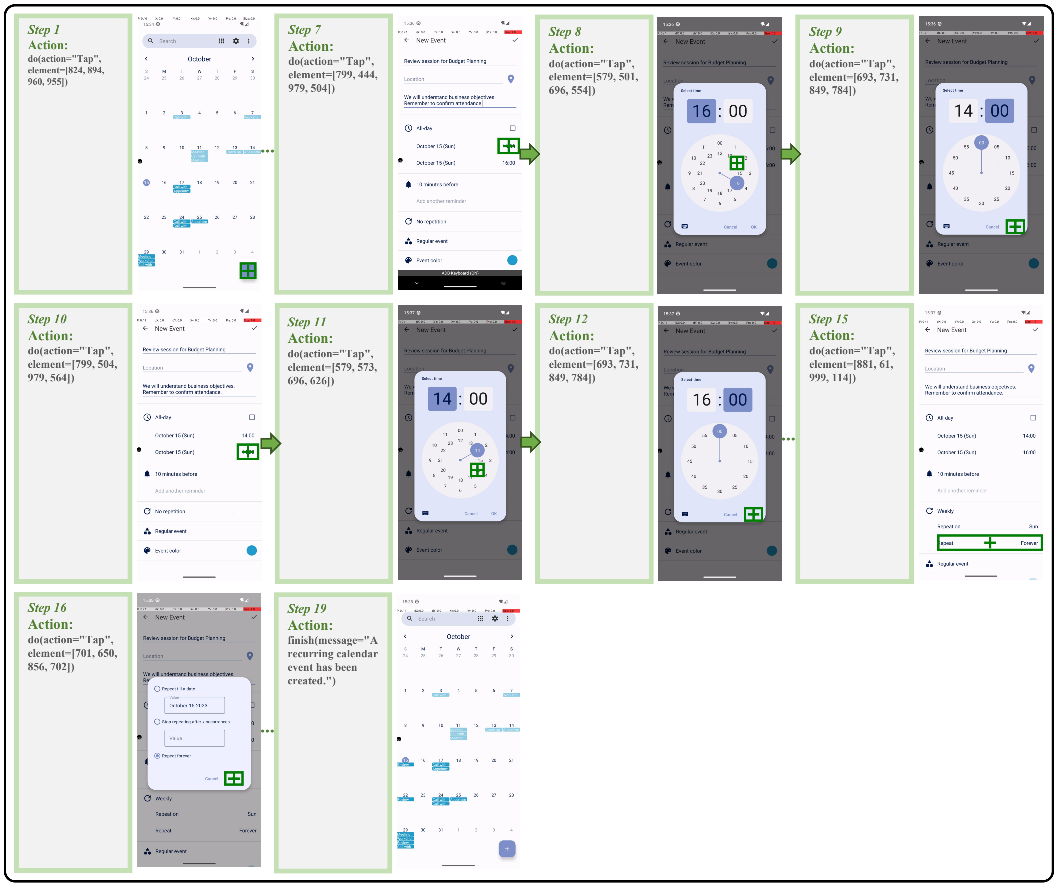


Figure 9: The Reasoning-Free SFT agent, where the expert data does not include the reasoning part, in the `SimpleCalendarAddRepeatingEvent` task in AndroidWorld. Because of space constraints, we only present the most critical and notable steps for comparison.

B.3 Training Tasks in AdaGRPO

For reinforcement learning, we construct extended training splits by varying initial states, requirements, and task compositions, while excluding any overlap with evaluation. The corpus combines AndroidWorld's full training split with ∼1,000 AndroidLab tasks, sampled at a fixed 4:1 ratio (AndroidWorld: AndroidLab).

C Reward Model for AndroidLab

C.1 Method Overview

Since the AndroidLab environment does not provide rule-based rewards for training data, we adopt a VLM-based reward model to supply reward signals during reinforcement learning. Specifically, we first execute all training and test tasks multiple times using medium-capability VLMs, including different versions of our models and proprietary VLMs, generating execution traces. Then, strong proprietary VLMs assign binary scores to each trace. For scoring, each model receives the full task screenshot sequence concatenated into a single image, along with step-by-step action descriptions, and is instructed to produce a reasoning process before outputting a score. For training-set tasks (non-rule-based), we use the majority vote among the three scores as the label and retain the reasoning; for test-set tasks, we use the existing rule-based reward as the label and keep only the model scores and reasoning consistent with it. This process yields several thousand labeled samples, which we use to fine-tune GLM-4.1V-9B-Thinking as the base model. After training, we evaluate the reward model

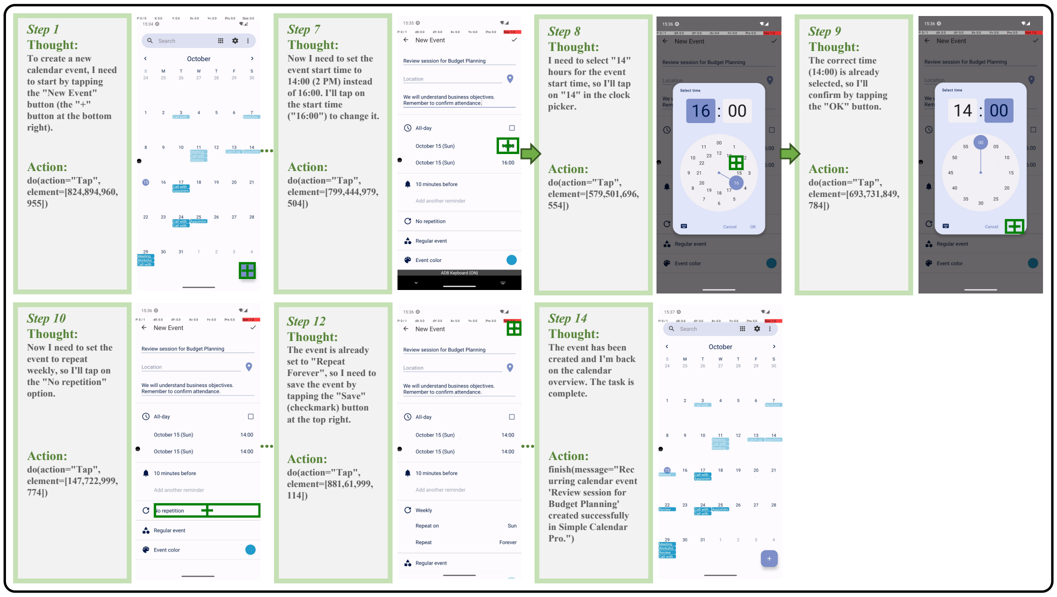


Figure 10: Reasoning SFT agent in the `SimpleCalendarAddRepeatingEvent` task in AndroidWorld. Because of space constraints, we only present the most critical and notable steps for comparison.

Table 2: Summary of Main Hyperparameters

Component	Hyperparameter	Value
Data	Max Prompt Length	16384
Data	Max Response Length	4096
Data	Train Batch Size	256
Data	Validation Batch Size	256
Actor / Policy	Strategy (Parallelism)	FSDP
Actor / Policy	PPO Micro Batch Size/GPU	4
Actor / Policy	Learning Rate (LR)	1e-6
Actor / Policy	Gradient Clipping	1.0
Actor / Policy	Clip Ratio	0.2
Actor / Policy	PPO Epochs	1
Rollout & Sampling	Sampling Temperature	1.0
Rollout & Sampling	Max New Tokens	4096
Rollout & Sampling	Number of Samples (n)	16
Rollout & Sampling	Max Turns	50
Rollout & Sampling	Max Pixels	500000
Rollout & Sampling	Min Pixels	65536
Algorithm	KL Loss Coefficient/ β	0.001
Algorithm	SPA/ α	1.0
Algorithm	ADAPR/ Replay Buffer Size	256
Algorithm	ADAPR/ γ	1.0
Algorithm	ADAPR/ κ	0.25

on a curated set of 1000 AndroidLab traces with verified labels, selecting the best-performing version (86% accuracy) for online reinforcement learning.

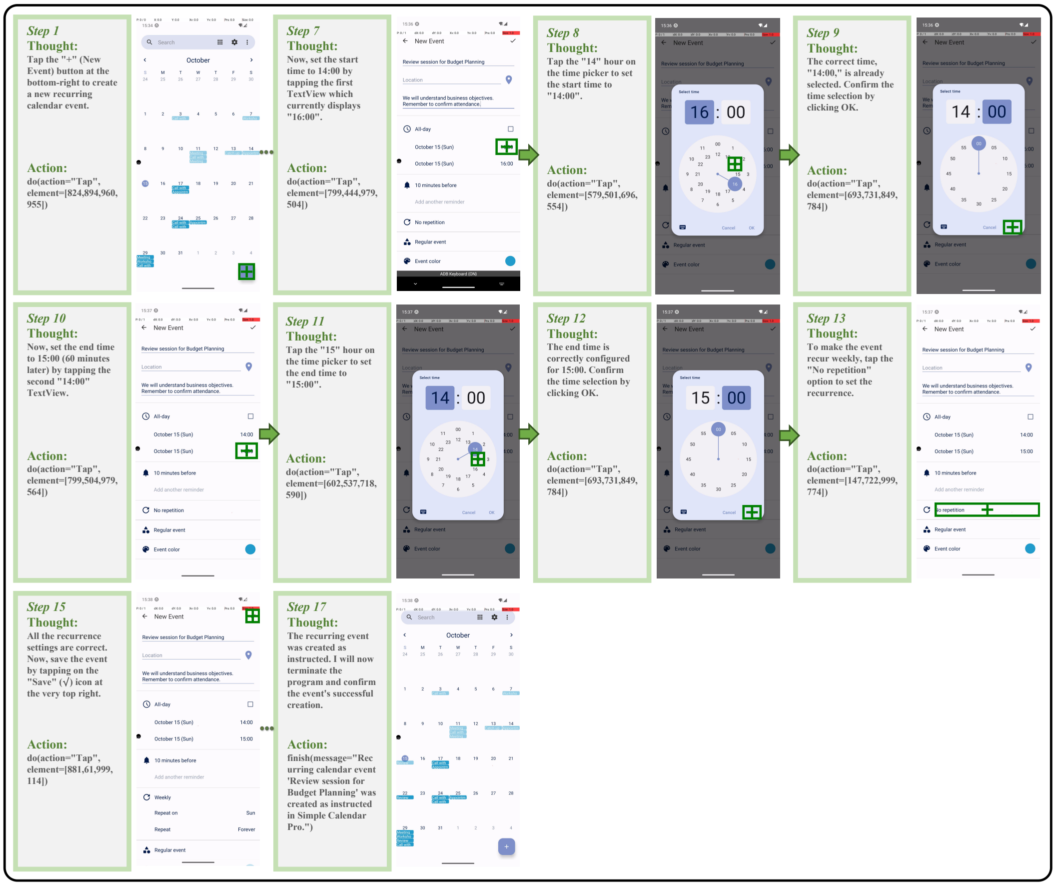


Figure 11: MOBILERL in the `SimpleCalendarAddRepeatingEvent` task in AndroidWorld. Because of space constraints, we only present the most critical and notable steps for comparison.

C.2 System Prompt for Trace Evaluation

The following system prompt guides the VLM in determining whether an agent has successfully completed a task.

```
You are an expert in determining whether a task
has been successfully and completely completed. You will receive:

1. The task description.
2. Step-by-step page states in XML format.
3. The agent's action descriptions.
4. A single image containing screenshots of all steps.
   - Green rectangles and crosses mark tap regions and positions.
   - Green arrows indicate swipe directions.
   - Red text shows typed input.

Action formats and meanings:
- do(action="Tap", element=[x1,y1,x2,y2])
    Tap on the specified screen region; green cross is the tap point.
- do(action="Launch", app="xxx")
    Launch the specified app.
- do(action="Type", text="xxx")
    Enter the specified text (shown in red).
- do(action="Swipe", element=[x1,y1,x2,y2], direction="x", dist="x")
```

Table 3: Action Space for Mobile GUI Interaction. We utilize the action space from AndroidLab (Xu et al., 2024), which represents screen positions with bounding boxes aligned to XML data. Unlike other works (Qin et al., 2025; Rawles et al., 2024), we exclude the "press home" action due to its variability across device models.

Action	Parameters	Description
Tap	`element=[x1,y1,x2,y2]`	Tap at the rectangle defined by top-left (x1,y1) and bottom-right (x2,y2).
Type	`text={string}`	Enter the given string into the focused input field.
Swipe	`direction={up/down/left/right}` `dist={short/medium/long}` `element=[x1,y1,x2,y2] (optional)`	Swipe in the given direction over the specified distance. Optionally constrain to the rectangle element.
Long Press	`element=[x1,y1,x2,y2]`	Press and hold on the given rectangle area.
Launch	`app={AppName}`	Launch the named application.
Back	none	Press the system Back button.
Finish	`message={string} (optional)`	End the session with an optional message.

```
    Swipe in the indicated direction; green arrow shows the swipe path.
- do(action="Long_Pres", element=[x1,y1,x2,y2])
    Long press on the specified region.
- do(action="Back")
    Navigate back to the previous screen.
- finish(message="xxx")
    End the task with the given message.

Scoring rules:
If the task is fully and correctly completed,
output a score of 1; otherwise, 0.

Output format:
<analysis>
Step 1 analysis: <Your analysis>
Step 2 analysis: <Your analysis>
...
Final step analysis: <Your analysis>
</analysis>
<ans>
[Your score]
</ans>
```

C.3 Analysis

We compared the training curves of AndroidWorld and AndroidLab (trained simultaneously), as shown in Figure 12. The training curve of AndroidWorld is smoother compared to that of AndroidLab, which we attribute to the use of a rule-based reward in AndroidWorld, whereas AndroidLab relies on a reward model. However, the reward model cannot fully match the accuracy of the rule-based reward. Therefore, we believe that evaluating whether a broader range of tasks has been successfully completed without relying on rule-based rewards remains an important challenge for future work.

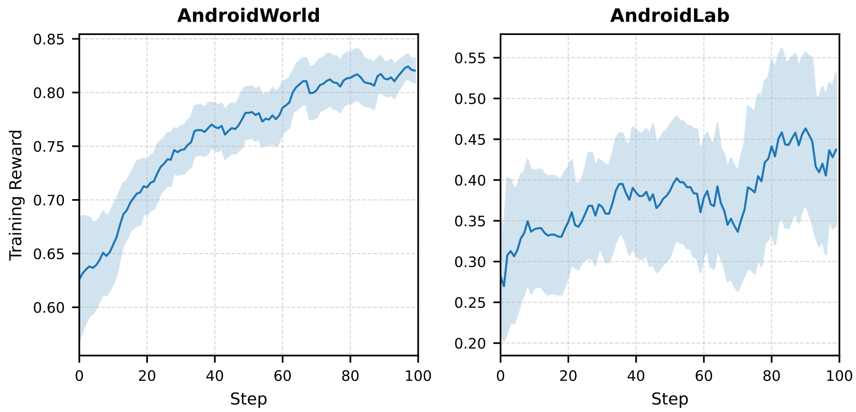


Figure 12: Trajectory-level rewards of MOBILERL with 95% CIs on training sets, showing consistent performance growth.

D XML Preprocessing for UI Representation

The original XML from the Android accessibility service defines the layout and elements of the user interface, including all components on a page. As a result, it contains many nodes used solely for structural or layout purposes, which do not provide useful semantic information. Moreover, scrollable pages often contain more nodes than are visible on the screen, leading to the inclusion of many off-screen nodes.

D.1 Removal of Off-Screen Nodes

We first determine whether to retain off-screen nodes via the input parameter `remain_nodes`:

- **`remain_nodes=True`**: Off-screen nodes are preserved, e.g., when summarizing the full page content without requiring scrolling.
- **`remain_nodes=False`**: Off-screen nodes are removed to avoid interference during action simulation (e.g., tapping, scrolling).

In the original XML, a node is considered on-screen if its `bounds` property lies entirely within the screen dimensions $[0, 0]$ to $[Window\_Height, Window\_Width]$ and is contained by its parent node. We check this condition recursively to identify on-screen nodes.

D.2 Removal of Redundant Nodes

Nodes that do not convey functional or semantic information are removed. A node is considered *functional* if it satisfies at least one of the following:

- Any of the boolean attributes is `True`: `checkable`, `checked`, `clickable`, `focusable`, `scrollable`, `long-clickable`, `password`, `selected`.
- The `text` or `content-desc` attribute is non-empty.

All nodes failing these criteria are considered redundant and are deleted.

D.3 Attribute Simplification

Attribute descriptions in the original XML are verbose and token-expensive. We simplify them as follows:

- Keep only `True` values for the boolean functional attributes listed above (omit `False` values).
- Remove `index`, `resource-id`, and `package` (not useful for semantic understanding).

- For `class`, retain only the last component (e.g., `android.widget.FrameLayout` → `FrameLayout`).
- Merge `text` and `content-desc` attributes and display them separately.
- Retain `bounds` in full, as it indicates the node's position on the page.

D.4 Example Transformation

Original node:

```
<node index="0" text="Audio Recorder"
      resource-id="com.dimowner.audiorecorder:id/txt_title"
      class="android.widget.TextView" package="com.dimowner.audiorecorder"
      content-desc="" checkable="false" checked="false" clickable="false"
      enabled="true" focusable="false" focused="false" scrollable="false"
      long-clickable="false" password="false" selected="false"
      bounds="[221,1095][858,1222]" />
```

Simplified node:

```
TextView;;Audio Recorder;[221,1095][858,1222]
```