SWIRL: A STAGED WORKFLOW FOR INTERLEAVED REINFORCEMENT LEARNING IN MOBILE GUI CONTROL

Quanfeng Lu[1*], **Zhantao Ma**[1*], **Shuai Zhong**[1], **Jin Wang**[1], **Dahai Yu**[3], **Michael K. Ng**[2], **Ping Luo**[1†]
[1]The University of Hong Kong [2]Hong Kong Baptist University
[3]TCL Corporate Research (Hong Kong) Co., Ltd
https://github.com/Lqf-HFNJU/SWIRL

ABSTRACT

The rapid advancement of large vision language models (LVLMs) and agent systems has heightened interest in mobile GUI agents that can reliably translate natural language into interface operations. Existing single-agent approaches, however, remain limited by structural constraints. Although multi-agent systems naturally decouple different competencies, recent progress in multi-agent reinforcement learning (MARL) has often been hindered by inefficiency and remains incompatible with current LVLM architectures. To address these challenges, we introduce SWIRL, a staged workflow for interleaved reinforcement learning designed for multi-agent systems. SWIRL reformulates MARL into a sequence of single-agent reinforcement learning tasks, updating one agent at a time while keeping the others fixed. This formulation enables stable training and promotes efficient coordination across agents. Theoretically, we provide a stepwise safety bound, a cross-round monotonic improvement theorem, and convergence guarantees on return, ensuring robust and principled optimization. In application to mobile GUI control, SWIRL instantiates a Navigator that converts language and screen context into structured plans, and an Interactor that grounds these plans into executable atomic actions. Extensive experiments demonstrate superior performance on both high-level and low-level GUI benchmarks. Beyond GUI tasks, SWIRL also demonstrates strong capability in multi-agent mathematical reasoning, underscoring its potential as a general framework for developing efficient and robust multi-agent systems.

1 INTRODUCTION

With the rapid progress of large vision–language models (LVLMs) (OpenAI, 2025; Zhu et al., 2025; Bai et al., 2025; Guo et al., 2025), increasing attention has been devoted to mobile GUI agents capable of translating natural language instructions into reliable interface manipulation (Qin et al., 2025; Xu et al., 2024; Wu et al., 2024b; Lu et al., 2024; Luo et al., 2025). These agents ground user instructions in the current screenshot and interaction history, reason over this evolving state, and iteratively generate the next action until the task is completed. Effective mobile GUI control depends on two key competencies: task planning, which forms global, goal-conditioned decisions under evolving contexts, and task execution, which translates these plans into executable actions with precise localization. Most existing systems adopt a single-agent design, which complicates the robust integration of these competencies.

We identify two fundamental challenges. First, coupling high-level planning with fine-grained perception and precise actuation makes single end-to-end policies prone to interference and brittleness (Wang et al., 2024; Erdogan et al., 2025; Mo et al., 2025; Wang et al., 2025). Second, end-to-end systems often exhibit a weak linkage between reasoning traces and executed actions, sometimes producing correct outcomes for spurious reasons or plausible traces paired with faulty actions (Turpin

*Equal Contribution
†Corresponding Author: pluo@cs.hku.hk

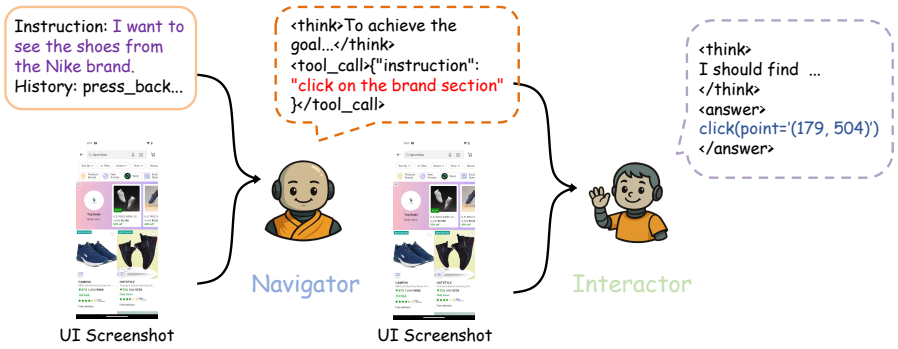


Figure 1: Our multi-agent inference pipelines. Given a high-level instruction, UI screenshots, and historical actions, the Navigator generates a low-level instruction, which the Interactor then uses together with UI screenshots to produce the final executable action. This design decouples task planning from execution, enabling specialization, and leverages explicit intermediate instructions to enhance transparency and interpretability.

et al., 2023; Arcuschin et al., 2025; Li et al., 2024a), thereby undermining safety and accountability in assurance-critical applications (Zhang et al., 2025; Shi et al., 2025; Kuntz et al., 2025).

A multi-agent design provides a principled approach to decoupling core competencies by assigning planning and execution to specialized agents. Beyond this division of labor, structured interactions between agents further enhance transparency by making the reasoning process more interpretable and the resulting actions more auditable. This explicit linkage between decision-making and execution not only improves accountability but also facilitates supervision and error analysis, which are essential for building reliable GUI control systems. However, training-free adaptation of generic LVLMs rarely suffices for domain-specific GUI control due to insufficient cooperation (Niu et al., 2024). Naive multi-agent reinforcement learning (MARL) further introduces practical challenges: joint optimization of multiple policies inflates compute budgets and limited capabilities (Wang et al., 2022; Gogineni et al., 2023). Meanwhile, high-throughput reinforcement learning (RL) frameworks developed for LVLMs are almost exclusively engineered for single-agent training (Hu et al., 2024; Sheng et al., 2025), making them ill-suited for MARL (Liao et al., 2025). These limitations motivate a central question: *can we train multi-agent systems that are resource-friendly while ensuring provable stability during training?*

We introduce SWIRL, a staged workflow for interleaved reinforcement learning. SWIRL decomposes multi-agent training into two phases: independent pre-warming of each module, followed by alternating optimization where one module is frozen while the other is updated. During this alternating process, we further incorporate an online reweighting mechanism to enhance training stability and accelerate convergence. SWIRL in Fig. 2(c) updates exactly one agent at a step. After several updates on one agent, it then switches to the next, turning joint optimization into sequential single-agent problems and enabling reuse of standard RL toolchains in agent training while maintaining effective coordination. Beyond practicality, we offer theoretical and system-level benefits: we establish a per-step safety bound, prove monotonic improvement across rounds, and derive a corollary for return convergence. In implementation, SWIRL requires only the currently updated agent to be resident on the training device, yielding $O(1)$ actor memory usage, smooth compatibility with standard stacks, and support for heteroge-

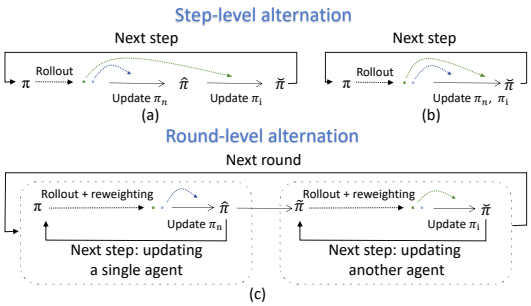


Figure 2: Alternating training paradigm. (a) HAPPO (Zhong et al., 2024): step-level single-agent sequential updates; (b) A2PO (Wang et al., 2023) & MARFT (Liao et al., 2025): enhancing sequential updates with preceding-agent off-policy correction for greater efficiency; (c) SWIRL: round-level alternation with inner solves.

neous model sizes and update budgets, contrasting with the $O(N)$ memory usage of other methods. Table 1 details the count of actor parameters loaded during training without extra optimizations.

We instantiate SWIRL on mobile GUI control with a dual-agent architecture. The Navigator interprets instructions, interaction history, and the current screenshot to form a task context and produce structured low-level instructions (LLI). The Interactor consumes the LLI together with the current UI view and outputs atomic actions such as click, scroll, and text input with precise localization. Fig. 1 presents the inference pipeline. Training alternates between the two agents: when optimizing the Navigator, the Interactor is fixed and executes the Navigator's instructions to yield actions and rewards; when optimizing the Interactor, the Navigator remains fixed and supplies instructions for each step. This instantiation separates planning from execution and enforces a tight linkage between reasoning and action. With this decoupled design and the stability of interleaved updates, our system attains state-of-the-art zero-shot performance on both high-level and low-level mobile GUI benchmarks using only $3{,}500$ training examples, outperforming baselines trained under diverse SFT and RL regimes. We further apply the same interleaved recipe to a mathematics setting following prior multi-agent work (Liao et al., 2025) and observe significant gains, including a $+14.8$ improvement on MATH500.

Table 1: Comparison of actor parameters loaded during training.

Method	Actor parameters
HAPPO	$\sum_{i=1}^{N} \lvert\theta_i\rvert$
A2PO	$\sum_{i=1}^{N} \lvert\theta_i\rvert$
MARFT	$\sum_{i=1}^{N} \lvert\theta_i\rvert$
SWIRL (Ours)	$\max\{\lvert\theta_i\rvert\}$

The contributions of this paper are as follows: (i) we introduce SWIRL, a multi-agent training framework that interleaves single-agent updates and transforms MARL to a sequence of single-agent RL problems; it achieves $O(1)$ actor memory usage by loading only the currently updated agent, and accommodates heterogeneous model sizes, data schedules, and update budgets; (ii) we establish formal guarantees, including a per step safety bound, a monotonic improvement result across rounds, and convergence of returns; (iii) we instantiate SWIRL for mobile GUI control with a Navigator and an Interactor and, through extensive experiments, show stable training and state-of-the-art zero-shot performance, together with ablations that clarify when interleaving helps; and (iv) we demonstrate transferability by applying the same training recipe to a non-GUI domain (e.g., mathematics) and observe consistent gains on standard benchmarks, indicating potential for broader multi-agent applications.

2 Related Work

Reinforcement Learning for Mobile GUI Control Reinforcement learning (RL) has recently emerged as a promising paradigm for GUI tasks. Unlike supervised fine-tuning (SFT), which requires large-scale annotated data, RL can learn effective policies from comparatively fewer samples while exhibiting stronger generalization to new tasks (Chu et al., 2025). A number of recent studies have investigated RL for GUI grounding, with research directions ranging from reward function design to policy optimization (Lu et al., 2025; Luo et al., 2025; Liu et al., 2025; Zhou et al., 2025; Yuan et al., 2025; Tang et al., 2025; Lee et al., 2025). By contrast, applications of RL to more complex mobile control scenarios that involve executing coarse-grained natural language instructions remain relatively limited (Luo et al., 2025). Furthermore, existing work has predominantly adopted single-agent settings, leaving multi-agent approaches largely unexplored.

Multi-Agent Systems Based on Large Language Models A parallel line of research explores multi-agent systems powered by LLMs to address complex tasks (Hu et al., 2025; Xiao et al., 2024; Xiang et al., 2025; Zhao et al., 2024b; Wu et al., 2024a; Zhao et al., 2024a; Du et al., 2023). These systems typically assign specialized roles such as debating, voting, or negotiation, thereby structuring interactions and facilitating coordination without training. To enhance robustness and mitigate distribution shift (Han et al., 2024), recent studies have proposed strategies including persuasion-aware training (Stengel-Eskin et al., 2024) and iterative self-improvement via SFT (Subramaniam et al., 2025; Zhao et al., 2025). An important challenge is how to effectively train model cooperation when only limited training data is available (Tran et al., 2025).

Multi-Agent Reinforcement Learning Works in Multi-Agent Reinforcement Learning (MARL) largely falls into value-based methods (Sunehag et al., 2017; Rashid et al., 2020) and actor-critic

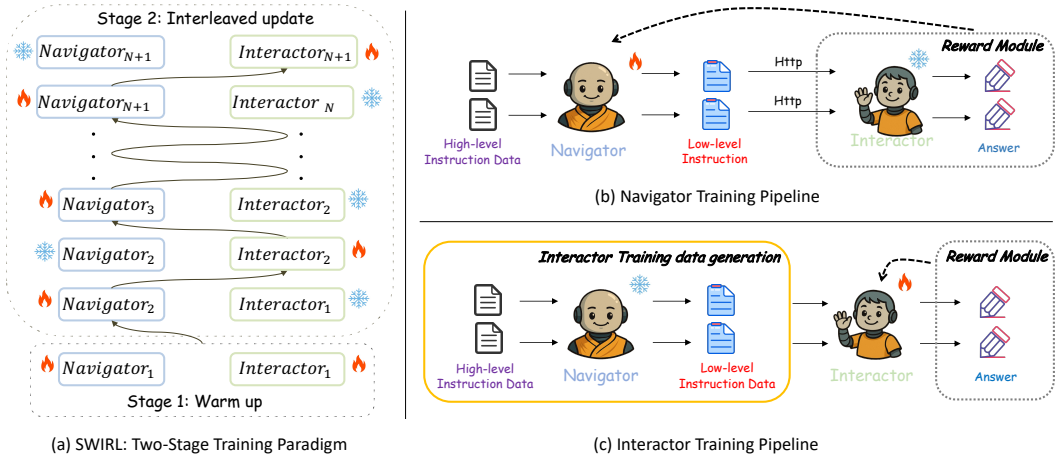


Figure 3: Our multi-agent training pipeline. (a) SWIRL decomposes multi-agent learning into two stages. Stage 1 performs warm-up, where each agent module (Navigator and Interactor) is initialized independently. Stage 2 proceeds with interleaved updates, where optimization alternates between agents: one module is updated while the other is frozen. (b) Given high-level instruction training data, the Navigator generates low-level instructions and obtains rewards by making HTTP calls to a reward module that includes the Interactor (indicated by the dashed box). (c) First, the Navigator generates training data (low-level instructions) for the Interactor (indicated by the orange box), which are then used to train the Interactor.

approaches (Chu et al., 2019; De Witt et al., 2020). A core challenge is non-stationarity, as one agent's update changes others' observations. Alternating optimization (e.g., A2PO (Wang et al., 2023), HARL (Zhong et al., 2024)) update agents sequentially at the step level but still face scalability issues (Canese et al., 2021; Tran et al., 2025). MARFT (Liao et al., 2025) combines MARL with LLMs for mathematical problem but suffers from gradient conflicts and parameter drift as the scale increases. It further argues that unifying MARL and LLMs is harder than addressing either alone, highlighting the need for scalable frameworks to integrate them efficiently.

3 METHOD

In Sec. 3.1, we introduce a theoretical multi-agent interleaved updating methodology in Alg. 1 and give the theoretical guarantees. In Sec. 3.2, we formulate the multi-agent framework for GUI navigation. Following that, Sec. 3.3 introduces SWIRL, a practical mobile GUI implementation of the multi-agent interleaved updating method, executed as a two-phase process involving warm-up and round-level alternating RL with online reweighting.

3.1 PRELIMINARY OF INTERLEAVED UPDATING

The integration of MARL and LVLMs poses challenges, as each introduces unique complexities while being closely interlinked. Similar frictions have long been recognized in mathematical optimization. The Alternating Direction Method of Multipliers (ADMM) (Boyd et al., 2011) provides an effective strategy for tackling challenges formed by complex coupled problems: it decomposes an interconnected problem into manageable subproblems that are solved in turns. This divide-and-conquer approach is widely used in optimization, especially advantageous for challenging non-convex or non-differentiable objectives that can be decomposed into feasible subproblems (Glowinski, 2014; Yang et al., 2022). Consider the following constrained optimization problem $\min_{x,y\in\mathbb{R}^n} f(x) + g(y)$ s.t. $Ax + By = c$ and its augmented Lagrangian form: $\mathcal{L}_\rho(x, y, \lambda) = f(x)+g(y)+\lambda^\top(Ax+By-c)+\frac{\rho}{2}\|Ax+By-c\|^2$. To solve this, ADMM performs alternating optimization via the following iterative process: $x^{k+1} := \arg\min_x \mathcal{L}_\rho(x, y^k, \lambda^k); y^{k+1} := \arg\min_y \mathcal{L}_\rho(x^{k+1}, y, \lambda^k); \lambda^{k+1} := \lambda^k + Ax^{k+1} + By^{k+1} - c$. It exemplifies problem decompo-

Algorithm 1: Multi-Agent Interleaved Updating with Monotonic Improvement Guarantee

Initialise independent pre-warming $\pi_0^i,\ 1 \le i \le n$;
for *round* $k = 0, 1, 2, \ldots$ **do**
  for *agent* $i = 1, \ldots, n$ **do**
    Initialize $\pi_{k,0}^i \leftarrow \pi_k^i$;
    for *micro-step* $j = 0, \ldots, K_i - 1$ **do**
      $\pi_{k,j+1}^i \leftarrow \arg\max_{\pi^i} F_{k,i,j}(\pi^i) = \left[L^i_{\Pi_{k,i,j}}(\tau_k^{-i}, \pi^i) - C_{k,i,j} D^{\max}_{\mathrm{KL}}(\pi^i_{k,j}, \pi^i)\right]$;
    Update $\pi_{k+1}^i \leftarrow \pi_{k,K_i}^i$;

sition and alternating optimization: rather than tackling a complex problem, alternating between simpler subproblems can achieve the overall objective.

From ADMM to Multi-Agent Interleaved Updating. In practical ADMM, each subproblem is solved by an inner loop to a prescribed accuracy before the outer iteration advances. The key is that these inner steps deliver enough improvement for the outer objective to make steady progress. Therefore, we propose a multi-agent round-level interleaved updating training scheme in Alg. 1. The algorithm first independently pre-warms each agent using any single-agent method to obtain initial policies. It then performs interleaved updates: one agent is continuously optimized while the others are fixed, then sequentially switches to the next agent until all are updated, repeating this cycle to decompose MARL into a sequence of single-agent optimization tasks. This structure leads to the following findings: Proposition 1 provides a safety bound for each micro-step, Theorem 1 confirms that the return increases consistently across rounds, and Corollary 1 says the returns converge, and all policy limits attain this value. A summary of notation can be found in Appendix A.1, with complete proofs located in Appendix A.2.

Proposition 1 (Lower bound at a micro-step)**.** *In round k, if agent $\pi^i_{k,j}$ updates to $\pi^i_{k,j+1}$, the new joint policy is $\Pi_{k,i,j+1} := (\tau_k^{-i},\ \pi^i_{k,j+1})$, and the performance satisfies*

$$J(\Pi_{k,i,j+1}) \ \ge\ J(\Pi_{k,i,j}) \ +\ L^i_{\Pi_{k,i,j}}\big(\tau_k^{-i},\ \pi^i_{k,j+1}\big) -\ C_{k,i,j}\, D^{\max}_{\mathrm{KL}}\big(\pi^i_{k,j},\ \pi^i_{k,j+1}\big). \tag{1}$$

Theorem 1 (Monotonic improvement)**.** *Every micro-step updates in Alg. 1 satisfies $F_{k,i,j}(\pi^i_{k,j+1}) \ge 0$, and for all outer rounds k we have $J(\pi_{k+1}) \ge J(\pi_k)$.*

Corollary 1 (Return convergence)**.** *The sequence $\{J(\pi_k)\}$ approaches a limit, referred to as $\bar{J}$, and the collection of limit points from $\{\pi_k\}$ is non-empty. For any subsequence $\{\pi_{k_j}\}_{j\ge 0}$ that converges such that $\pi_{k_j} \to \bar{\pi}$, it holds that $J(\bar{\pi}) = \bar{J}$.*

Alg. 1 provides a methodology: in each round, a single active agent performs several micro-steps while all other agents are frozen. Since the baseline stays constant throughout these micro-steps, each step simplifies to a typical single-agent policy update. As a result, the surrogate goal $L - D^{\max}_{\mathrm{KL}}$ can be estimated using well-established single-agent techniques like TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017), and GRPO (Shao et al., 2024), which apply feasible trust region or clipped KL updates. Previous studies have verified both the theoretical validity and empirical effectiveness of this approximation (Schulman et al., 2015; Zhong et al., 2024). We then apply this methodology to the GUI navigation task.

3.2 Multi-Agent Framework for GUI Control

Task Formulation. We formulate GUI control task as a sequential decision-making problem. With a natural language instruction I, the agent reviews a series of historical screenshots and actions $H_t = \{X_{t-\delta_s}, \ldots, X_{t-1}, r^a_{t-\delta_a}, \ldots, r^a_{t-1}\}$ along with the current screenshot X_t to craft a structured text reply r_t at each time step t. Here, δ_s and δ_a denote the counts of past screenshots and actions, respectively. This reply includes a reasoning (r^r_t) and a low-level instruction (LLI, r^i_t) that outlines the next planned step. The LLI is then succeeded by the actual action (r^a_t), where $r_t \sim \pi_{\theta_n,\theta_i}(r \mid I, H_t, X_t),\ \{r^r_t, r^i_t, r^a_t\} \in r_t$. The objective is to generate the next action r^a_t that complies with the given instruction I. Appendix E presents illustrative examples of GUI agents

completing GUI control tasks. Nevertheless, navigating GUI-based instructions introduces specific challenges: it necessitates high-level navigation to deduce the next-step instruction and detailed perception to engage with UI components, each demanding distinct skills.

Architecture and Training Objective. To proficiently manage the complexities of GUI operations, we utilize a multi-agent system that distinctly separates the responsibilities between the *Navigator* (π_{θ_n}) and the *Interactor* (π_{θ_i}). The *Navigator* is responsible for high-level planning, where it interprets the natural language instructions, merges past actions with the UI views, and establishes a coherent task context with reasoning. It then creates a detailed LLI, reflecting the intended next actions, based on the reasoning process. Subsequently, the *Interactor* combines the LLI and the current UI view to generate concrete atomic actions, including actions like click, scroll, etc. This involves precise cursor positioning and visual interpretation to ensure accurate execution of the planned steps within the interface. The inference pipeline of the system is illustrated in Fig. 1.

Building on the system architecture above, we train the two agents with a round-level interleaved scheme (Alg. 1). In each round, we select one role, either the Navigator or the Interactor, and run multiple inner updates on its parameters while freezing the other agent. We then swap roles and repeat. Optimizing the theoretical update for an individual agent demands the calculation of advantage A, surrogate L, and $D_{\mathrm{KL}}^{\max}$, which is cost-prohibitive. Consequently, we implement practical relaxations, similar to (Zhong et al., 2024), by approximating the theoretical goal using a GRPO objective (Shao et al., 2024). Concretely, we calculate single-agent improvements using *group-relative advantages*, denoted as A_k, which are derived from multiple rollouts (standardized across the batch). We control intractable $D_{\mathrm{KL}}^{\max}$ with two tractable terms: clipped ratios around π_{old} and a KL anchor to π_{ref}, ensuring local trust-region control and curbing drift for stable. This preserves the ascent direction of the theoretical target $L - D_{\mathrm{KL}}^{\max}$, whilst ensuring stability and efficiency. Each agent's action is an autoregressive sequence, responses for rollouts K are $\{r_{k,\ell}\}_{1\le k\le K}^{1\le \ell\le |r_k|}$. We compute token-wise importance ratios aligned with GRPO, in line with (Luo et al., 2025). Each token is assigned to either the navigator or interactor by the indicator $\mathbb{I}_{k,\ell}^{(j)} = 1$ if and only if $r_{k,\ell} \sim \pi_{\theta_j}$ (and 0 otherwise), securing agent-specific credit assignment without breaching the "freeze-the-complement" rule. In conclusion, our overarching multi-agent training objective is:

$$\mathcal{J}(\theta_n, \theta_i) = \mathbb{E}_{\substack{r\sim\pi \\ (I,H_t,X_t)\sim\mathcal{D}}} \left[\sum_{k,\ell} \sum_{j\in\{n,i\}} \frac{\mathbb{I}_{k,\ell}^{(j)}}{K \sum_{\ell=1}^{|r_k|} \mathbb{I}_{k,\ell}^{(j)}} \cdot \left(\mathrm{clip}(v_{k,\ell}^{(j)}, A_k) - \lambda D_{\mathrm{KL}}[\pi_{\theta_j} \| \pi_{\theta_j}^{\text{ref}}] \right) \right]. \quad (2)$$

The clipped surrogate is: $\mathrm{clip}(v, A) = \min\left(vA,\ \mathrm{clip}(v, 1-\epsilon, 1+\epsilon)A\right)$, where the value of importance ratio is: $v_{k,\ell}^{(j)} = \frac{\pi_{\theta_j}(r_{k,\ell}|I,H_t,X_t,r_{k,<\ell})}{\pi_{\theta_j}^{\text{old}}(r_{k,\ell}|I,H_t,X_t,r_{k,<\ell})}$. The scalar reward is composed of: $R_k = \alpha R_{\text{form}} + \beta R_{\text{acc}}$, where R_{form} denotes the reward for format correctness (e.g., proper usage of required tags), and R_{acc} is the reward for action accuracy, defined as $R_{\text{acc}} = \lambda_1 R_{\text{act}} + \lambda_2 R_{\text{info}}$, where R_{act} measures the correctness of the predicted action type, and R_{info} quantifies the accuracy of the action parameters (e.g., the click location). Finally, the normalized advantage is computed as $A_k = \frac{R_k - \mu}{\sigma}$, where μ and σ are the mean and standard deviation of rewards across sampled trajectories.

3.3 SWIRL: Staged Workflow for Interleaved Reinforcement Learning

Building on the multi-agent architecture and learning objective described above, we introduce **SWIRL**, a Staged Workflow for Interleaved Reinforcement Learning. This approach is crafted to efficiently coordinate and enhance the performance of both the Navigator and Interactor concurrently, as demonstrated in Fig. 3 and Alg. 2. The benefits of SWIRL are listed in Appendix. B.2.

Stage 1: Warm-up initialization. The Navigator is first initialized through lightweight Chain-of-Thought (Wei et al., 2022) SFT, while the interactor is bootstrapped via initial reinforcement learning. The primary goal of this stage is to let each agent clearly learn its designated role: the navigator focuses on producing reasoning steps and LLI, whereas the interactor outputs the corresponding action. This warm-up phase establishes a robust foundation and minimizes variability at the outset.

Stage 2: Interleaved update. After warm-up, SWIRL progresses into a round-level alternating training stage. Within each round, one agent undergoes continuously optimization via reinforcement learning, while the other is kept static. This approach simplifies the intertwined multi-agent learning challenge into a series of static single-agent tasks. Therefore, it allows us to directly leverage modern single-agent RL algorithms, such as GRPO (see Alg. 3), thereby decreasing implementation complexity while preserving training stability and cooperative efficiency. In the Navigator training process, the Navigator generates a reply $r = \{r^r, r^i\}$ at each step. The frozen Interactor then executes r^i to produce the final action r^a. Rewards are computed as described in Sec. 3.2, with the accuracy reward of r^a serving as the Navigator's R_{acc}. To further enhance training efficiency and scalability, we deploy the Interactor on a separate server and integrate it within the reward module. The Navigator communicates with the Interactor via HTTP requests, enabling efficient reward computation during RL optimization. For Interactor training, we first use the frozen Navigator to generate low-level instructions for each training sample. The Interactor is then optimized via RL using these instructions as input, with rewards also computed according to Sec. 3.2. The training pipelines for the Navigator and Interactor are illustrated in Fig. 3b and Fig. 3c, respectively.

Online reweighting. After computing the GRPO-type advantages for each batch, we exclude those low-quality samples (Meng et al., 2025; Cui et al., 2025) by regulation $\mathcal{R}$. These low-quality samples typically arise due to collaborator mistakes, noise, or simplicity (Shi et al., 2025). As the model's performance increases, it might happen that the number of high-quality samples meeting our filtering standard becomes fewer than the batch size, leading to partially filled batches. To address this, we replenish the batch by randomly resampling from the remaining high-confidence instructions. This process produces reliable batches without introducing additional rollout cost, implicitly up-weights the informative samples to enhance training stability and convergence.

Algorithm 2: SWIRL: Staged Workflow for Interleaved Reinforcement Learning

Input: Dataset $\mathcal{D}$, hyperparameters N, μ_n, μ_i;

Stage 1: Warm-up initialization.

Initialize navigator $\pi_{\theta_n}^{(1)}$ via supervised fine-tuning on instruction-action pairs;

Initialize interactor $\pi_{\theta_i}^{(1)}$ via RL using fixed planner outputs;

Stage 2: Interleaved reinforcement learning.

for $k = 1$ **to** N **do**

Navigator update: freeze $\pi_{\theta_i}^{(k)}$, update π_{θ_n} using RL in Alg. 3:

$$\theta_n^{(k+1)} \leftarrow RL(\theta_n \mid \mathcal{J}, \theta_n^{(k)}, \theta_i^{(k)}, \mu_n);$$

Interactor update: freeze $\pi_{\theta_n}^{(k+1)}$, update π_{θ_i} using RL in Alg. 3:

$$\theta_i^{(k+1)} \leftarrow RL(\theta_i \mid \mathcal{J}, \theta_n^{(k+1)}, \theta_i^{(k)}, \mu_i);$$

4 EXPERIMENT

The experimental setup is described in Sec. 4.1, and the complete experimental details are provided in Appendix C. Sec. 4.2 and Sec. 4.3 present evaluations of SWIRL's zero-shot performance on high-level and low-level tasks, respectively. Sec. 4.4 investigates the proposed multi-agent training framework in the mathematics domain. Comprehensive ablation studies are reported in Appendix D.

4.1 SETTINGS

Implementation Details. We use Qwen2.5-VL-3B (Bai et al., 2025) as the base model for both the Navigator and Interactor agents. For historical context, we include only the sequence of past actions, excluding previous screenshots (i.e., $\delta_a = t - 1$, $\delta_s = 0$). The weighting coefficients in the reward function in Sec. 3.2 are set to $\alpha = 0.1$, $\beta = 0.9$, $\lambda_1 = 0.2$, and $\lambda_2 = 0.8$. For online

Table 2: Performance comparison on the AndroidControl-High and GUIOdyssey datasets. **Bold** and underline indicate the best and second-best performers, respectively.

Models	Method	AndroidControl-High			GUIOdyssey			Overall
		Type	GR	SR	Type	GR	SR	
GPT-4o	Closed-Source	63.06	30.90	21.17	37.50	14.17	5.36	28.69
OS-Atlas-4B	SFT	49.01	49.51	22.77	49.63	34.63	20.25	37.63
OS-Atlas-7B	SFT	57.44	54.90	29.83	60.42	39.74	26.96	44.88
UI-R1-3B	RFT	43.97	63.99	26.30	20.93	56.35	8.65	36.70
UI-R1-E-3B	RFT	29.67	61.50	14.37	7.59	62.87	1.94	29.66
GUI-R1-3B	RFT	58.04	56.24	46.55	54.84	41.52	41.33	49.75
GUI-R1-7B	RFT	**71.63**	65.56	**51.67**	65.49	43.64	38.79	56.13
ReGUIDE-7B	RFT	–	–	50.00	–	–	–	–
GPT-5 / UI-R1-3B	Multi-Agent	55.08	73.47	37.27	62.97	62.42	35.93	54.52
GPT-5 / UI-R1-E-3B	Multi-Agent	60.36	**74.79**	40.65	65.28	63.41	38.22	57.12
GPT-5 / Interactor	Multi-Agent	64.68	74.62	49.53	68.77	62.70	44.21	60.75
Navigator / Interactor	SWIRL	66.72	71.19	51.24	**74.87**	**66.39**	**51.65**	**63.68**

Table 3: Performance comparison on web and desktop low-level tasks. The best and second-best in each column are indicated by **bold** and underline, respectively.

Models	GUI-Act-Web			OmniAct-Web			OmniAct-Desktop			Overall
	Type	GR	SR	Type	GR	SR	Type	GR	SR	
GPT-4o	77.09	45.02	41.84	79.33	42.79	34.06	79.97	63.25	50.67	57.11
OS-Atlas-4B	79.22	58.57	42.62	46.74	49.24	22.99	63.30	42.55	26.94	48.02
OS-Atlas-7B	86.95	75.61	57.02	85.63	69.35	59.15	90.24	62.87	56.73	71.51
UI-R1-3B	75.89	79.43	67.31	75.42	61.35	61.33	73.41	64.12	63.98	69.14
GUI-R1-3B	89.86	87.42	76.31	88.58	75.10	75.08	91.86	78.37	78.31	82.32
GUI-R1-7B	90.85	**88.06**	80.31	91.16	77.29	**77.35**	92.20	**83.36**	**83.33**	84.88
Interactor	**95.00**	87.85	**84.85**	**94.52**	**81.67**	77.32	**94.65**	77.09	72.97	**85.10**

reweighting, we define the rule $\mathcal{R}$ as: $\mathcal{R} = \begin{cases} \text{keep}, & 0.1 < \overline{R_x} < 1, \\ \text{discard}, & \text{otherwise.} \end{cases}$, where $\overline{R_x}$ denotes the average reward of sample x across rollouts.

Training. We collected $1,500$ and $2,000$ mobile-control samples for Stage 1 and Stage 2, respectively (see Appendix C.1 for details). In Stage 1 (warm-up), the Navigator is trained for 1 epoch with SFT and the Interactor for 5 epochs with GRPO (Shao et al., 2024), both using a learning rate of 1×10^{-6}, batch size 32, and DeepSpeed ZeRO-1 for the Navigator; the Interactor uses 5 rollouts per sample. In Stage 2 (SWIRL alternating updates), the two agents are alternately trained for 2 epochs each per round over 20 rounds, maintaining the same hyperparameters; the Navigator uses 8 rollouts per sample and the Interactor 5. Stage 1 runs on $8\times$ NVIDIA A800 GPUs; Stage 2 uses $16\times$ A800s, with half for Interactor deployment as a vLLM-based (Kwon et al., 2023) inference service and half for training. The training framework builds on Qwen2.5-VL[1] (Bai et al., 2025) for SFT and VeRL[2] (Sheng et al., 2025) for RL.

Evaluation. We evaluate SWIRL on high-level tasks (AndroidControl-High (Li et al., 2024b), GUIOdyssey (Lu et al., 2024)) and low-level tasks (AndroidControl-Low (Li et al., 2024b), GUI-Act (Chen et al., 2024), OmniAct (Kapoor et al., 2024)). Following the previous work (Wu et al., 2024b; Luo et al., 2025), we report Type, GR, and SR in a zero-shot prompt setting to assess out-of-domain generalization. Appendix C.2 provides detailed information. All evaluations are conducted in a zero-shot prompt setting to assess the models' out-of-domain generalization ability.

[1] https://github.com/QwenLM/Qwen2.5-VL/tree/main/qwen-vl-finetune
[2] https://github.com/volcengine/verl

Table 5: Math results for training-time multi-agent frameworks. SWIRL and MARFT share the same base model Qwen2.5-Coder-3B-Instruct, and are both trained on the MATH training set.

Model	Method	Training Data	MATH500	CMATH	GSM8K	Overall
Qwen2.5-Coder-3B-Instruct	Multi-Agent	–	47.2	81.1	77.3	68.5
MARFT	Multi-Agent	MATH	49.8	83.0	78.7	70.5
SWIRL (ours)	Multi-Agent	MATH	**64.6**	**83.5**	**81.4**	**76.5**

4.2 High-Level Task Zero-shot Performance

We benchmark our approach against models trained under different paradigms and observe substantial gains. As shown in Table 2, our multi-agent framework with two 3B models achieves state-of-the-art zero-shot performance, surpassing the SFT-trained OS-Atlas-7B(Wu et al., 2024b) by 18.8 points and the RFT-trained GUI-R1-7B (Luo et al., 2025) by 7.55 points. When GPT-5 (OpenAI, 2025) is used as a high-level to low-level planner for UI-R1-3B and UI-R1-E-3B (Lu et al., 2025), the scores increase by 17.82 and 24.86 points, confirming the advantage of separating planning from execution. Replacing GPT-5 with our Navigator trained using SWIRL and interleaved updates yields a further improvement of 2.93 points, demonstrating the effectiveness of our training strategy in enhancing coordination and boosting downstream performance.

4.3 Low-Level Task Zero-shot Performance

The Interactor trained with SWIRL interleaved updates can also operate independently as a low-level task executor. Although its low-level instruction inputs during Stage 2 training are generated by the Navigator, it still achieves strong results, recording the highest SR and GR scores (78.81 and 92.20) and the second-highest Type score (84.62) on the AndroidControl-Low benchmark (Li et al., 2024b), as shown in Table 4. Notably, despite all 3,500 training samples across both stages originating exclusively from mobile devices, the model achieves an overall score of 85.10 on low-level tasks in the web- and desktop-based GUI-Act (Chen et al., 2024) and Omni-Act (Kapoor et al., 2024) datasets, representing the best average performance (Table 3). This remarkable cross-domain generalization underscores the robustness of our model and provides strong empirical evidence for the effectiveness of our training methodology.

Table 4: Performance comparison on the AndroidControl-Low dataset. **Bold** and underline indicate the best and second-best performers, respectively.

Model	AndroidControl-Low		
	Type	GR	SR
GPT-4o	74.33	38.67	28.39
OS-Atlas-4B	64.58	71.19	40.62
OS-Atlas-7B	73.00	73.37	50.94
UI-R1	72.49	88.48	57.37
UI-R1-E	73.91	<u>91.91</u>	55.58
GUI-R1-3B	83.58	81.59	64.41
GUI-R1-7B	**85.17**	84.02	66.52
ReGUIDE-7B	–	–	67.40
SE-GUI-7B	–	79.60	<u>68.20</u>
Interactor	<u>84.62</u>	**92.20**	**78.81**

4.4 Multi-Agent Training Framework in the Mathematics Domain

We adapt SWIRL to the mathematics domain to validate the generalizability and transferability of our multi-agent training framework. In line with the prior framework MARFT (Liao et al., 2025), we employ Qwen2.5-Coder-3B-Instruct (Hui et al., 2024) as the base model to construct a dual-agent architecture, which is then trained on the MATH (Hendrycks et al., 2021) training set. Implementation details are provided in Appendix C.3. As shown in Table 5, SWIRL delivers consistent improvements across all benchmarks, achieving a 14.8-point gain over MARFT on the in-domain MATH500 test set, and further surpassing it on the cross-domain CMATH (Wei et al., 2023) (83.5 vs. 83.0) and out-of-domain GSM8K (Cobbe et al., 2021) (81.4 vs. 78.7) datasets. These results demonstrate that SWIRL not only excels in its original GUI mobile control domain but also transfers effectively to substantially different problem settings, highlighting its robustness and broad applicability.

5 CONCLUSION

In this paper, we presented SWIRL, an interleaved reinforcement learning paradigm that reformulates multi-agent training into tractable single-agent updates. The central principle of SWIRL lies in its round-level alternating training strategy, where in each round one agent is continuously optimized through reinforcement learning while the others remain fixed. We provided theoretical guarantees for stable optimization and validated the effectiveness of SWIRL through extensive experiments in both mobile GUI control and multi-agent mathematical reasoning. Looking forward, we hope that SWIRL can inspire new approaches to multi-agent training in broader domains, such as finance and AI for science, where efficient coordination and reliable optimization remain critical challenges.

REFERENCES

Iván Arcuschin, Jett Janiak, Robert Krzyzanowski, Senthooran Rajamanoharan, Neel Nanda, and Arthur Conmy. Chain-of-thought reasoning in the wild is not always faithful. *arXiv preprint arXiv:2503.08679*, 2025.

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.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11):4948, 2021.

Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Dingyu Zhang, Peng Gao, Shuai Ren, and Hongsheng Li. Amex: Android multi-annotation expo dataset for mobile gui agents. *arXiv preprint arXiv:2407.17490*, 2024.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, et al. Guicourse: From general vision language models to versatile gui agents. *arXiv preprint arXiv:2406.11317*, 2024.

Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE transactions on intelligent transportation systems*, 21(3): 1086–1095, 2019.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2023.

Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*, 2025.

Roland Glowinski. On alternating direction methods of multipliers: a historical perspective. *Modeling, simulation and optimization for science and technology*, pp. 59–82, 2014.

Kailash Gogineni, Peng Wei, Tian Lan, and Guru Venkataramani. Towards efficient multi-agent learning systems. *arXiv preprint arXiv:2305.13411*, 2023.

Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, et al. Seed1. 5-vl technical report. *arXiv preprint arXiv:2505.07062*, 2025.

Shanshan Han, Qifan Zhang, Yuhang Yao, Weizhao Jin, and Zhaozhuo Xu. Llm multi-agent systems: Challenges and open problems. *arXiv preprint arXiv:2402.03578*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Jian Hu, Xibin Wu, Zilin Zhu, Weixun Wang, Dehao Zhang, Yu Cao, et al. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, et al. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation. *arXiv preprint arXiv:2505.23885*, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem AlShikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *European Conference on Computer Vision*, pp. 161–178. Springer, 2024.

Thomas Kuntz, Agatha Duzan, Hao Zhao, Francesco Croce, Zico Kolter, Nicolas Flammarion, and Maksym Andriushchenko. Os-harm: A benchmark for measuring safety of computer use agents. *arXiv preprint arXiv:2506.14866*, 2025.

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

Hyunseok Lee, Jeonghoon Kim, Beomjun Kim, Jihoon Tack, Chansong Jo, Jaehong Lee, Cheonbok Park, Sookyo In, Jinwoo Shin, and Kang Min Yoo. Reguide: Data efficient gui grounding via spatial reasoning and search. *arXiv preprint arXiv:2505.15259*, 2025.

Jiachun Li, Pengfei Cao, Yubo Chen, Jiexin Xu, Huaijun Li, Xiaojian Jiang, Kang Liu, and Jun Zhao. Towards better chain-of-thought: A reflection on effectiveness and faithfulness. *arXiv preprint arXiv:2405.18915*, 2024a.

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*, 2024b.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. Marft: Multi-agent reinforcement finetuning. *arXiv preprint arXiv:2504.16129*, 2025.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*, 2025.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *arXiv preprint arXiv:2406.08451*, 2024.

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. *arXiv preprint arXiv:2503.21620*, 2025.

Run Luo, Lu Wang, Wanwei He, and Xiaobo Xia. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

Fanqing Meng, Lingxiao Du, Zongkai Liu, Zhixiang Zhou, Quanfeng Lu, Daocheng Fu, Tiancheng Han, Botian Shi, Wenhai Wang, Junjun He, Kaipeng Zhang, Ping Luo, Yu Qiao, Qiaosheng Zhang, and Wenqi Shao. Mm-eureka: Exploring the frontiers of multimodal reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2503.07365*, 2025.

Fanglin Mo, Junzhe Chen, Haoxuan Zhu, and Xuming Hu. Building a stable planner: An extended finite state machine based planning module for mobile gui agent. *arXiv preprint arXiv:2505.14141*, 2025.

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

OpenAI. Gpt4o, 2024.

OpenAI. Introducing gpt-5. `https://openai.com/index/introducing-gpt-5/`, 2025. Accessed: August 2025.

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

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.

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.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 1279–1297, 2025.

Yucheng Shi, Wenhao Yu, Wenlin Yao, Wenhu Chen, and Ninghao Liu. Towards trustworthy gui agents: A survey. *arXiv preprint arXiv:2503.23434*, 2025.

Elias Stengel-Eskin, Peter Hase, and Mohit Bansal. Teaching models to balance resisting and accepting persuasion. *arXiv preprint arXiv:2410.14596*, 2024.

Vighnesh Subramaniam, Yilun Du, Joshua B Tenenbaum, Antonio Torralba, Shuang Li, and Igor Mordatch. Multiagent finetuning: Self improvement with diverse reasoning chains. *arXiv preprint arXiv:2501.05707*, 2025.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Fei Tang, Zhangxuan Gu, Zhengxi Lu, Xuyang Liu, Shuheng Shen, Changhua Meng, Wen Wang, Wenqi Zhang, Yongliang Shen, Weiming Lu, et al. Gui-g ^{2}: Gaussian reward modeling for gui grounding. *arXiv preprint arXiv:2507.15846*, 2025.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025.

Miles Turpin, Julian Michael, Ethan Perez, and Samuel Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems*, 36:74952–74965, 2023.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, et al. Gui agents with foundation models: A comprehensive survey. *arXiv preprint arXiv:2411.04890*, 2024.

Xihuai Wang, Zhicheng Zhang, and Weinan Zhang. Model-based multi-agent reinforcement learning: Recent progress and prospects. *arXiv preprint arXiv:2203.10603*, 2022.

Xihuai Wang, Zheng Tian, Ziyu Wan, Ying Wen, Jun Wang, and Weinan Zhang. Order matters: Agent-by-agent policy optimization. *arXiv preprint arXiv:2302.06205*, 2023.

Ziwei Wang, Weizhi Chen, Leyang Yang, Sheng Zhou, Shengchu Zhao, Hanbei Zhan, Jiongchao Jin, Liangcheng Li, Zirui Shao, and Jiajun Bu. Mp-gui: Modality perception with mllms for gui understanding. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 29711–29721, 2025.

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

Tianwen Wei, Jian Luan, Wei Liu, Shuang Dong, and Bin Wang. Cmath: Can your language model pass chinese elementary school math test? *arXiv preprint arXiv:2306.16636*, 2023.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024a.

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

Violet Xiang, Charlie Snell, Kanishk Gandhi, Alon Albalak, Anikait Singh, Chase Blagden, Duy Phung, Rafael Rafailov, Nathan Lile, Dakota Mahan, et al. Towards system 2 reasoning in llms: Learning how to think with meta chain-of-thought. *arXiv preprint arXiv:2501.04682*, 2025.

Yijia Xiao, Edward Sun, Di Luo, and Wei Wang. Tradingagents: Multi-agents llm financial trading framework. *arXiv preprint arXiv:2412.20138*, 2024.

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

Yu Yang, Xiaohong Guan, Qing-Shan Jia, Liang Yu, Bolun Xu, and Costas J Spanos. A survey of admm variants for distributed optimization: Problems, algorithms and features. *arXiv preprint arXiv:2208.03700*, 2022.

Xinbin Yuan, Jian Zhang, Kaixin Li, Zhuoxuan Cai, Lujian Yao, Jie Chen, Enguang Wang, Qibin Hou, Jinwei Chen, Peng-Tao Jiang, et al. Enhancing visual grounding for gui agents via self-evolutionary reinforcement learning. *arXiv preprint arXiv:2505.12370*, 2025.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*, 2024.

Li Zhang, Longxi Gao, and Mengwei Xu. Does chain-of-thought reasoning help mobile gui agent? an empirical study. *arXiv preprint arXiv:2503.16788*, 2025.

Jun Zhao, Can Zu, Hao Xu, Yi Lu, Wei He, Yiwen Ding, Tao Gui, Qi Zhang, and Xuanjing Huang. Longagent: scaling language models to 128k context through multi-agent collaboration. *arXiv preprint arXiv:2402.11550*, 2024a.

Wanjia Zhao, Mert Yuksekgonul, Shirley Wu, and James Zou. Sirius: Self-improving multi-agent systems via bootstrapped reasoning. *arXiv preprint arXiv:2502.04780*, 2025.

Xiutian Zhao, Ke Wang, and Wei Peng. An electoral approach to diversify llm-based multi-agent collective decision-making. *arXiv preprint arXiv:2410.15168*, 2024b.

Yifan Zhong, Jakub Grudzien Kuba, Xidong Feng, Siyi Hu, Jiaming Ji, and Yaodong Yang. Heterogeneous-agent reinforcement learning. *Journal of Machine Learning Research*, 25(32): 1–67, 2024.

Yuqi Zhou, Sunhao Dai, Shuai Wang, Kaiwen Zhou, Qinglin Jia, and Jun Xu. Gui-g1: Understanding r1-zero-like training for visual grounding in gui agents. *arXiv preprint arXiv:2505.15810*, 2025.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

A PRELIMINARY AND PROOFS

A.1 PRELIMINARY WITH NOTATION

Environment and policies. We consider a cooperative Markov game $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}, r, P, d \rangle$ with n agents: $\mathcal{N} = [n] = \{1, \dots, n\}$. Let $\mathcal{A} := \prod_{i \in \mathcal{N}} \mathcal{A}^i$ be the joint action space. For each $i \in \mathcal{N}$, the (stochastic Markov) policy $\pi^i(\cdot \mid s) \in \Delta(\mathcal{A}^i)$ together form the joint policy $\pi = (\pi^i)_{i \in \mathcal{N}}$, which induces the joint action distribution $\pi(a \mid s) = \prod_{i \in \mathcal{N}} \pi^i(a^i \mid s)$ for $a = (a^i)_{i \in \mathcal{N}}$. Here $\Delta(\cdot)$ denotes the set of probability distributions over a set. The environment transitions according to $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, written $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, with initial state $s_0 \sim d$ and joint reward $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Let ρ_π be the average state-visitation distribution induced by the joint policy π. For agent-wise decomposition we can write $a = (a^{-i}, a^i)$ and denote $\pi^{-i} = (\pi^j)_{j \neq i}$.

Joint return $J(\pi)$. For discount $\gamma \in [0, 1)$ and any fixed initial-state distribution,

$$J(\pi) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, a_t) \right].$$

It represents the expected total reward from this point forward under the policy π: summing the reward from each subsequent step and applying a discount factor of γ^t at step t. The notation ∞ indicates "assess each future step"; in tasks with a finite horizon, this sum naturally concludes upon task completion. The discount factor γ dictates the temporal range: $\gamma = 0$ focuses only on immediate rewards, suitable for offline tasks, where $J(\pi) = \mathbb{E}_\pi[r(s_0, a_0)]$, whereas a higher value of γ emphasizes rewards further in the future.

Rounds, order, and micro-steps. Outer rounds are indexed by $k = 0, 1, 2, \dots$. Agent i executes a block of K_i *micro-steps* indexed by $j = 0, \dots, K_i - 1$, and we denote a micro-step by (k, i, j). The joint policy at the start (resp. end) of round k is π_k (resp. π_{k+1}). During agent i's block, $\pi^i_{k,j}$ is its temporary policy after j micro-steps, initialized by $\pi^i_{k,0} = \pi^i_k$; after finishing the block, set $\pi^i_{k+1} := \pi^i_{k,K_i}$. Other agents are held fixed according to the rolling baseline defined next.

Rolling baseline and complement policy. The baseline joint policy at micro-step (k, i, j) is

$$\Pi_{k,i,j} := \big(\{\pi^r_{k+1}\}_{r<i}, \, \pi^i_{k,j}, \, \{\pi^r_k\}_{r>i} \big) = \big(\tau^{-i}_k, \, \pi^i_{k,j} \big),$$

where the *complement policy* (all agents except i) is

$$\tau^{-i}_k := \big(\{\pi^r_{k+1}\}_{r<i}, \, \{\pi^r_k\}_{r>i} \big).$$

During micro-step j of agent i in the kth round, agents positioned earlier in the sequence ($r < i$) have already adopted their policies for round $(k+1)$, the current agent follows its internal iterate $\pi^i_{k,j}$, and those positioned later ($r > i$) continue to use their round-k policies.

Value, Q, and advantages. Define

$$V_\pi(s) := \mathbb{E}_\pi \Big[\sum_{t \geq 0} \gamma^t r(s_t, a_t) \,\Big|\, s_0 = s \Big], \qquad Q_\pi(s, a) := \mathbb{E}_\pi \Big[\sum_{t \geq 0} \gamma^t r(s_t, a_t) \,\Big|\, s_0 = s, \, a_0 = a \Big].$$

The joint advantage is $A_\pi(s, a) := Q_\pi(s, a) - V_\pi(s)$.

For agent i, define the marginal state–action value and the single-agent advantage by

$$Q^i_\pi(s, a^i) := \mathbb{E}_{a^{-i} \sim \pi^{-i}(\cdot \mid s)} \big[\, Q_\pi\big(s, (a^{-i}, a^i)\big) \, \big], \qquad A^i_\pi(s, a^i) := Q^i_\pi(s, a^i) - V_\pi(s).$$

$A^i_\pi(s, a^i)$ is the local (per-state/per-action) improvement if agent i takes a^i at s while others act according to π^{-i}. A basic identity we use is the zero-mean property $\mathbb{E}_{a^i \sim \pi^i(\cdot \mid s)} \big[A^i_\pi(s, a^i) \big] = 0$ for every s.

Surrogate improvement. Given the baseline $\Pi_{k,i,j}$, the complement τ_k^{-i}, and a candidate policy $\hat{\pi}^i$ for agent i, define

$$L^i_{\Pi_{k,i,j}}(\tau_k^{-i}, \hat{\pi}^i) := \mathbb{E}_{\substack{s\sim\rho_{\Pi_{k,i,j}} \\ a^i\sim\hat{\pi}^i(\cdot|s)}} \Big[A^i_{\Pi_{k,i,j}}(s, a^i) \Big].$$

L aggregates the local signal A^i into a policy-level surrogate by averaging. It satisfies the baseline-zero property $L^i_{\Pi_{k,i,j}}(\tau_k^{-i}, \pi^i_{k,j}) = 0$, because $\mathbb{E}_{a^i\sim\pi^i_{k,j}(\cdot|s)}[A^i_{\Pi_{k,i,j}}(s, a^i)] = 0$ for all s.

Micro-step objective. Let $\varepsilon_{k,i,j} := \max_{s,a} |A_{\Pi_{k,i,j}}(s, a)|$ and $C_{k,i,j} := \frac{4\gamma\,\varepsilon_{k,i,j}}{(1-\gamma)^2}$. Define the max conditional KL by $D^{\max}_{\mathrm{KL}}(\mu, \nu) := \sup_s \mathrm{KL}\big(\mu(\cdot \mid s) \,\|\, \nu(\cdot \mid s)\big)$, and the per-micro-step surrogate

$$F_{k,i,j}(\hat{\pi}^i) \;:=\; L^i_{\Pi_{k,i,j}}(\tau_k^{-i}, \hat{\pi}^i) \;-\; C_{k,i,j}\, D^{\max}_{\mathrm{KL}}\big(\pi^i_{k,j}, \hat{\pi}^i\big).$$

$F_{k,i,j}$ scores a candidate update as "local surrogate improvement L minus a KL safety penalty". Taking the current iterate as the candidate reveals that it possesses the baseline-zero property: $F_{k,i,j}(\pi^i_{k,j}) = 0$. This is because L equals zero due to the zero-mean property of the single-agent advantage under the policy $\pi^i_{k,j}$, and $D^{\max}_{\mathrm{KL}}(\mu, \mu) = 0$.

A.2 Theoretical Proofs

Proposition 1. In round k, if agent $\pi^i_{k,j}$ updates to $\pi^i_{k,j+1}$, the new joint policy is $\Pi_{k,i,j+1} := (\tau_k^{-i},\ \pi^i_{k,j+1})$, and the performance satisfies

$$J(\Pi_{k,i,j+1}) \;\geq\; J(\Pi_{k,i,j}) \;+\; L^i_{\Pi_{k,i,j}}\big(\tau_k^{-i},\, \pi^i_{k,j+1}\big) \;-\; C_{k,i,j}\, D^{\max}_{\mathrm{KL}}\big(\pi^i_{k,j},\, \pi^i_{k,j+1}\big). \tag{3}$$

Proof of Proposition 1. Set old $= \Pi_{k,i,j}$ and new $= \Pi_{k,i,j+1} = (\tau_k^{-i},\, \pi^i_{k,i,j+1})$. Apply Lemma 6 in (Zhong et al., 2024) with $\pi =$ old and $\bar{\pi} =$ new. Because only agent i changes, the multi-agent surrogate in Lemma 6 reduces to $L^i_{\Pi_{k,i,j}}(\tau_k^{-i},\, \pi^i_{k,j+1})$, and the max-conditional KL reduces to $D^{\max}_{\mathrm{KL}}(\pi^i_{k,j},\, \pi^i_{k,j+1})$. With $C_{k,i,j} = \frac{4\gamma\,\varepsilon_{k,i,j}}{(1-\gamma)^2}$ and $\varepsilon_{k,i,j} = \max_{s,a} |A_{\Pi_{k,i,j}}(s, a)|$, this yields exactly inequality 3. □

Theorem 1. Every micro-step updates in Alg. 1 satisfies $F_{k,i,j}(\pi^i_{k,j+1}) \geq 0$, then for all outer rounds k we have $J(\pi_{k+1}) \geq J(\pi_k)$.

Proof of Theorem 1. Leveraging the baseline-zero characteristic, $F_{k,i,j}(\pi^i_{k,j}) = 0$, it follows that $\arg\max_{\pi^i} F_{k,i,j}(\pi^i) \geq 0$. Therefore, by Proposition 1 and the update rule,

$$J(\Pi_{k,i,j+1}) - J(\Pi_{k,i,j}) \;\geq\; F_{k,i,j}\big(\pi^i_{k,j+1}\big) \;\geq\; 0$$

for every micro-step (k, i, j). Summing these inequalities in order over all micro-steps within round k gives

$$J(\pi_{k+1}) - J(\pi_k) = \sum_{i=1}^{n} \sum_{j=0}^{K_i-1} \big[J(\Pi_{k,i,j+1}) - J(\Pi_{k,i,j})\big] \;\geq\; 0,$$

hence $J(\pi_{k+1}) \geq J(\pi_k)$ for all k. □

Corollary 1. The sequence $\{J(\pi_k)\}$ has a limit, denoted as $\bar{J}$, and the set comprised of limit points of $\{\pi_k\}$ is not empty. Additionally, for any convergent subsequence $\{\pi_{k_j}\}_{j\geq 0} : \pi_{k_j} \to \bar{\pi}$, $J(\bar{\pi}) = \bar{J}$.

Proof of Corollary 1. By Theorem 1, the performance sequence $\{J(\pi_k)\}_{k\geq 0}$ is nondecreasing. With discount $\gamma \in [0, 1)$ and bounded rewards $|r| \leq R_{\max}$, every policy satisfies $|J(\pi)| \leq R_{\max}/(1-\gamma)$, so $\{J(\pi_k)\}$ is bounded above and converges to some $\bar{J} \in \mathbb{R}$. Furthermore, as in (Zhong et al., 2024), the sequence of policies is bounded, hence it admits a convergent subsequence by the Bolzano-Weierstrass Theorem. Therefore, the set of limit points of $\{\pi_k\}$ is nonempty. Let $(\pi_{k_j})_{j\geq 0}$ be any subsequence converging to a limit policy $\bar{\pi}$. By continuity of J in π, we have

$$J(\bar{\pi}) \;=\; J\left(\lim_{j\to\infty} \pi_{k_j}\right) \;=\; \lim_{j\to\infty} J(\pi_{k_j}) \;=\; \bar{J}.$$

□

B Details in Method

B.1 Illustration of Algorithm 3

In Alg. 3, $RL(\theta \mid \mathcal{J}, \cdot, \cdot, \mu)$ denotes GRPO-type single agent RL based on the objective $\mathcal{J}$ defined in equation 2 and hyperparameter μ, where the first argument θ is the parameter to be updated.

- When updating the *navigator*, the interactor parameters are frozen. In this case, the input θ_n serves as the initial setting for the navigator, whereas θ_{frozen} pertains to the fixed interactor:

$$RL(\theta \mid \mathcal{J}, \theta_n, \theta_{\text{frozen}}, \mu).$$

- When updating the *interactor*, the navigator parameters are frozen. In this case, the input θ_i serves as the initial setting for the interactor, whereas θ_{frozen} pertains to the fixed navigator:

$$RL(\theta \mid \mathcal{J}, \theta_{\text{frozen}}, \theta_i, \mu),$$

Below, we take the case of freezing the interactor and updating the navigator as an example.

Algorithm 3: GRPO-Type policy optimization $RL(\theta \mid \mathcal{J}, \theta_n, \theta_{\text{frozen}}, \mu)$

```
// Here we take the case of freezing the interactor and
   updating the navigator as an example.
```

Input: Dataset $\mathcal{D}$, current policy θ_n, frozen model θ_{frozen}, hyperparameters $\mu = (M, B, K)$;
Initialize $\theta \leftarrow \theta_n$;
for *iteration* $= 1 \dots M$ **do**
  Initialize reference policy $\theta_{\text{ref}} \leftarrow \theta$;
  for *step* $= 1 \dots B$ **do**
    Sample a batch $\mathcal{D}_b$ from $\mathcal{D}$;
    Update the old policy $\theta_{\text{old}} \leftarrow \theta$;
    Sample K outputs $\{r_k\}_{k=1}^{K} \sim \pi_{\theta_{\text{old}}, \theta_{\text{frozen}}}(\cdot \mid I, H_t, X_t)$;
    Compute rewards $\{R_k\}_{k=1}^{K}$ via reward model;
    Compute GRPO-type advantages: $A_k \leftarrow \frac{R_k - \mu}{\sigma}$ over the group;
    Online reweighting: discard samples with low quality and resample to refill the batch;
    Update the policy model by maximizing the objective: $\theta \leftarrow \arg\max_\theta \mathcal{J}(\theta, \theta_{\text{frozen}})$;
return *Updated parameters* θ*;*

B.2 Benefits of SWIRL

Different from MARL methods such as HAPPO (Zhong et al., 2024) (Fig. 2a), A2PO (Wang et al., 2023), and MARFT (Liao et al., 2025) (Fig. 2b), our approach (Fig. 2c) offers four key advantages:

1. **Seamless compatibility.** SWIRL reformulates complex MARL tasks into an alternating sequence of simpler single-agent reinforcement learning problems. This design naturally integrates with modern distributed single-agent RL frameworks (e.g., veRL (Sheng et al., 2025), OPEN-RLHF (Hu et al., 2024)), eliminating intrusive modifications to communication protocols and enabling rapid adaptation to future efficient RL frameworks. Consequently, our divide-and-conquer alternating solution fundamentally resolves the integration challenges between MARL and contemporary LVLM-based training pipelines.
2. **Resource-friendly scalability.** Consider a system with N agents, each with model size $|\theta_i|$. Table 1 quantitatively summarizes the number of actor parameters loaded on the device during training. In practice, A2PO, HAPPO, and MARFT keep all agents resident on the training device for local rollout. Under this setting, these methods typically load all actor parameters in the training device during learning, leading to a total parameter size of $\sum_{i=1}^{N} |\theta_i|$ and memory consumption that scales linearly with the number of agents ($O(N)$). In contrast, SWIRL only loads the currently updated agent model locally, while executing the remaining agents as "Model-as-a-Service" modules remotely. As a result, the memory usage of the training device remains constant regardless of N (i.e., $O(1)$), greatly reducing hardware requirements and improving scalability for large-scale multi-agent systems.

3. **Adaptation to heterogeneity.** Our method allows agents to adopt diverse model architectures and training configurations, such as different numbers of training steps or heterogeneous datasets. Because SWIRL decomposes multi-agent training into per-agent optimization, it enables online, agent-conditioned reweighting that adapts to each agent's distribution by filtering samples that are low-quality for that agent. As a result, heterogeneous agents receive tailored, high-confidence training signals; experiments on Appendix D.5 empirically confirms reduced uninformative or misleading samples and improved overall training performance.
4. **Stability.** The alternating update methodology efficiently addresses the non-stationary challenges present in concurrent multi-agent training, thereby minimizing policy misalignment and distributional shift, as shown in Appendix D.2 and D.5. Experimental results in Fig. 4 consistently demonstrate stable performance enhancements with each alternating phase, thus facilitating effective coordination among multiple agents.

C Experiment Details

C.1 Training Data

Training Data Collection. We construct our two-stage training dataset based on the AITW(Rawles et al., 2023) (using the expanded version from AITZ(Zhang et al., 2024)) and AMEX(Chai et al., 2024) (adopting the Aguvis variant(Xu et al., 2024)), as both provide richer low-level annotations. In Stage 1, we leverage Qwen2.5-VL-3B to generate 8 rollouts for each low-level instruction, then select $1,500$ samples whose average reward falls within the range $[0.3, 0.4]$. These high-quality samples are used in two ways: (1) the low-level instruction data is employed as RL training data for the Interactor, and (2) the reasoning traces associated with each sample are used to construct Chain-of-Thought supervised fine-tuning data with high-level instructions for the Navigator. In Stage 2, we utilize the multi-agent system trained in Stage 1 to generate 8 rollouts for each high-level instruction, filtering for $2,000$ samples with an average reward below 0.6 and variance more than 0.175. Notably, samples in Stage 2 include only high-level instructions and do not contain additional semantic annotations. In total, we curate $3,500$ high-quality training samples across both stages, supporting robust chain-of-thought reasoning and effective RL optimization for both agents.

Action Space. Following the action space design style of UI-Tars (Qin et al., 2025), we define dataset-specific action spaces for different GUI benchmarks, as shown in Table 6.

Prompt. Fig. 8 illustrates the prompts used for the Navigator and Interactor.

C.2 Evaluation Details

Metrics. Following prior work (Wu et al., 2024b; Luo et al., 2025), we evaluate our models using three standard metrics for GUI agents: Type, GR, and SR, which assess the accuracy of action type prediction, coordinate prediction, and step success rate, respectively. Type measures the exact match between the predicted action type (e.g., `click`, `scroll`) and the ground truth. GR evaluates the performance of GUI grounding in downstream tasks. SR (step-wise success rate) is computed by considering a step correct only if both the predicted action type and all associated arguments (e.g., coordinates for a click action) match the ground truth. For click-based actions (e.g., `click`, `long_press`), the model must predict both the action type and the target coordinates (x, y). When the ground-truth bounding box is available, a prediction is considered correct if its coordinates fall within the box. If no bounding box is provided, or the prediction lies outside it, correctness is determined by whether the predicted coordinates are within 14% of the screen width from the ground-truth position. For type-based actions (e.g., `type`, `open_app`), both the action type and content must match the ground truth. We compute the F1 score between the predicted and reference text, and an action is considered correct if F1 > 0.5. For scroll actions, the predicted direction argument (up, down, left, or right) must exactly match the ground truth. For all other actions (e.g., `press_enter`), correctness requires an exact match between the predicted action and the ground truth.

Baselines. To ensure a fair comparison of cross-domain generalization, we select baseline models that have not been trained on the corresponding training sets of the evaluation domains. For high-level tasks, the comparison includes four categories of models: the proprietary GPT-4o(OpenAI, 2024); the state-of-the-art OS-Atlas-4B/7B(Wu et al., 2024b), trained via supervised fine-tuning (SFT) on large-scale GUI grounding datasets; models trained with reinforcement fine-tuning (RFT) on GUI grounding datasets, including UI-R1-3B, UI-R1-E-3B(Lu et al., 2025), GUI-R1-3B/7B(Luo et al., 2025), and ReGUIDE-7B(Lee et al., 2025); and a multi-agent approach employing GPT-5(OpenAI, 2025) as the planner to translate high-level instructions into low-level actions. For low-level tasks, the baselines consist of GPT-4o, OS-Atlas-4B/7B, and RFT-trained GUI grounding models, including UI-R1-3B, UI-R1-E-3B, GUI-R1-3B/7B, ReGUIDE-7B, and SE-GUI-7B(Yuan et al., 2025).

Evaluation Datasets. **AndroidControl**(Li et al., 2024b) is a mobile control dataset in which each GUI interaction trajectory is annotated with both coarse-grained high-level instructions and fine-grained low-level instructions, along with detailed XML metadata from which the bounding boxes of individual UI elements can be parsed. For click actions, we iterate over all bounding boxes in the XML, identify those containing the ground-truth coordinates, and select the smallest one by area as the candidate bounding box for click action evaluation. **GUIOdyssey**(Lu et al., 2024) consists of tasks involving cross-application operations, posing a significant challenge for models' planning abilities. In its latest release[3], each click action is supplemented with the bounding box of the corresponding UI element obtained via SAM2(Ravi et al., 2024) segmentation, which we also use for click action evaluation. Notably, the latest version contains $8{,}834$ samples, compared to $7{,}735$ in the earlier release[4], with changes in the test set size. To ensure consistency with other baselines, we use the Test-Random split from the earlier version. **GUI-Act-Web**(Chen et al., 2024) contains web-based interaction data. To better assess GUI manipulation capabilities, we remove several QA-style samples from the test set and retain the remaining samples for evaluation. Bounding boxes are not used for click action evaluation in this dataset. **OmniAct**(Kapoor et al., 2024) includes both web and desktop interaction data. Due to action space compatibility constraints, we filter out samples whose original action space involves hotkeys and keep the remaining samples for evaluation. Similarly, bounding boxes are not used for click action evaluation in this dataset.

C.3 Mathematics Domains

Implementation Details. We design a dual-agent framework in which one agent acts as the *Teacher*, providing a concise outline of the problem-solving approach, and the other acts as the *Student*, generating the final solution by incorporating the Teacher's guidance. The detailed prompt formulations for both agents are provided in Fig. 9. Both agents are initialized from the Qwen2.5-Coder-3B-Instruct(Hui et al., 2024) model. For the reward function, we assign a reward of 1 if the generated answer is correct and 0 otherwise, i.e., $R_x = \begin{cases} 1, & \text{if answer is correct,} \\ 0, & \text{otherwise.} \end{cases}$. For online reweighting, we define the rule $\mathcal{R}$ as follows: we retain samples whose average reward across multiple rollouts lies strictly between 0.2 and 0.8, i.e., $\mathcal{R} = \begin{cases} \text{keep}, & 0.2 < \overline{R_x} < 0.8, \\ \text{discard}, & \text{otherwise.} \end{cases}$, where $\overline{R_x}$ denotes the mean reward of sample x over all rollouts.

Training. We train the dual-agent framework on the MATH(Hendrycks et al., 2021) training set, which contains $7,500$ samples, by directly initiating Stage 2 of the SWIRL alternating-update procedure and bypassing the warm-up stage. In each round, the Teacher is updated first, followed by the Student, with each agent trained for 1 epoch per round over a total of 10 rounds. Both agents use a batch size of 128 and a learning rate of 1×10^{-6} . The Teacher and Student perform 4 and 8 rollouts per sample, respectively. Training is conducted on $16\times$ NVIDIA A800 GPUs, with half allocated to deploying the Student as a vLLM-based (Kwon et al., 2023) inference service and the remaining half for model training.

[3] https://huggingface.co/datasets/hflqf88888/GUIOdyssey

[4] https://huggingface.co/datasets/OpenGVLab/GUIOdyssey

Table 6: Action spaces used for each dataset.

Dataset	Action Space

AITW, AMEX	`click(point='(x1, y1)')`
	`type(content='xxx')`
	`scroll(direction='down|up|right|left')`
	`press_home()`
	`press_back()`
	`press_enter()`
	`finished()`

AndroidControl	`click(point='(x1, y1)')`
	`long_press(point='(x1, y1)')`
	`type(content='xxx')`
	`scroll(direction='down|up|right|left')`
	`open_app(app_name='xxx')`
	`press_home()`
	`press_back()`
	`wait()`
	`finished()`

GUIOdyssey	`click(point='(x1, y1)')`
	`long_press(point='(x1, y1)')`
	`type(content='xxx')`
	`scroll(direction='down|up|right|left')`
	`press_home()`
	`press_back()`
	`press_appselect()`
	`error(content='xxx')`
	`finished()`

GUI-Act-Web	`click(point='(x1, y1)')` `scroll(direction='down|up')`

OmniAct-Web)	`click(point='(x1, y1)')`
	`rightclick(point='(x1, y1)')`
	`scroll(direction='down|up')`

OmniAct-Desktop	`click(point='(x1, y1)')`
	`rightclick(point='(x1, y1)')`
	`doubleclick(point='(x1, y1)')`
	`moveto(point='(x1, y1)')`
	`scroll(direction='down|up')`

D Ablation Study

D.1 Effect of Interleaved Update

To isolate the effect of SWIRL's second training stage (interleaved updates), we compare two models trained on the same total dataset of $3,500$ samples. Both start from the Stage 1 warm-up with $1,500$ samples. The first continues training on the remaining $2,000$ samples using the Stage 1 strategy until convergence, while the second proceeds with our Stage 2 alternating update scheme. The only difference between them is the training strategy applied in the second stage. As shown in Table 7, although extended warm-up training with more data yields performance gains, its upper bound remains lower than that achieved by our interleaved update approach (61.69 vs. 63.68), highlighting the latter's effectiveness.

Table 7: Effect of interactive updates. Numbers in *italics* indicate performance gains relative to Stage 1 with $1,500$ samples.

Training Strategy	AndroidControl-High			GUIOdyssey			Overall
	Type	GR	SR	Type	GR	SR	
Stage 1 + Stage 1	63.77 *(+0.90)*	70.25 *(+0.35)*	48.04 *(+1.21)*	72.25 *(+1.52)*	66.19 *(+2.87)*	49.64 *(+3.22)*	61.69 *(+1.68)*
Stage 1 + Stage 2	**66.72** ***(+3.85)***	**71.19** ***(+1.29)***	**51.24** ***(+4.41)***	**74.87** ***(+4.14)***	**66.39*(+3.07)***	**51.65*(+5.23)***	**63.68** ***(+3.67)***

D.2 Co-evolution of Navigator and Interactor

We investigate the impact of the proposed interleaved update on individual agents within the multi-agent framework. For the Interactor, Table 8 compares its performance on low-level tasks with and without Stage 2 interleaved updates. The results show a substantial improvement of 4.89 points with Stage 2, particularly in SR (a metric that directly measures the correctness of individual actions and thus serves as a more critical indicator of the Interactor's precision), which increases by nearly 7 points. For the Navigator, direct evaluation is less straightforward because its outputs are low-level instructions rather than executable actions. To address this, we adopt an indirect evaluation approach: we feed the Navigator's outputs into a fully trained Interactor and assess the resulting task performance. As shown in Table 9, pairing the Interactor with a Navigator trained using Stage 2 interleaved updates delivers consistently higher performance on high-level tasks, raising the overall score from 56.32 to 63.68 and yielding notable improvements in both GR and SR across benchmarks. This demonstrates that the updated Navigator produces more detailed and accurate low-level instructions from high-level goals. Overall, these findings provide strong evidence that interleaved updates effectively enhance each agent's ability to fulfill its specific role, enabling them to co-evolve and achieve better collaborative performance within the multi-agent system.

Table 8: Effect of interleaved updates on Interactor performance in low-level tasks.

Training Strategy	GUI-Act-Web			OmniAct-Web			OmniAct-Desktop			AndroidControl-Low			Overall
	Type	GR	SR	Type	GR	SR	Type	GR	SR	Type	GR	SR	
Stage 1	94.31	**88.22**	77.00	89.22	81.62	72.40	91.38	74.45	68.03	**85.48**	71.91	68.87	80.24
Stage 1 + Stage 2	**95.00**	87.85	**84.85**	**94.52**	**81.67**	**77.32**	**94.65**	**77.09**	**72.97**	84.62	**92.20**	**78.81**	**85.13**

Table 9: Performance of multi-Agent systems on high-level tasks with Navigators trained with and without interleaved updates.

Stage 2 Training	AndroidControl-High			GUIOdyssey			Overall
	Type	GR	SR	Type	GR	SR	
✗	63.16	54.65	39.91	73.19	60.49	46.51	56.32
✓	**66.72**	**71.19**	**51.24**	**74.87**	**66.39**	**51.65**	**63.68**

D.3 Stability and Potential

As illustrated in Fig. 4, even with only $2,000$ samples used during the SWIRL's stage 2, the proposed training paradigm consistently enhances the model's generalization capability. The model's performance improves steadily with each training round, demonstrating the stability of the alternating optimization process. Furthermore, its out-of-domain generalization continues to increase in the later rounds, suggesting that the performance upper bound has not yet been reached. These results highlight the effectiveness and scalability of SWIRL for robust multi-agent training in GUI navigation tasks.

D.4 Synergy and Individual Competence in Multi-Agent Training

In multi-agent training, overall system performance depends not only on enhancing the capabilities of individual agents but also on achieving effective coordination among them. To disentangle and quantify the relative contributions of these two factors, we vary the number of alternating training rounds (i.e., the frequency and intensity of inter-agent coordination) and the number of epochs per

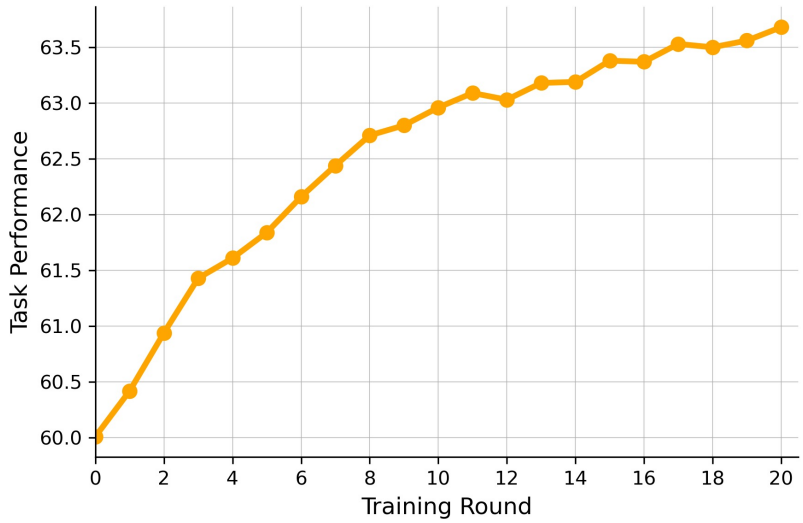


Figure 4: SWIRL training dynamics showing steady performance gains and stability across rounds. Task performance is measured as the average score across all high-level benchmarks, including AndroidControl-High and GUIOdyssey.

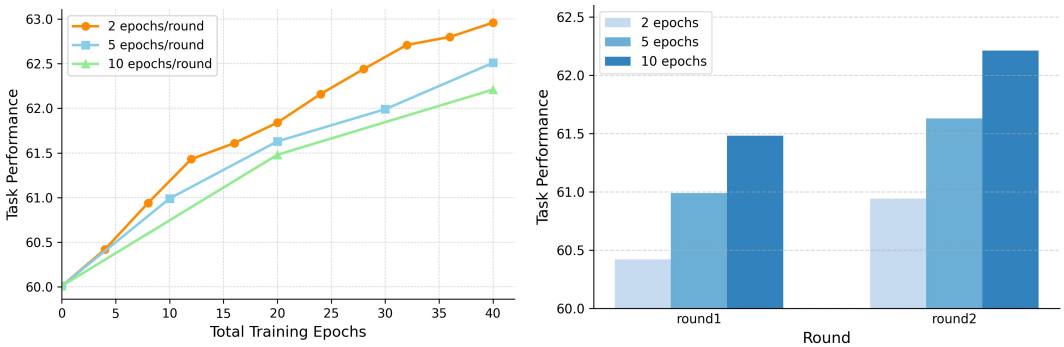
round (i.e., the depth of single-agent training), while keeping the total number of training epochs constant. As shown in Fig.5(a) and Table 10, increasing the number of alternating rounds, thereby providing more opportunities for inter-agent interaction and iterative policy refinement, consistently leads to greater performance gains than allocating the same budget to deeper single-agent training within fewer rounds. At the same time, Fig. 5(b) indicates that increasing the training depth per round still brings incremental benefits, demonstrating that single-agent optimization remains valuable. Taken together, these results suggest that, under a fixed training budget, inter-agent coordination is the primary driver of performance improvement, while individual agent refinement plays a complementary yet meaningful role.

Table 10: Effect of the rounds-epochs schedule under a fixed training budget. Here, 'Epochs/round' denotes the number of epochs trained within each round. The total training epochs are held constant across settings.

Rounds	Epochs/round	AndroidControl			GUIOdyssey			Overall
		Type	GR	SR	Type	GR	SR	
2	10	65.74	70.42	49.88	73.13	64.85	49.27	62.21
5	4	65.26	**70.94**	49.81	73.85	65.21	49.98	62.51
10	2	**66.00**	70.91	**50.56**	**74.03**	**65.71**	**50.58**	**62.96**

D.5 Effectiveness Analysis of Online Reweighting

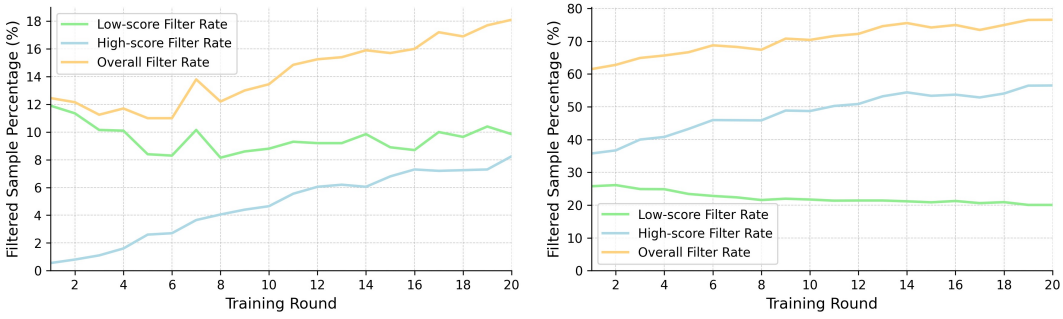
To assess the impact of the online reweighting mechanism, we conduct comparative experiments with and without weighted resampling. As shown in Fig. 7(a), models trained without online reweighting perform significantly worse and even exhibit a degradation trend, underscoring the necessity of this mechanism. We hypothesize that online reweighting dynamically prioritizes informative samples, ensuring that they exert greater influence during training, which is essential for the stable improvement of multi-agent systems. Further analysis, as illustrated in Fig. 6, reveals two important findings. First, the Interactor filters out approximately four times as many samples as the Navigator ($\sim$75 vs. $\sim$18), indicating a substantial difference in how the same dataset contributes to the learning process of each agent. The online reweighting strategy thus adapts to the evolving capabilities of each agent, assigning dynamic weights to those training samples most beneficial at each stage. Second, it is important to recognize that not all samples filtered for being completely incorrect are attributable solely to the deficiencies of a single agent. Some errors may arise from



(a) (b)

Figure 5: Effect of the training schedule on SWIRL performance with varying numbers of rounds and epochs per round. (a) Training curves under a fixed total number of training epochs, comparing different numbers of epochs per round. (b) Performance comparison after each round for different epochs-per-round settings.

noisy data or from mistakes originating with another agent (e.g., if the Planner generates an erroneous instruction, the Executor may repeatedly fail to execute the correct action). In these cases, the online reweighting mechanism helps to exclude uninformative or misleading samples, thereby further improving the robustness and effectiveness of the training process.

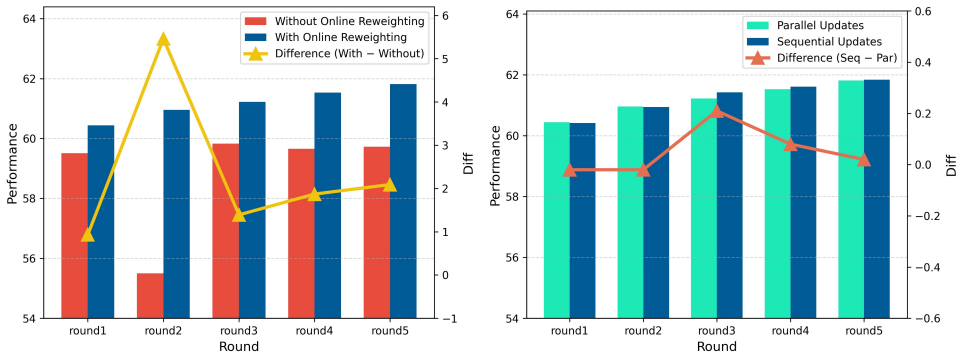


(a) Navigator Training (b) Interactor Training

Figure 6: Filtered sample rates with online reweighting in SWIRL.

D.6 Sequential Updates versus Parallel Updates

Our default SWIRL implementation employs a strictly sequential update scheme, in which the Navigator is updated first in each training round, followed by the Interactor. We also explore a parallel update strategy, where both agents are updated simultaneously within each round, rather than following a fixed order, i.e., $\begin{cases} \theta_n^{(k+1)} \leftarrow RL(\theta_n | \mathcal{J}, \theta_n^{(k)}, \theta_i^{(k)}, \mu_n) \\ \theta_i^{(k+1)} \leftarrow RL(\theta_i | \mathcal{J}, \theta_n^{(k)}, \theta_i^{(k)}, \mu_i) \end{cases}$. This approach aligns with the update mechanism of Jacobian ADMM (Yang et al., 2022), which relaxes the sequential constraints in standard ADMM and allows for simultaneous updates, i.e., $\begin{cases} x^{k+1} := \arg\min_x \mathcal{L}_\rho(x, y^k, \lambda^k) \\ y^{k+1} := \arg\min_y \mathcal{L}_\rho(x^k, y, \lambda^k) \end{cases}$, thereby increasing training efficiency. As shown in Fig. 7(b), parallel updates can achieve performance comparable to or surpassing that of sequential updates, even without strict alternation. This finding suggests that relaxing the sequential constraint in alternating multi-agent training can improve efficiency without sacrificing model quality.



(a) Effect of online reweighting (b) Effect of update order

Figure 7: Ablation results on SWIRL zero-shot performance for AndroidControl-High and GUIOdyssey. (a) Online reweighting vs. no reweighting. (b) Sequential vs. parallel updates. Performance differences are indicated by lines.

Navigator

<img>current_screenshot.png</img>

You are a GUI Planner Agent. Your role is to actively collaborate with the Executor Agent to complete complex GUI navigation tasks. Given a task description, the current screenshot, and the action history from the Executor Agent, your goal is to provide a clear and precise fine-grained instruction for the Executor Agent to help accomplish the task.

Tools
You need to interact with the Executor Agent by making a tool call:
<tools>
{"type": "function", "function": {"name": "executor_agent", "description": "an Executor Agent capable of executing fine-grained instruction", "parameters": {"type": "object", "properties": {"instruction": {"type": "string", "description": "A clear and precise fine-grained instruction for the executor agent"}}, "required": ["instruction"]}, "strict": false}}
</tools>

Return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

Note
- You should first outline the overall task flow and clarify your next intention. Then, generate a fine-grained, precise, and unambiguous instruction that will guide the Executor Agent to execute one of its available actions: {action_space}.
- Please keep your reasoning within <think> </think> tags, and then output the fine-grained instruction as a tool call in the following format:
<think>...</think><tool_call>...</tool_call>

User Instruction
{user_high_level_instruction}

Interactor

<img>current_screenshot.png</img>

You are a reasoning GUI Executor Agent. Given the attached UI screenshot and the instruction: "{planner_output_instruction}", please determine the next action to fulfill the instruction.

Action Space
{action_space}

Note
- Please keep your reasoning in <think> </think> tags brief and focused. Output the final action in <answer> </answer> tags:
<think>...</think><answer>...</answer>

Figure 8: Prompt design for the dual-agent framework in the GUI domain.

Teacher
Two LLM agents (Teacher -> Student) collaborate step-by-step to solve math problems. You are the Teacher: Provide only a concise problem-solving strategy without revealing the full solution, and guide the Student to complete the problem. question: {question}
Student
Two LLM agents (Teacher -> Student) collaborate step-by-step to solve math problems. You are the Student: Follow the original problem and the Teacher's guidance to carry out the necessary operations, and present the final answer within \\boxed{}. question: {question} <teacher_response>{teacher_response}</teacher_response>

Figure 9: Prompt design for the dual-agent framework in the mathematics domain.

E EXAMPLES

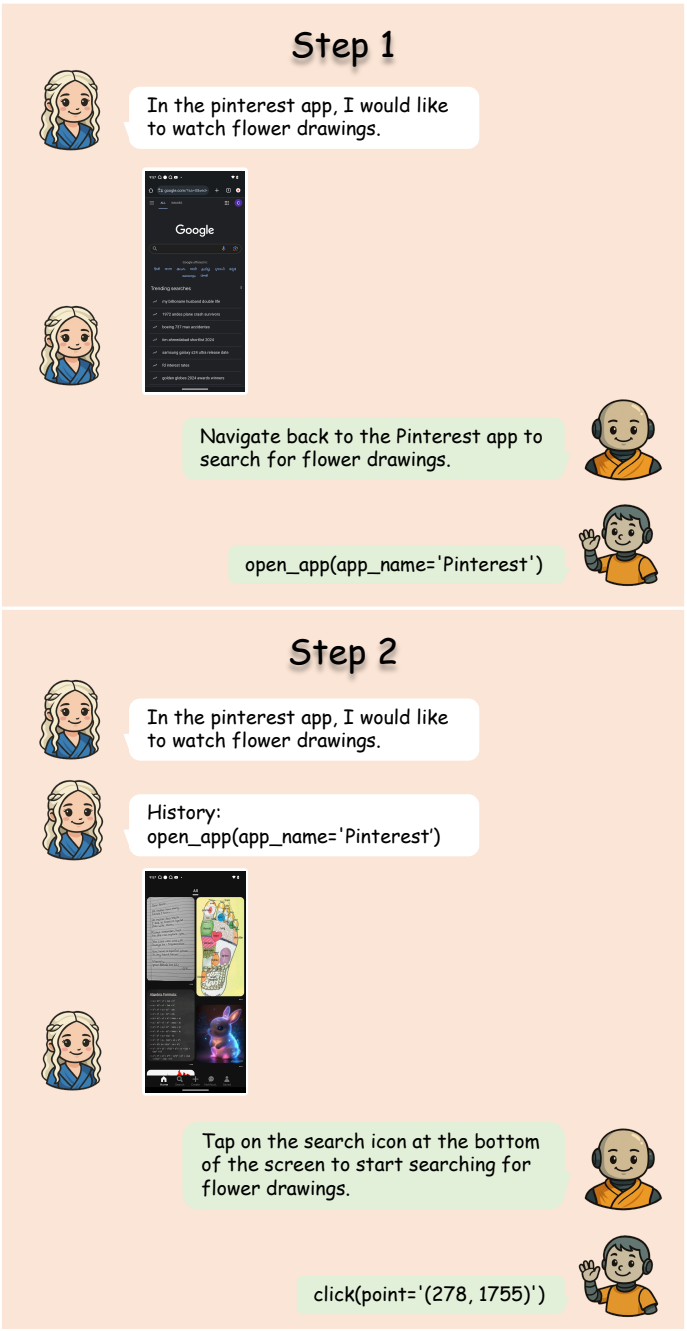


Figure 10: Example of GUI agents collaboratively performing a mobile GUI control task (Part 1 of 3). At each step, the system determines the next action based on the user instruction, the current screenshot, and the action history, iterating until the task is completed.

Figure 11: Example of GUI agents collaboratively performing a mobile GUI control task (Part 2 of 3).

Figure 12: Example of GUI agents collaboratively performing a mobile GUI control task (Part 3 of 3).